Software attached to the paper "A Bayesian approach to Mendelian randomisation with multiple pleiotropic variants", by C. Berzuini, H. Guo, S. Burgess and L. Bernardinelli

April 17, 2018

The software described in this document is maintained by Carlo Berzuini, Centre for Biostatistics, The University of Manchester, Manchester, UK:

carlo.berzuini@manchester.ac.uk

Use of this software requires an installation of the statistical package `R` and of the probabilistic modelling language `STAN`. For links to up-to-date code, examples, manuals, bug reports, feature requests, and everything else related to `STAN`, see the `STAN` home page: http://mc-stan.org/. The `R` interface to `STAN` is `Rstan`. The `Rstan` home page is http://mc-stan.org/rstan.html. One of `STAN`'s virtues is support of variational inference. This is an approximate Bayesian inference technique (Jordan et al., 1999; Wainwright and Jordan, 2008) that provides an approximation nof the posterior distribution, $P(unknown\ parameters \mid known\ quantities)$, for an efficient bounding of the search space.

The user may wish to implement our method with the aid of the `R` function and of the `STAN` program described in the following two sections.

STAN program for MCMC estimation of the proposed Mendelian randomisation method

The function below specifies model (3.1-3.3) in a format suitable for input to `STAN`. More specifically, the model is specified through a conditional probability function $P(\psi \mid X, Y, U; Z)$, where ψ denotes a vector of modeled

(random) unknown model parameters (or latent variables, or missing data), X is the vector of modeled observed exposures, Y is the vector of modeled observed outcome values, Z is the unmodeled observed matrix of instrumental variable values, and (N, J) its dimensions. The symbols used in the program below are consistent with those used in the paper. The matrix Z and its dimensions must reside in the `R` workspace.

The program begins by specifying a function named VECSQRT() that calculates the element-wise square root of an input vector.

The DATA BLOCK section of the program contains declarations of the observed variables in the problem (data). These are:

N is the number of sample units. Constrained (> 0) integer.

J is the number of instruments. Constrained (> 0) integer.

Z is an $N \times J$ real-valued matrix of instrumental information, with z_{ik} representing the value of instrument k in unit i. Unconstrained.

X is a real-valued N-vector of exposure values, with x_i representing the value of the exposure for unit i. Unconstrained.

Y is a real-valued N-vector of outcome values, with y_i representing the value of the outcome for unit i, expressed on an interval scale. Unconstrained.

The PARAMETERS BLOCK section of the program contains declarations of the unobserved model parameters and hyperparameters..

The TRANSFORMED PARAMETERS section of the program specifies parameter vector β to follow a horseshoe independence prior. β is a transformed parameter in the sense that it is a stochastic quantity defined in terms of a deterministic function of other parameters in the model.

The first three lines of the MODEL BLOCK in the program define the conditional likelihood of the data, as a function of the model parameters. The remaining lines of the block specify the prior distribution of the model parameters and hyperparameters. Most parameters and hyperparameters in our model have a uniform, "ignorance", prior distribution, that requires no input from the user. For such parameters, in a `STAN` program, we can simply omit the specification of a prior. A non-ignorance prior specification is only given for vewctor α. The strength α_k of each kth instrument is specified to follow a $N(\mu_\alpha, \sigma_\alpha)$ prior whose hyperparameters, μ_α and σ_α, are taken to follow a uniform prior distribution. Of special inferential interest among the parameters in this model is the causal effect of X on Y, denoted as θ.

```
> stanmodelcode <-'
+ /* lg_t.stan */
+ functions {
```

```
+ // Vector square root
+ vector vecsqrt(vector x) {
+         vector[dims(x)[1]] res;
+         for (m in 1:dims(x)[1]){
+                 res[m] = sqrt(x[m]);
+         }
+ return res; }
+ }
+ data {
+   int<lower=0> N;
+   int<lower=0> J;
+   matrix[N,J] Z;
+   vector[N] X;
+   vector[N] Y;
+ }
+ parameters {
+   real <lower=0> sigmax;
+   real <lower=0> sigmay;
+   real <lower=0> sigmaalpha;
+   real<lower=0> r1_global;
+   real<lower=0> r2_global;
+   real mualpha;
+   real omegax;
+   real omegay;
+   real deltax;
+   real deltay;
+   real theta;
+   vector[N] u;
+   vector[J] z;
+   vector<lower=0>[J] r1_local;
+   vector<lower=0>[J] r2_local;
+   vector[J] alpha;
+ }
+ transformed parameters {
+    real<lower=0> tau;
+    vector<lower=0> [J] lambda;
+    vector[J] beta;
+    tau      = r1_global * sqrt(r2_global);
+    lambda          = r1_local .* vecsqrt(r2_local);
+    beta            =  z .* lambda*tau;
+ }
+ model {
+   X         ~ normal(omegax+Z*alpha+u*deltax, sigmax);
+   Y         ~ normal(omegay+Z*beta+X*theta+u*deltay, sigmay);
+   u         ~ normal(0,1);
+
```

```
+   for(k in 1:J){
+     alpha[k] ~ normal(mualpha, sigmaalpha);
+   }
+ // Constructing the prior for the lambda vector
+     z ~ normal(0, 1);
+     r1_local ~ normal(0.0, 1.0);
+     r2_local ~ inv_gamma(0.5, 0.5);
+ // Constructing the prior for tau
+     r1_global ~ normal(0.0, 1.0);
+     r2_global ~ inv_gamma(0.5, 0.5);
+     }
+ '
```

GENERATE function

AIM: This R function generates a simulated dataset. Can be used within a simulation loop.

OUTPUT: conditional on an input $N \times J$ matrix of values for the instrumental variables, Z, and on a value THETA for the causal effect, this function produces an output list with containing realisations of X and Y. The output list also contains Z.

```
> generate <- function(N,J,THETA,Z,deltamin,deltamax,deltaxsd,
+                      deltaysd,sigmaxmin,sigmaxmax,sigmaymin,
+                      sigmaymax,fracnonzeroAlpha,alphamean,
+                      alphasd,omegaxmean,omegaymean,omegaxsd,
+                      omegaysd,pleiotropy,pleiomin, pleiomax,
+                      fractionpleiotropic,pleiosd){
+ U                                  = array(rnorm(N, mean=0, sd=1), dim = N)
+ MEANDELTA = runif(1, deltamin, deltamax)
+ DELTAX                 = rnorm(1, mean= MEANDELTA, sd=deltaxsd)
+ DELTAY                 = rnorm(1, mean= MEANDELTA, sd=deltaysd)
+ nonzeroAlpha  = round(fracnonzeroAlpha*J)
+ ALPHA = array(rep(0,J),dim=J)
+ ALPHA[sample(c(1:J),size=nonzeroAlpha,prob=rep(1,J),replace=F)] =
+           rnorm(nonzeroAlpha, mean= alphamean, sd=alphasd)
+ OMEGAX                 = rnorm(1,mean=omegaxmean,sd=omegaxsd)
+ OMEGAY                 = rnorm(1,mean=omegaymean,sd=omegaysd)
+ X                                  = array(Z%*%ALPHA, dim = N)
+ X                                  = X+OMEGAX+U*DELTAX
+ SIGMAX                = runif(1,sigmaxmin,sigmaxmax)
+ EPS                          = array(rnorm(N, mean=0, sd=SIGMAX), dim=N)
+ X                            = X+EPS
+ MEANPLEIO = 0
+ PLEIO                 = runif(1,pleiomin,pleiomax)
```

```
+ if(pleiotropy == "POSITIVE")MEANPLEIO = PLEIO
+ if(pleiotropy == "NEGATIVE")MEANPLEIO = -PLEIO
+ BETA = array(rep(0,J), dim=J)
+ HOWMANYPLEIOTROPIC = round(fractionpleiotropic*J)
+ BETA[sample(c(1:J),size=HOWMANYPLEIOTROPIC,prob=rep(1,J),replace=F)] =
+           rnorm(HOWMANYPLEIOTROPIC,mean=MEANPLEIO,sd=pleiosd)
+ SIGMAY                  = runif(1,sigmaymin,sigmaymax)
+ EPSILON          = array(rnorm(N, mean=0, sd=SIGMAY), dim=c(N,1))
+ esposizione          = array(X*THETA, dim=c(N,1))
+ confo                 = array(U*DELTAY, dim= c(N,1))
+ YBUF                  = OMEGAY+Z%*%BETA+esposizione+confo
+ YBUF                  = YBUF+EPSILON
+ Y                          = array(YBUF, dim=N)
+ dataset= list(X=X,Y=Y,Z=Z,U=U)
+ dataset
+ }
```

HOW IS THE OUTPUT OF THIS FUNCTION GENERATED: realisations of δ_X and δ_Y are independently drawn from $N($ `MEANDELTA, deltaxsd` $)$ and from $N($ `MEANDELTA, deltaysd` $)$, respectively, where `MEANDELTA` is from a rectangular distribution on (`deltamin, deltamax`). Realisations of `SIGMAX` and `SIGMAY` are drawn from rectangular distributions on (`sigmaxmin, sigmaxmax`) and (`sigmaymin, sigmaymax`), respectively. The elements of `ALPHA` are set to 0, except that a fraction `fracnonzeroAlpha` of them are randomly selected and independently drawn from $N(mean =$ `alphamean` $, sd =$ `alphasd`). Parameters `OMEGAX` and `OMEGAY` are randomly drawn from $N(mean =$ `omegaxmean` $, sd =$ `omegaxsd`) and $N(mean =$ `omegaymean` $, sd =$ `omegaysd`), respectively. Elements of `BETA` are set to zero, except for a fraction `fractionpleiotropic` of them, which are randomly selected and independently drawn from $N(mean =$ `MEANPLEIO` $, sd =$ `pleiosd`), with `MEANPLEIO` from a rectangular distribution on (`pleiomin, pleiomax`). The elements of `BETA` are then multiplied by $(1, -1, 0)$ according as the input argument `pleiotropy` is "POSITIVE", "NEGATIVE" or "BALANCED". Conditional on all the above parameters, on the value `THETA` for the causal effect and on Z, the function generates the values of X and Y in accord with Equations (3.1-3.3) of the paper and returns them bundled with Z in the output list. The above described, rather involved, parameter setting procedure has been designed to allow the user to simulate a large variety of data generating processes.

Elementary Application Example

We set the `R` environment by loading two required libraries, which the user can install from CRAN:

```
> library(rstan)
> library(MendelianRandomization)
```

We define a useful formatting function:

```
> f = function(X1)gsub("0\\.","\\.", X1)
```

We read the data, which in this illustrative example consist of an $N \times J$ matrix denoted as Z, with $N = 500$ and $J = 60$, where z_{ij} is the value of instrument j in individual i. The components of Z arise with values in $(0, 1, 2)$, but have been centered prior to analysis.

```
> load(file="Z.rda")
> J = ncol(Z)
> N = nrow(Z)
```

Unlike existing frequentist Mendelian randomisation methods, ours has been designed with both independent and correlated instruments in mind. Let us now calculate the average instrument-instrument correlation in our data:

```
> signif(mean(cor(Z)),3)
```

```
[1] 0.329
```

For purposes of illustration, we are now going to simulate a dataset, and then apply to the generated dataset the Mendelian randomisation method proposed in the paper. In simulating the data, we assume that the causal effect of X on Y is 0.35:

```
> THETA = 0.35
```

The dataset is simulated with the aid of the previously described GENERATE function:

```
> simulated = generate(N,J,THETA,Z,
+ deltamin      =-0.2,
+ deltamax      =-0.1,
+ deltaxsd      =0.02,
+ deltaysd      =0.02,
+ sigmaxmin     =0.05,
+ sigmaxmax     = 0.15,
+ sigmaymin    = 0.2,
+ sigmaymax                = 0.4,
+ fracnonzeroAlpha = 0.3,
+ alphamean          = 1.0, #-0.07
+ alphasd                  = 0.2,
+ omegaxmean    = 3.3,
+ omegaymean    = 0.9,
```

```
+ omegaxsd      = 0.2,
+ omegaysd      = 0.2,
+ pleiotropy = "POSITIVE",
+ pleiomin = -0.1,#0.006,
+ pleiomax   = 0.3,#0.25, #0.36, #0.012,
+ fractionpleiotropic = 0.49,
+ pleiosd = 0.1#0.05
+ )
```

The generated dataset is re-expressed in terms of an exposure vector X, an outcome vector Y, a confounder vector U and the matrix of instrumental data, Z:

```
> X = array(simulated$X, dim = N)
> Y = array(simulated$Y, dim = N)
> Z = array(simulated$Z, dim = c(N,J))
> U = array(simulated$U, dim = N)
```

We now calculate a J-vector β_X, that containins the estimated unconditional slope of the (univariate) least squares regression of X on each of the J instrumental variables. Also calculated is a J-vector β_Y, that contains the estimated unconditional slope of the least squares regression of Y on each instrumental variable. We also calculate the J-vectors `sebetaX` and `sebetaY`, containing the standard errors for each element of β_X and β_Y, respectively:

```
> betaX              <- array(NA,    dim=J)
> betaY              <- array(NA,    dim=J)
> sebetaY            <- array(NA,    dim=J)
> sebetaX            <- array(NA,    dim=J)
> for(isnp in 1:J){
+    regX                              <- lm(X ~ Z[,isnp])
+    regY                              <- lm(Y ~ Z[,isnp])
+    betaX[isnp]                 <- summary(regX)$coefficients[2,1]
+    sebetaX[isnp]         <- summary(regX)$coefficients[2,2]
+    betaY[isnp]                 <- summary(regY)$coefficients[2,1]
+    sebetaY[isnp]         <- summary(regY)$coefficients[2,2]
+ }
```

The β_X and β_Y (and their standard errors) are used to calculate a preliminary estimate of the causal effect, called `thetamedianestimate`, via weighted median estimator:

```
> oggetto           = mr_input(bx = as.numeric(betaX),
+                         bxse = as.numeric(sebetaX),
+                         by   = as.numeric(betaY),
+                         byse = as.numeric(sebetaY),
+                         correlation                = cor(Z),
```

```
+       exposure = "X ", outcome = "Y",
+                   snps = colnames(Z))
> risultato = mr_allmethods(oggetto, method = "all")
> thetamedianestimate         = risultato$Values[2,2]
> thetamedianestimate

[1] 0.3872661
```

The next step consists of setting the initial values of some model parameters, for use in the definition of the starting points of the Markov chains. The preliminary estimate of the causal effect is included as initial value for the causal effect parameter:

```
> init_list                          = list(c1=list(theta=thetamedianestimate,
+                                                   beta=rep(0,J),alpha=betaX,deltax=0,
+                                                               deltay=0,u=rep(0,N)))
```

600 iteration of a Markov chain with equilibrium distribution equal to the posterior distribution of our model are run. The first 300 iterations are regarded as the "warming" stage of the Markov chain, after which the chain is assumed to be in equilibrium. The parameter values generated during the remaining 300 iterations are used for inference, such as for calculating posterior means and credible intervals for the quantities f inferential interest or for functions of them. In this particular example, for simplicity, we run a single chain:

```
> fit       <- stan(model_code=stanmodelcode, iter=1000,
+                chains=1, init=init_list, verbose=F)
```

We shall now analyse the generated posterior samples. First, we extract the samples of parameter `theta`, that represents the causal effect of inferential interest:

```
> theta                                  = extract(fit,pars='theta',permuted=FALSE)
> motheta                          = monitor(theta)

Inference for the input samples (1 chains: each with iter=500; warmup=250):

      mean se_mean sd 2.5% 25% 50% 75% 97.5% n_eff Rhat
theta  0.4       0  0  0.3 0.3 0.4 0.4   0.4    48    1

For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

We previously set the "true" value of `theta` to be equal to 0.35.Its output 95% credible interval is:

```
> cat("\n (", motheta[4],",",
+ motheta[8],")",sep="")

 (0.3374125,0.3787554)
```

Its posterior mean is:

```
> motheta[6]

[1] 0.3577514
```