simulation

Xiyue Zhang

2023-02-18

Stan Code

We pass the code as a string.

```
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

The function is to generate the data based on our simulation rules.

```
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
}
```

library loaded

```
library(StanHeaders)
```

```
## Warning: package 'StanHeaders' was built under R version 4.1.3
```

```
library(ggplot2)
library(rstan)
```

```
## rstan (Version 2.21.8, GitRev: 2e1f913d3ca3)

## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)

## Do not specify '-march=native' in 'LOCAL_CPPFLAGS' or a Makevars file
```

```
library(MendelianRandomization)
```

```
## Warning: package 'MendelianRandomization' was built under R version 4.1.3

## Warning in .recacheSubclasses(def@className, def, env): undefined subclass
## "packedMatrix" of class "replValueSp"; definition not updated

## Warning in .recacheSubclasses(def@className, def, env): undefined subclass
## "packedMatrix" of class "mMatrix"; definition not updated
```

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

load matrix

The author did not upload the data, so we just generate a matrix with same dimension to test the code. Note: why a "random" matrix can still justify the result?

```
Z<-matrix(nrow = 500,ncol=60)
n<-60
p<-runif(30000)
for (i in 1:500){
  for(j in 1:60){
    Z[i,j]<-rbinom(1,2,p[i])
  }
}
J = ncol(Z)
N = nrow(Z)
```

The average SNP-SNP correlation in this particular set of data.

```
mean(cor(Z))
```

```
## [1] 0.5225524
```

Experiment

use the experiment code provided

```
THETA = 0.35
```

```
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
)
```

```
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)
```

```
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]
}
```

```
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] 0.4276356
```

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

```
dat = list(N=N, J=J,Z=Z,X=X,Y=Y)
```

Stan

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

```
##
## SAMPLING FOR MODEL '41a98f6109f52c148ad9631ea340aeee' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 1: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 1: Iteration: 200 / 1000 [ 20%]  (Warmup)
```

```
## Chain 1: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 1: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 1: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 1: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 1: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 1: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 1: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 1: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 1: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 48.714 seconds (Warm-up)
## Chain 1:                73.992 seconds (Sampling)
## Chain 1:                122.706 seconds (Total)
## Chain 1:
```

```
## Warning: There were 5 divergent transitions after warmup. See
## https://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.
```

```
## Warning: There were 441 transitions after warmup that exceeded the maximum treedepth. Increase max_t
## https://mc-stan.org/misc/warnings.html#maximum-treedepth-exceeded
```

```
## Warning: There were 1 chains where the estimated Bayesian Fraction of Missing Information was low. S
## https://mc-stan.org/misc/warnings.html#bfmi-low
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
## Warning: The largest R-hat is 1.11, indicating chains have not mixed.
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#r-hat
```

```
## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians may be
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#bulk-ess
```

```
## Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and tail quant
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#tail-ess
```

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

```
motheta = monitor(theta)
```

```
## Inference for the input samples (1 chains: each with iter = 500; warmup = 250):
##
##        Q5 Q50 Q95 Mean SD  Rhat Bulk_ESS Tail_ESS
## theta 0.3 0.4 0.4  0.4  0  1.01      124      173
##
## For each parameter, Bulk_ESS and Tail_ESS are crude measures of
## effective sample size for bulk and tail quantities respectively (an ESS > 100
## per chain is considered good), and Rhat is the potential scale reduction
## factor on rank normalized split chains (at convergence, Rhat <= 1.05).
```

```
motheta
```

```
##            mean      se_mean          sd      2.5%       25%       50%       75%
## theta 0.3561174 0.0008676348 0.009529923 0.3376064 0.349654 0.3562361 0.3630779
##           97.5% n_eff     Rhat valid        Q5       Q50       Q95   MCSE_Q2.5
## theta 0.3733635   114 1.011577     1 0.3406359 0.3562361 0.3711463 0.001385454
##           MCSE_Q25    MCSE_Q50     MCSE_Q75  MCSE_Q97.5      MCSE_SD Bulk_ESS
## theta 0.0005365998 0.001004845 0.0009889806 0.001577531 0.0006168772      124
##       Tail_ESS
## theta      173
```

95% confidence interval

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

```
##
##  (0.3376064,0.3733635)
```

Its posterior mean is:

```
motheta[,6]
```

```
##     theta
## 0.3562361
```