

Chapter 1

Examples

In the following section the Bayesian LMG implementation is presented on two examples. The first examples simulates data, the second examples uses real data.

1.1 Simulated Data

Lets assume a simple model:

$$Y_i \sim \mathcal{N}(\beta_0 + x_1\beta_1 + x_2\beta_2 + x_3\beta_3 + x_4\beta_4, \sigma^2), \quad (1.1)$$

$$\beta_1 = 0.5, \beta_2 = 1, \beta_3 = 2, \beta_4 = 0, \sigma^2 = 1 \quad (1.2)$$

$$\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3, \mathbf{X}_4 \sim \mathcal{N}(0, 1) \quad (1.3)$$

The values of the four predictors are sampled from a standard normal distribution. These values are then multiplied by the regression coefficients to obtain the dependent variable. A standard normal distributed error is added. Fifty observations were sampled.

The following Code was used to simulate the data :

```
x1 <- rnorm(50, 0, 1); x2 <- rnorm(50, 0, 1)
x3 <- rnorm(50, 0, 1); x4 <- rnorm(50, 0, 1)
#b1 <- 0.5; b2 <- 1; b3 <- 2; b4 <- 0
b1 <- 1; b2 <- 1; b3 <- 1; b4 <- 1

y <- b1*x1 + x2*b2 + b3*x3 + b4*x4 + rnorm(50, 0, 1)

df <- data.frame(y = y, x1 = x1, x2 = x2, x3 = x3, x4 = x4)
```

The model is fitted using the rstanarm package with the default priors for the regression and σ^2 parameter. For computational reasons a small burning period of 1000 and a sample size of 1000 were chosen. For each posterior sample of the parameters the R^2 value is calculated. The R^2 of the submodels is then calculated by the conditional variance formula for each posterior sample.

```
post2 <- stan_glm(y ~ 1 + x1 + x2 + x3 + x4,
                    data = df,
                    chains = 2, cores = 1)

##  
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 1).  
##  
## Gradient evaluation took 0.000137 seconds  
## 1000 transitions using 10 leapfrog steps per transition would take 1.37 seconds.  
## Adjust your expectations accordingly!  
##  
##  
## Iteration: 1 / 2000 [  0%] (Warmup)  
## Iteration: 200 / 2000 [ 10%] (Warmup)  
## Iteration: 400 / 2000 [ 20%] (Warmup)  
## Iteration: 600 / 2000 [ 30%] (Warmup)  
## Iteration: 800 / 2000 [ 40%] (Warmup)  
## Iteration: 1000 / 2000 [ 50%] (Warmup)  
## Iteration: 1001 / 2000 [ 50%] (Sampling)  
## Iteration: 1200 / 2000 [ 60%] (Sampling)  
## Iteration: 1400 / 2000 [ 70%] (Sampling)  
## Iteration: 1600 / 2000 [ 80%] (Sampling)  
## Iteration: 1800 / 2000 [ 90%] (Sampling)  
## Iteration: 2000 / 2000 [100%] (Sampling)  
##  
## Elapsed Time: 0.07019 seconds (Warm-up)  
##                      0.064703 seconds (Sampling)  
##                      0.134893 seconds (Total)  
##  
##  
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 2).  
##  
## Gradient evaluation took 1.5e-05 seconds  
## 1000 transitions using 10 leapfrog steps per transition would take 0.15 seconds.  
## Adjust your expectations accordingly!  
##  
##  
## Iteration: 1 / 2000 [  0%] (Warmup)  
## Iteration: 200 / 2000 [ 10%] (Warmup)  
## Iteration: 400 / 2000 [ 20%] (Warmup)  
## Iteration: 600 / 2000 [ 30%] (Warmup)  
## Iteration: 800 / 2000 [ 40%] (Warmup)  
## Iteration: 1000 / 2000 [ 50%] (Warmup)
```

```

## Iteration: 1001 / 2000 [ 50%] (Sampling)
## Iteration: 1200 / 2000 [ 60%] (Sampling)
## Iteration: 1400 / 2000 [ 70%] (Sampling)
## Iteration: 1600 / 2000 [ 80%] (Sampling)
## Iteration: 1800 / 2000 [ 90%] (Sampling)
## Iteration: 2000 / 2000 [100%] (Sampling)
##
## Elapsed Time: 0.078599 seconds (Warm-up)
##                 0.072947 seconds (Sampling)
##                 0.151546 seconds (Total)

#posterior sample
post.sample <- as.matrix(post2)

#example of the first 10 posterior samples
post.sample[1:10,]

##           parameters
## iterations (Intercept)      x1      x2      x3      x4      sigma
## [1,]  0.25146161 0.8120795 0.9876217 1.055278 0.7568951 1.014648
## [2,] -0.25162203 1.1284339 0.8604222 1.770105 1.2114690 1.131450
## [3,]  0.20280485 1.0829865 1.0009234 1.576368 1.1886219 1.072287
## [4,]  0.10389384 1.0563978 1.0562895 1.579383 1.1433637 1.004666
## [5,] -0.19671469 0.9334791 0.8991408 1.039873 0.7894007 1.078298
## [6,]  0.02623394 1.0352149 1.2775867 1.740727 1.0874321 1.143698
## [7,] -0.14006697 1.0400488 0.5799490 1.257080 1.0452126 1.023704
## [8,]  0.06589059 0.9446801 0.7583340 1.569386 1.2945990 1.144864
## [9,] -0.27616382 1.0443262 1.0203259 1.404316 0.6486706 0.924302
## [10,] -0.02227921 1.2252321 1.1855809 1.204604 0.7119914 1.063315

#no need for the intercept, last parameter is sigma
post.sample <- post.sample[,-1]

#data frame with all submodels
df.rtwos <- rtwos(df[,2:5], post.sample)

df.rtwos[,1:5]

##           X1      X2      X3      X4      X5
## none      0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## x1       0.1467998 0.1846805 0.1791542 0.1712197 0.1913004
## x2       0.2483680 0.1188297 0.1708345 0.1911564 0.1963248
## x3       0.2204477 0.3571245 0.3020489 0.3119271 0.2044250

```

```
## x4      0.1802315 0.1943277 0.2196437 0.2137950 0.1841791
## x1 x2  0.4417630 0.3396232 0.3925358 0.4063801 0.4347560
## x1 x3  0.3638336 0.5369329 0.4767891 0.4787622 0.3919718
## x1 x4  0.2827283 0.3273623 0.3447681 0.3329124 0.3243058
## x2 x3  0.4749532 0.4813682 0.4788446 0.5094907 0.4060045
## x2 x4  0.3522687 0.2586075 0.3204942 0.3317824 0.3116590
## x3 x4  0.4487135 0.6151917 0.5838086 0.5880307 0.4353470
## x1 x2 x3 0.6649319 0.6975455 0.6962816 0.7204433 0.6407866
## x1 x2 x4 0.5013984 0.4242556 0.4849256 0.4923512 0.4985243
## x1 x3 x4 0.5425040 0.7358258 0.6977385 0.6960826 0.5656068
## x2 x3 x4 0.6160094 0.6758613 0.6804402 0.7013900 0.5588807
## all     0.7539647 0.8267264 0.8311791 0.8482520 0.7336243
```

After the R^2 for each posterior sample and their corresponding submodels is calculated, the package hier.part is used to calculate the LMG value for each posterior sample.

```
# prepare data frame for LMG values

LMG.Vals<-matrix(0, 4, dim(df.rtwos)[2])

for(i in 1:dim(df.rtwos)[2]){

  gofn<-df.rtwos[,i]

  obj.Gelman<-partition(gofn, pcan = 4, var.names = names(df[,2:5]))

  LMG.Vals[,i]=obj.Gelman$IJ[,1]
}

# posterior LMG distribution of each variable
quantile(LMG.Vals[1,], c(0.025, 0.5, 0.975))

##          2.5%      50%      97.5%
## 0.09166948 0.17637113 0.27580559

quantile(LMG.Vals[2,], c(0.025, 0.5, 0.975))

##          2.5%      50%      97.5%
## 0.08274643 0.16225299 0.25941496

quantile(LMG.Vals[3,], c(0.025, 0.5, 0.975))

##          2.5%      50%      97.5%
## 0.1941646 0.3021694 0.4103921
```

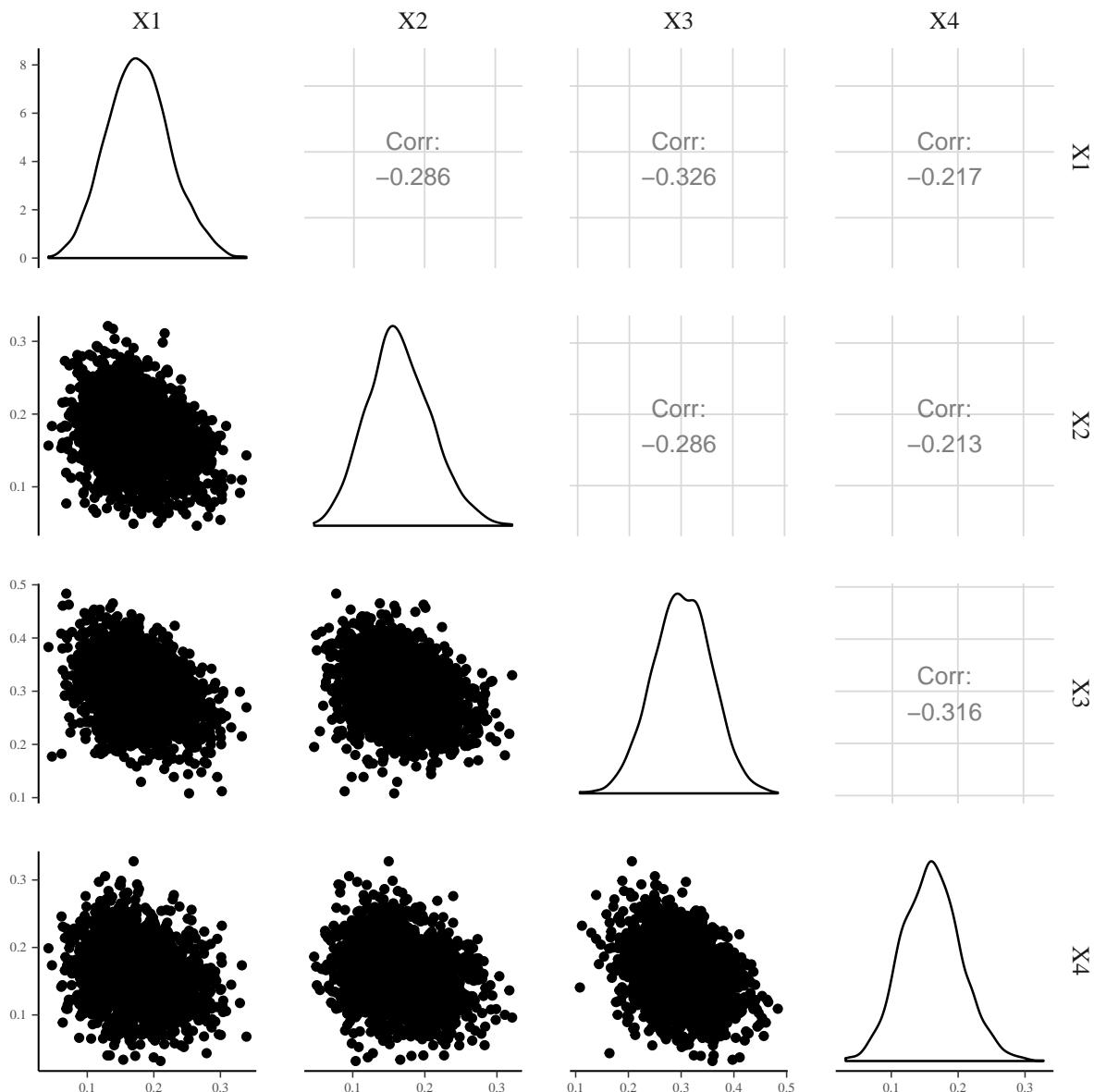
```
quantile(LMG.Vals[4,], c(0.025, 0.5, 0.975))

##      2.5%      50%    97.5%
## 0.07724791 0.15951723 0.24966424

# some example how it could be displayed
dat <- data.frame(t(LMG.Vals))

pairs.chart <- ggpairs(dat, lower = list(list(combo = "facetdensity")),
upper = list(list(combo = "jitter")))

pairs.chart
```



```
# Comparison to relaimpo package

fit <- lm(y~., data=df)
```

```

##### compare to relimp package

run<-boot.relimp(fit, fixed=TRUE)

booteval.relimp(run, bty = "perc", level = 0.95,
                 sort = FALSE, norank = TRUE, nodiff = TRUE,
                 typesel = c("lmg"))

## Response variable: y
## Total response variance: 5.552254
## Analysis based on 50 observations
##
## 4 Regressors:
## x1 x2 x3 x4
## Proportion of variance explained by model: 82.41%
## Metrics are not normalized (rela=FALSE).
##
## Relative importance metrics:
##
##          lmg
## x1 0.1811296
## x2 0.1666997
## x3 0.3132778
## x4 0.1629547
##
## Average coefficients for different model sizes:
##
##          1X      2Xs      3Xs      4Xs
## x1 1.074823 1.0625800 1.0486902 1.0286499
## x2 1.013182 0.9951801 0.9758422 0.9506321
## x3 1.293776 1.3391547 1.3728345 1.3889366
## x4 1.250949 1.1850119 1.0992265 0.9949324
##
## Confidence interval information ( 1000 bootstrap replicates, bty= perc ):
## Relative Contributions with confidence intervals:
##
##          Lower   Upper
##          percentage 0.95  0.95
## x1.lmg 0.1811    0.1009 0.2782
## x2.lmg 0.1667    0.0922 0.2607
## x3.lmg 0.3133    0.2169 0.4317
## x4.lmg 0.1630    0.0949 0.2506

```

```
##  
## CAUTION: Bootstrap confidence intervals can be somewhat liberal.  
## NOTE: X-matrix has been considered as fixed for bootstrapping.
```

Using the default uninformative priors, the LMG distributions obtained from the Bayesian framework are very similar to the bootstrap confidence intervals of the LMG estimates obtained from the relaimpo package. In both cases fixed regressors are assumed. In the example above the predictors were sampled from a normal distribution. It would therefore be more reasonable to assume stochastic predictors. As noted in ... under the assumption of weak exogeneity and conditional independence the posterior distributions of the regression parameters β are valid for fixed and stochastic predictors. Inference about the covariance matrix can be seen as an independent problem. G recommends in most cases to use the non fixed regressor option when calculating bootstrap confidence intervals. The confidence intervals will then in general be a bit larger. If we want to include this uncertainty in the Bayesian framework, we would need some ideas about the distribution of the predictor variables \mathbf{X} . It is then possible to obtain posterior distributions of their corresponding covariance matrix. As a practical solution nonparametric bootstrap may be used to include the uncertainty of the covariance matrix. The following code includes the uncertainty of the stochastic predictors.

```
#Code assuming stochastic predictors  
run.stochastic<-boot.relimp(fit, fixed=FALSE)  
  
booteval.relimp(run.stochastic, bty = "perc", level = 0.95,  
                 sort = FALSE, norank = TRUE, nodiff = TRUE,  
                 typesel = c("lmg"))  
  
## Response variable: y  
## Total response variance: 5.552254  
## Analysis based on 50 observations  
##  
## 4 Regressors:  
## x1 x2 x3 x4  
## Proportion of variance explained by model: 82.41%  
## Metrics are not normalized (rela=FALSE).  
##  
## Relative importance metrics:  
##  
##          lmg  
## x1 0.1811296  
## x2 0.1666997  
## x3 0.3132778  
## x4 0.1629547  
##  
## Average coefficients for different model sizes:
```

```

##          1X      2Xs      3Xs      4Xs
## x1 1.074823 1.0625800 1.0486902 1.0286499
## x2 1.013182 0.9951801 0.9758422 0.9506321
## x3 1.293776 1.3391547 1.3728345 1.3889366
## x4 1.250949 1.1850119 1.0992265 0.9949324
##
##
## Confidence interval information ( 1000 bootstrap replicates, bty= perc ):
## Relative Contributions with confidence intervals:
##
##          Lower   Upper
##      percentage 0.95  0.95
## x1.lmg 0.1811    0.0784 0.3107
## x2.lmg 0.1667    0.0579 0.2927
## x3.lmg 0.3133    0.1648 0.4536
## x4.lmg 0.1630    0.0567 0.3131
##
## CAUTION: Bootstrap confidence intervals can be somewhat liberal.

#-----
```



```

df.rtwos.boot <-rtwos.boot(df[,2:5], post.sample, 10)

n.boot = 10

LMG.Vals.boot<-array(0, c(4,dim(df.rtwos.boot)[2], n.boot))

for (b in 1:n.boot){

  for(i in 1:dim(df.rtwos.boot)[2]){

    gofn<-df.rtwos.boot[,i,b]

    obj.Gelman<-partition(gofn, pcan = 4, var.names = names(df[,2:5]))

    LMG.Vals.boot[,i, b]=obj.Gelman$IJ[,1]
  }

}

quantile(c(LMG.Vals.boot[1,1:1000,]), c(0.025, 0.5, 0.975))
```

```

##      2.5%      50%     97.5%
## 0.0563607 0.1407858 0.2943099

quantile(c(LMG.Vals.boot[2,1:1000,]), c(0.025, 0.5, 0.975))

##      2.5%      50%     97.5%
## 0.0758204 0.1941894 0.3464876

quantile(c(LMG.Vals.boot[3,1:1000,]), c(0.025, 0.5, 0.975))

##      2.5%      50%     97.5%
## 0.1151979 0.2832940 0.4079618

quantile(c(LMG.Vals.boot[4,1:1000,]), c(0.025, 0.5, 0.975))

##      2.5%      50%     97.5%
## 0.04244554 0.16971870 0.31231758

#very similar values as in the confidence intervals for stochastic predictors

#what if we have prior knowledge
my_prior <- normal(location = c(1, 1, 1, 1), scale = c(0.01, 0.01, 0.01, 0.01), autoscale = FALSE)

post2 <- stan_glm(y ~ 1 + x1 + x2 + x3 + x4,
                    data = df, prior = my_prior,
                    chains = 2, cores = 1)

##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 1).
##
## Gradient evaluation took 0.000117 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 1.17 seconds.
## Adjust your expectations accordingly!
##
##
## Iteration:    1 / 2000 [  0%]  (Warmup)
## Iteration:   200 / 2000 [ 10%]  (Warmup)
## Iteration:   400 / 2000 [ 20%]  (Warmup)
## Iteration:   600 / 2000 [ 30%]  (Warmup)
## Iteration:   800 / 2000 [ 40%]  (Warmup)
## Iteration:  1000 / 2000 [ 50%]  (Warmup)
## Iteration:  1001 / 2000 [ 50%]  (Sampling)
## Iteration:  1200 / 2000 [ 60%]  (Sampling)
## Iteration:  1400 / 2000 [ 70%]  (Sampling)

```

```

## Iteration: 1600 / 2000 [ 80%] (Sampling)
## Iteration: 1800 / 2000 [ 90%] (Sampling)
## Iteration: 2000 / 2000 [100%] (Sampling)
##
## Elapsed Time: 0.08434 seconds (Warm-up)
##                 0.062999 seconds (Sampling)
##                 0.147339 seconds (Total)
##
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 2).
##
## Gradient evaluation took 1.6e-05 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 0.16 seconds.
## Adjust your expectations accordingly!
##
##
## Iteration:    1 / 2000 [  0%] (Warmup)
## Iteration:   200 / 2000 [ 10%] (Warmup)
## Iteration:   400 / 2000 [ 20%] (Warmup)
## Iteration:   600 / 2000 [ 30%] (Warmup)
## Iteration:   800 / 2000 [ 40%] (Warmup)
## Iteration:  1000 / 2000 [ 50%] (Warmup)
## Iteration: 1001 / 2000 [ 50%] (Sampling)
## Iteration: 1200 / 2000 [ 60%] (Sampling)
## Iteration: 1400 / 2000 [ 70%] (Sampling)
## Iteration: 1600 / 2000 [ 80%] (Sampling)
## Iteration: 1800 / 2000 [ 90%] (Sampling)
## Iteration: 2000 / 2000 [100%] (Sampling)
##
## Elapsed Time: 0.103181 seconds (Warm-up)
##                 0.063035 seconds (Sampling)
##                 0.166216 seconds (Total)

#posterior sample
post.sample <- as.matrix(post2)

#example of the first 10 posterior samples
post.sample[1:10,]

##           parameters
## iterations (Intercept)      x1      x2      x3      x4      sigma
## [1,]  0.20013187 0.9951292 0.9836621 0.9862314 0.9920344 1.0069258
## [2,] -0.17301698 1.0027072 1.0099915 1.0217840 0.9964729 1.1497936

```

```

##      [3,]  0.26606459 1.0002421 0.9906716 0.9853976 1.0025311 1.1194660
##      [4,]  0.23934385 0.9913646 1.0054863 0.9887745 0.9994093 0.9760771
##      [5,]  0.10217015 0.9917333 1.0084500 0.9925039 0.9939052 0.9815683
##      [6,] -0.17613916 1.0034315 1.0090995 1.0053460 0.9811486 1.0075087
##      [7,]  0.01952345 0.9996066 0.9969790 1.0070605 0.9794494 0.9698949
##      [8,]  0.10767092 1.0006765 1.0017746 1.0014465 1.0108357 1.1981013
##      [9,] -0.20811507 1.0081525 1.0074328 0.9894853 0.9867378 1.0760004
##     [10,]  0.09935476 1.0115203 0.9977477 1.0142334 0.9973044 1.0099387

#no need for the intercept, last parameter is sigma
post.sample <- post.sample[,-1]

#data frame with all submodels

df.rtwos.boot <- rtwos.boot(df[,2:5], post.sample, 10)

n.boot = 10

LMG.Vals.boot<-array(0, c(4,dim(df.rtwos.boot)[2], n.boot))

for (b in 1:n.boot){

  for(i in 1:dim(df.rtwos.boot)[2]){

    gofn<-df.rtwos.boot[,i,b]

    obj.Gelman<-partition(gofn, pcan = 4, var.names = names(df[,2:5]))

    LMG.Vals.boot[,i, b]=obj.Gelman$IJ[,1]
  }
}

quantile(c(LMG.Vals.boot[1,1:1000,]), c(0.025, 0.5, 0.975))

##      2.5%      50%      97.5%
## 0.1381109 0.2030829 0.3103296

quantile(c(LMG.Vals.boot[2,1:1000,]), c(0.025, 0.5, 0.975))

##      2.5%      50%      97.5%
## 0.1497110 0.1935097 0.2664145

```

```
quantile(c(LMG.Vals.boot[3,1:1000,]), c(0.025, 0.5, 0.975))

##      2.5%      50%     97.5%
## 0.1015362 0.1829171 0.2721100

quantile(c(LMG.Vals.boot[4,1:1000,]), c(0.025, 0.5, 0.975))

##      2.5%      50%     97.5%
## 0.06347948 0.16874963 0.24271955
```

In the following example we know that the \mathbf{X} are coming from a normal distribution. The covariance matrix of the distribution is estimated in a Bayesian way. The package Jags is used.

```
zy = df[,2:5]

#-----
# The rest can remain unchanged, except for the specification of difference of
# correlations at the very end.
#-----

# Load some functions used below:
# Install the ellipse package if not already:
# Standardize the data:

#zy = apply(y,2,function(yVec){(yVec-mean(yVec))/sd(yVec)})
# Assemble data for sending to JAGS:
dataList = list(
  zy = zy ,
  Ntotal = nrow(zy) ,
  Nvar = ncol(zy) ,
  # Include original data info for transforming to original scale:
  # For wishart (dwish) prior on inverse covariance matrix:
  zRscal = ncol(zy) , # for dwish prior
  zRmat = diag(x=1,nrow=ncol(zy)) # Rmat = diag(apply(y,2,var))
)

# Define the model:
modelString =
model {
  for ( i in 1:Ntotal ) {
    zy[i,1:Nvar] ~ dmnorm( zMu[1:Nvar] , zInvCovMat[1:Nvar,1:Nvar] )
  }
  for ( varIdx in 1:Nvar ) { zMu[varIdx] ~ dnorm( 0 , 1/2^2 ) }
  zInvCovMat ~ dwish( zRmat[1:Nvar,1:Nvar] , zRscal )
```

```

# Convert invCovMat to sd and correlation:
zCovMat <- inverse( zInvCovMat )

}

" # close quote for modelString
writeLines( modelString , con="Jags-MultivariateNormal-model.txt" )

# Run the chains:
nChain = 3
nAdapt = 500
nBurnIn = 500
nThin = 10
nStepToSave = 20000
require(rjags)
jagsModel = jags.model( file="Jags-MultivariateNormal-model.txt" ,

update( jagsModel , n.iter=nBurnIn )
codaSamples = coda.samples( jagsModel ,
                           n.chains=nChain ,
                           n.burn=nBurnIn ,
                           n.thin=nThin ,
                           n.saving=nStepToSave )
mcmcMat = as.matrix(codaSamples)
chainLength = nrow(mcmcMat)

covMat <- array(NA, c(4,4,chainLength))

```

```

for (i in 1:chainLength){
covMat[1:4,1:4,i]<-matrix(mcmcMat[i,], 4, 4)
}

covMat <- covMat[1:4,1:4,sample(1:20000, replace=F)]


df.rtwos <-rtwos.covm(df[,2:5], post.sample, covMat, 10)

n.boot = 10

LMG.Vals<-array(0, c(4,dim(df.rtwos)[2], n.boot))

for (b in 1:n.boot){

  for(i in 1:dim(df.rtwos)[2]){

    gofn<-df.rtwos[,i,b]

    obj.Gelman<-partition(gofn, pcan = 4, var.names = names(df[,2:5]))

    LMG.Vals[,i, b]=obj.Gelman$IJ[,1]
  }
}

quantile(c(LMG.Vals[1,1:1000,]), c(0.025, 0.5, 0.975))

##      2.5%      50%     97.5%
## 0.07955934 0.19561327 0.25671047

quantile(c(LMG.Vals[2,1:1000,]), c(0.025, 0.5, 0.975))

##      2.5%      50%     97.5%
## 0.1059393 0.1921079 0.3129919

quantile(c(LMG.Vals[3,1:1000,]), c(0.025, 0.5, 0.975))

##      2.5%      50%     97.5%
## 0.1073336 0.1822257 0.2556199

quantile(c(LMG.Vals[4,1:1000,]), c(0.025, 0.5, 0.975))

##      2.5%      50%     97.5%
## 0.1192879 0.1863754 0.3401010

```

Using the bootstrap samples of the covariance matrix or samples from the posterior covariance matrix resulted in very similar LMG values. Bootstrap seems to be a valuable option for stochastic predictors when the distribution of the predictors is unknown. Even when the distribution is known the difference seems to be tiny. A benefit of going the full Bayesian way is that we can also include prior knowledge of the covariance matrix. For stochastic predictors the uncertainty about the covariance matrix is reflected in the large credible intervals. Even when we would knew the exact regression parameters, there is alot of uncertainty in the LMG values caused by the uncertainty about the covariance matrix. Code xx shows the uncertainty about the LMG values caused by the uncertainty about the covariance matrix.

Another interesting option in some cases would be to use the shrinkage estimate of the covariance matrix.

```
#How much variance is effectively in the bootstrap matrix when we know the regression parameter

#fake post sample

x1 <- rnorm(50, 0, 1); x2 <- rnorm(50, 0, 1)
x3 <- rnorm(50, 0, 1); x4 <- rnorm(50, 0, 1)
#b1 <- 0.5; b2 <- 1; b3 <- 2; b4 <- 0
b1 <- 1; b2 <- 1; b3 <- 1; b4 <- 1

y <- b1*x1 + x2*b2 + b3*x3 + b4*x4 + rnorm(50, 0, 1)

df <- data.frame(y = y, x1 = x1, x2 = x2, x3 = x3, x4 = x4)

post.sample <- matrix(1, 1, 5)

df.rtwos.boot <- rtwos.boot(df[, 2:5], post.sample, 1000)

n.boot = 1000

LMG.Vals.boot<-array(0, c(4, dim(df.rtwos.boot)[2], n.boot))

for (b in 1:n.boot){

  for(i in 1:dim(df.rtwos.boot)[2]){

    gofn<-df.rtwos.boot[, i, b]

    obj.Gelman<-partition(gofn, pcan = 4, var.names = names(df[, 2:5]))

    LMG.Vals.boot[, i, b]=obj.Gelman$IJ[, 1]
  }
}
```

```
}

quantile(c(LMG.Vals.boot[1,1,]), c(0.025, 0.5, 0.975))

##      2.5%      50%     97.5%
## 0.0979818 0.1894039 0.2799671

quantile(c(LMG.Vals.boot[2,1,]), c(0.025, 0.5, 0.975))

##      2.5%      50%     97.5%
## 0.1773573 0.2536383 0.3293491

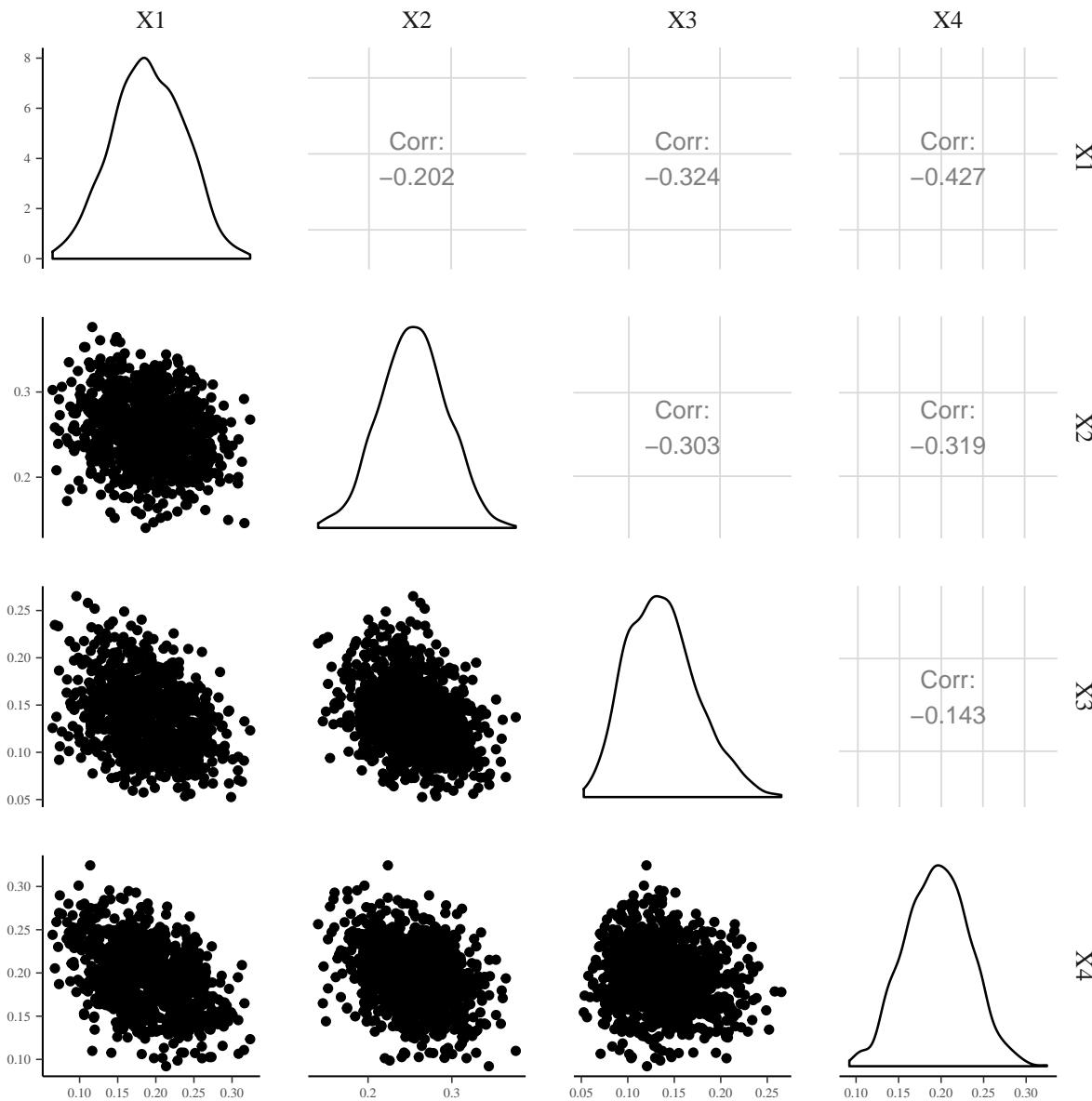
quantile(c(LMG.Vals.boot[3,1,]), c(0.025, 0.5, 0.975))

##      2.5%      50%     97.5%
## 0.07376485 0.13427948 0.21775383

quantile(c(LMG.Vals.boot[4,1,]), c(0.025, 0.5, 0.975))

##      2.5%      50%     97.5%
## 0.1266168 0.1955925 0.2684755

dat <- data.frame(t(LMG.Vals.boot[1:4,1,]))
pairs.chart <- ggpairs(dat, lower = list(list(combo = "facetdensity")), upper = list(list(c
pairs.chart
```



```
cov(df[,2:5])

##          x1          x2          x3          x4
## x1  0.8507778  0.2803422 -0.10769517 -0.10821297
## x2  0.2803422  1.0379131 -0.16375752  0.06331260
## x3 -0.1076952 -0.1637575  0.80799891  0.05999761
## x4 -0.1082130  0.0633126  0.05999761  0.89839912
```

#Comparison of sample covariance and shrink covariance matrix

```
cov(df[,2:5])

##          x1          x2          x3          x4
## x1  0.918802371 -0.10264450  0.009003572  0.1281456
```

```

## x2 -0.102644497 0.97559948 -0.012534282 0.1849557
## x3 0.009003572 -0.01253428 0.961040604 -0.0892520
## x4 0.128145565 0.18495569 -0.089252004 0.7174392

cov.shrink(df[,2:5])

## Estimating optimal shrinkage intensity lambda.var (variance vector): 1
##
## Estimating optimal shrinkage intensity lambda (correlation matrix): 1
##          x1      x2      x3      x4
## x1 0.9399215 0.0000000 0.0000000 0.0000000
## x2 0.0000000 0.9399215 0.0000000 0.0000000
## x3 0.0000000 0.0000000 0.9399215 0.0000000
## x4 0.0000000 0.0000000 0.0000000 0.9399215
## attr(,"lambda")
## [1] 1
## attr(,"lambda.estimated")
## [1] TRUE
## attr(,"class")
## [1] "shrinkage"
## attr(,"lambda.var")
## [1] 1
## attr(,"lambda.var.estimated")
## [1] TRUE

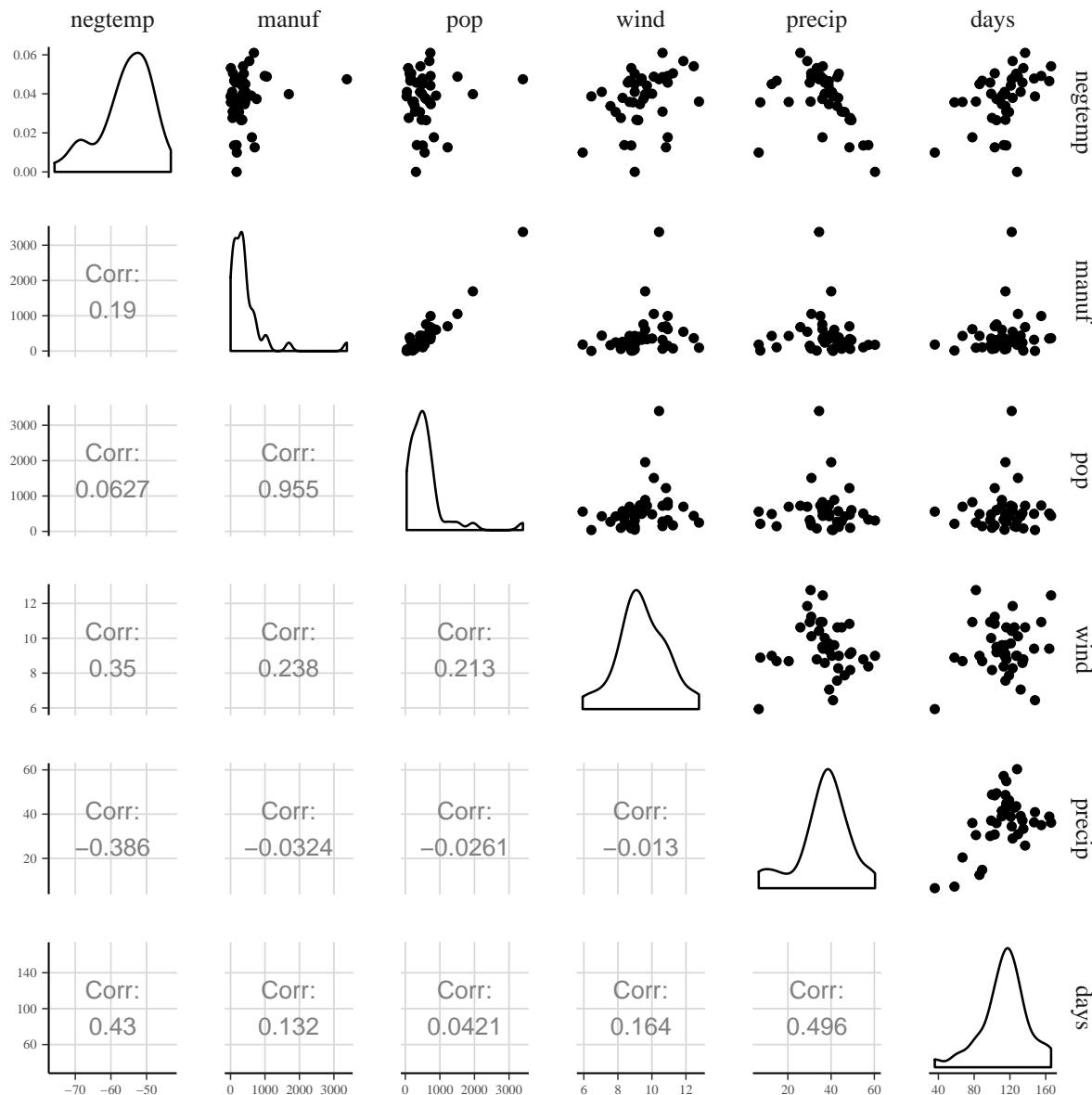
```

1.2 Real World Data

The following example data are taken from the book Bayesian Regression Modeling with INLA. The data were about air pollution in 41 cities in the United States originally published in Everitt (2006). The data consists of the SO₂ level as the dependent variable and six explanatory variables. Two of the explanatory variables are related to human ecology (pop, manuf) and four others are related to climate (negtemp, wind, precip, days).

Table 1.1: Variable description of air pollution data

Variable Name	Description	Codes/Values
SO2	sulfur dioxide content of air	microgrammes per cubic meter
negtemp	negative value of average	fahrenheit
manuf	number of manufacturing enterprises employing 20 or more workers	integers
pop	population size in thousands (1970 census)	numbers
wind	average wind speed	miles per hour
precip	average annual precipitation	inches
days	average number of days with precipitation per year	integers



A simple linear regression model including the dependent variable SO2 and the six explanatory variables is fitted with the lm command in R. The R^2 of the full model is 0.670.

```

usair.lm <- lm(SO2~, data = usair)

#Code assuming stochastic predictors
run<-boot.relimp(usair.lm, fixed=TRUE)

booteval.relimp(run, bty = "perc", level = 0.95,
                 sort = FALSE, norank = TRUE, nodiff = TRUE,
                 typesel = c("lmg"))

## Response variable: SO2
## Total response variance: 550.9476
## Analysis based on 41 observations
##
## 6 Regressors:
## negtemp manuf pop wind precip days
## Proportion of variance explained by model: 66.95%
## Metrics are not normalized (rela=FALSE).
##
## Relative importance metrics:
##
##          lmg
## negtemp 0.11112440
## manuf   0.29645486
## pop     0.16612543
## wind    0.01670048
## precip  0.01867491
## days    0.06043172
##
## Average coefficients for different model sizes:
##
##          1X        2Xs        3Xs        4Xs        5Xs
## negtemp 1.40813251  1.334422627  1.281176139  1.25765114  1.28261316
## manuf   0.02685872  0.037288506  0.046426172  0.05392217  0.05982752
## pop     0.02001359  0.004397813 -0.009249699 -0.02072002 -0.03035031
## wind    1.55574078 -0.024806053 -1.384826041 -2.37946774 -2.98034367
## precip  0.10826200  0.114668437  0.164271600  0.26014506  0.40567633
## days    0.32726028  0.297872739  0.241114488  0.15574609  0.04248847
##
##          6Xs
## negtemp 1.26794109
## manuf   0.06491817
## pop     -0.03927674
## wind    -3.18136579

```

```

## precip  0.51235896
## days    -0.05205019
##
##
## Confidence interval information ( 1000 bootstrap replicates, bty= perc ):
## Relative Contributions with confidence intervals:
##
##           Lower   Upper
## percentage 0.95  0.95
## negtemp.lmg 0.1111  0.0395 0.2157
## manuf.lmg   0.2965  0.2031 0.4133
## pop.lmg     0.1661  0.1137 0.2405
## wind.lmg    0.0167  0.0088 0.0686
## precip.lmg  0.0187  0.0074 0.0658
## days.lmg    0.0604  0.0221 0.1436
##
## CAUTION: Bootstrap confidence intervals can be somewhat liberal.
## NOTE: X-matrix has been considered as fixed for bootstrapping.

bayes.usair <- stan_glm(SO2 ~ . ,
                           data = usair,
                           chains = 4, cores = 4)

#posterior sample
post.sample <- as.matrix(bayes.usair)

#example of the first 10 posterior samples
post.sample[1:10,]

##
##           parameters
## iterations (Intercept) negtemp      manuf       pop       wind
## [1,]    76.63403 0.9372802 0.04841073 -0.023218029 -2.64495618
## [2,]   125.55302 1.2466290 0.09637679 -0.066305907 -3.19036264
## [3,]    57.36694 0.9911093 0.03160349 -0.009311882  0.01044934
## [4,]    84.42258 1.4791986 0.03711957 -0.014665873 -0.34841785
## [5,]   93.21212 1.2233823 0.03128086 -0.008084000  0.33863006
## [6,]   87.03184 1.5349165 0.00412235  0.012872617 -2.55050086
## [7,]   94.75793 0.7006997 0.05364753 -0.031105908 -3.31781375
## [8,]   83.86657 0.7426624 0.04801954 -0.025619233 -1.38722141
## [9,]  102.88610 1.4435390 0.05184942 -0.025740406 -1.64110264
## [10,] 118.68871 1.0830053 0.05654027 -0.029810740 -5.16775861
##
##           parameters
## iterations      precip        days      sigma

```

```

##      [1,]  0.1604472  0.118589312 13.37797
##      [2,]  0.5069793 -0.137335376 14.59451
##      [3,]  0.5041612 -0.005711373 14.32118
##      [4,]  0.4686286  0.039197926 14.92060
##      [5,]  0.2656314 -0.159223921 15.44457
##      [6,]  0.5662126  0.185683615 16.19240
##      [7,] -0.1457218  0.051595750 14.96856
##      [8,]  0.3112843 -0.139255549 14.45331
##      [9,]  0.2724058  0.048202080 14.16934
##     [10,]  0.2932432  0.029287718 12.75397

#no need for the intercept, last parameter is sigma
post.sample <- post.sample[,-1]

#data frame with all submodels
df.rtwos <- rtwos(usair[,2:7], post.sample)

df.rtwos[,1:3]

##          X1        X2        X3
## none 0.000000000 0.000000e+00 0.00000000
## x1   0.2095431804 1.775796e-01 0.13489062
## x2   0.4446934425 4.179614e-01 0.41661681
## x3   0.2929164865 2.203668e-01 0.30988091
## x4   0.0142179243 9.678623e-03 0.06630532
## x5   0.0002332062 3.691272e-07 0.01541407
## x6   0.1708118912 8.639088e-02 0.13975755
## x1 x2  0.5583792612 5.104269e-01 0.47869292
## x1 x3  0.4732622423 3.746202e-01 0.42079549
## x1 x4  0.2114451600 1.803153e-01 0.15386482
## x1 x5  0.2529084140 2.084866e-01 0.21806295
## x1 x6  0.2669814140 1.931435e-01 0.19205192
## x2 x3  0.5496435031 6.689090e-01 0.45766492
## x2 x4  0.4463420384 4.212209e-01 0.42806286
## x2 x5  0.4460556275 4.183760e-01 0.43768634
## x2 x6  0.5524401553 4.622856e-01 0.50146874
## x3 x4  0.2929345455 2.203690e-01 0.33015361
## x3 x5  0.2937818488 2.205027e-01 0.32912977
## x3 x6  0.4456911919 2.956683e-01 0.43288956
## x4 x5  0.0145009021 9.679073e-03 0.08256418
## x4 x6  0.1735285646 8.897505e-02 0.17929668
## x5 x6  0.2185774367 1.148293e-01 0.14474330
## x1 x2 x3  0.5999081239 6.806839e-01 0.49085217

```

```

## x1 x2 x4      0.5828622187 5.369044e-01 0.47953024
## x1 x2 x5      0.5913931426 5.328805e-01 0.54740304
## x1 x2 x6      0.6010744974 5.189484e-01 0.52124884
## x1 x3 x4      0.4962294308 3.959121e-01 0.42165365
## x1 x3 x5      0.5170723418 4.058519e-01 0.50460899
## x1 x3 x6      0.5266597023 3.883851e-01 0.47376230
## x1 x4 x5      0.2583510585 2.145589e-01 0.22766353
## x1 x4 x6      0.2692358837 1.960944e-01 0.20998386
## x1 x5 x6      0.2678265140 2.091053e-01 0.21831904
## x2 x3 x4      0.5529165888 6.757592e-01 0.46702982
## x2 x3 x5      0.5514266268 6.697259e-01 0.47971614
## x2 x3 x6      0.6100903238 6.738347e-01 0.51696588
## x2 x4 x5      0.4476880079 4.216230e-01 0.44930231
## x2 x4 x6      0.5599544061 4.698455e-01 0.50602109
## x2 x5 x6      0.5747343757 4.722065e-01 0.50147391
## x3 x4 x5      0.2938018636 2.205046e-01 0.34970572
## x3 x4 x6      0.4491269234 2.977539e-01 0.44059086
## x3 x5 x6      0.4819547629 3.164232e-01 0.43459853
## x4 x5 x6      0.2193728852 1.158819e-01 0.18169678
## x1 x2 x3 x4   0.6190277770 6.963630e-01 0.49239417
## x1 x2 x3 x5   0.6222290346 6.872889e-01 0.55188213
## x1 x2 x3 x6   0.6323045906 6.819204e-01 0.52802696
## x1 x2 x4 x5   0.6247895508 5.669954e-01 0.54744668
## x1 x2 x4 x6   0.6259280279 5.455982e-01 0.52201955
## x1 x2 x5 x6   0.6019013867 5.348317e-01 0.54741474
## x1 x3 x4 x5   0.5508487894 4.358934e-01 0.50477150
## x1 x3 x4 x6   0.5505484630 4.101271e-01 0.47445457
## x1 x3 x5 x6   0.5284278757 4.073263e-01 0.50460907
## x1 x4 x5 x6   0.2708395551 2.159926e-01 0.22869636
## x2 x3 x4 x5   0.5546775480 6.765545e-01 0.48921418
## x2 x3 x4 x6   0.6181232920 6.824053e-01 0.52131583
## x2 x3 x5 x6   0.6210337734 6.739417e-01 0.51746614
## x2 x4 x5 x6   0.5847711002 4.814350e-01 0.50603453
## x3 x4 x5 x6   0.4880867456 3.200922e-01 0.44165482
## x1 x2 x3 x4 x5 0.6488611356 7.067140e-01 0.55188379
## x1 x2 x3 x4 x6 0.6523787469 6.977712e-01 0.52929783
## x1 x2 x3 x5 x6 0.6324408123 6.895395e-01 0.55189795
## x1 x2 x4 x5 x6 0.6300577358 5.728615e-01 0.54746790
## x1 x3 x4 x5 x6 0.5566788981 4.406473e-01 0.50477765
## x2 x3 x4 x5 x6 0.6308577065 6.827467e-01 0.52211213
## all            0.6544018382 7.119186e-01 0.55189833

# prepare data frame for LMG values

```

```

LMG.Vals<-matrix(0, 6, dim(df.rtwos)[2])

for(i in 1:dim(df.rtwos)[2]){

gofn<-df.rtwos[,i]

obj.Gelman<-partition(gofn, pcan = 6, var.names = names(usair[,2:7]))

LMG.Vals[,i]=obj.Gelman$IJ[,1]
}

# posterior LMG distribution of each variable
quantile(LMG.Vals[1,], c(0.025, 0.5, 0.975))

##      2.5%      50%      97.5%
## 0.03140146 0.10425536 0.20332655

quantile(LMG.Vals[2,], c(0.025, 0.5, 0.975))

##      2.5%      50%      97.5%
## 0.1528304 0.2641768 0.3565642

quantile(LMG.Vals[3,], c(0.025, 0.5, 0.975))

##      2.5%      50%      97.5%
## 0.08782959 0.15190153 0.21056465

quantile(LMG.Vals[4,], c(0.025, 0.5, 0.975))

##      2.5%      50%      97.5%
## 0.007501645 0.017654467 0.071564974

quantile(LMG.Vals[5,], c(0.025, 0.5, 0.975))

##      2.5%      50%      97.5%
## 0.006988704 0.020612799 0.069884355

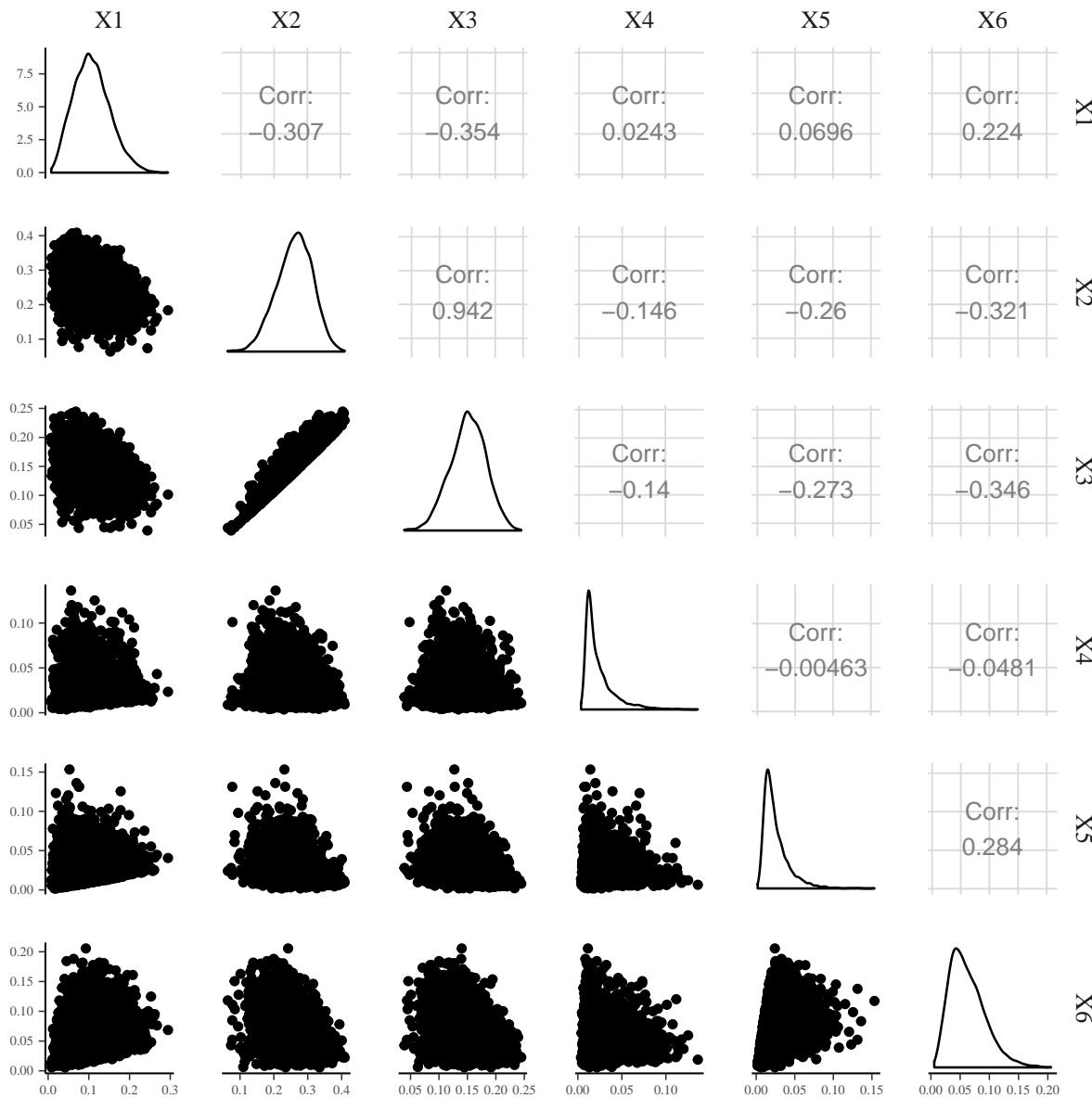
quantile(LMG.Vals[6,], c(0.025, 0.5, 0.975))

##      2.5%      50%      97.5%
## 0.01761491 0.05739723 0.13408836

#Visualization
dat <- data.frame(t(LMG.Vals))

pairs.chart <- ggpairs(dat, lower = list(list(combo = "facetdensity")), upper = list(list(c
pairs.chart

```



```
#use prior knowledge
my_prior <- normal(location = c(1, 0.05,-0.03,-3, 0, 0), scale = c(0.1, 0.01,0.01,0.1, 0.1, 0.1))

bayes.usair <- stan_glm(SO2 ~ . ,
                         data = usair, prior = my_prior,
                         chains = 4, cores = 4)

#posterior sample
post.sample <- as.matrix(bayes.usair)

#example of the first 10 posterior samples
post.sample[1:10,]
```

```

##           parameters
## iterations (Intercept)    negtemp      manuf       pop      wind
## [1,]     89.81921 0.9954251 0.04757398 -0.01827321 -2.874278
## [2,]    102.34929 0.8576861 0.05986172 -0.04000167 -3.068902
## [3,]    82.48015 0.8726391 0.06028279 -0.02943372 -2.886816
## [4,]   104.68428 1.1246337 0.06291728 -0.02986882 -2.992389
## [5,]   105.32564 1.0099265 0.06004396 -0.03390757 -3.042098
## [6,]   97.62453 0.9849631 0.05775336 -0.03569145 -3.181789
## [7,]  108.99691 1.1103095 0.06485238 -0.03683087 -3.038839
## [8,]  108.00249 1.1095669 0.06392837 -0.03489973 -3.022639
## [9,]  104.88886 0.8596561 0.06802693 -0.04236175 -3.114606
## [10,] 110.01475 1.0939596 0.03827552 -0.01521879 -2.874750
##           parameters
## iterations      precip        days      sigma
## [1,]  0.276227708 0.012587472 15.36679
## [2,] -0.083183938 0.032998990 17.82745
## [3,]  0.059481326 0.090567911 11.39204
## [4,]  0.166154108 -0.005130413 15.37028
## [5,] -0.005799239 0.011521322 15.34251
## [6,]  0.078044836 0.044796613 18.45930
## [7,]  0.224391786 -0.076565709 16.99801
## [8,]  0.233775974 -0.075735576 17.02467
## [9,]  0.017462691 -0.001904609 14.16142
## [10,] -0.019851667 -0.010796473 15.93392

#no need for the intercept, last parameter is sigma
post.sample <- post.sample[,-1]

#data frame with all submodels
df.rtwos <- rtwos(usair[,2:7], post.sample)

# prepare data frame for LMG values

LMG.Vals<-matrix(0, 6, dim(df.rtwos)[2])

for(i in 1:dim(df.rtwos)[2]){

  gofn<-df.rtwos[,i]

  obj.Gelman<-partition(gofn, pcan = 6, var.names = names(usair[,2:7]))

  LMG.Vals[,i]=obj.Gelman$IJ[,1]
}

```

```
}  
  
# posterior LMG distribution of each variable  
quantile(LMG.Vals[1,], c(0.025, 0.5, 0.975))  
  
##      2.5%      50%     97.5%  
## 0.07362282 0.11363424 0.16192618  
  
quantile(LMG.Vals[2,], c(0.025, 0.5, 0.975))  
  
##      2.5%      50%     97.5%  
## 0.1695037 0.2591710 0.3325388  
  
quantile(LMG.Vals[3,], c(0.025, 0.5, 0.975))  
  
##      2.5%      50%     97.5%  
## 0.09346488 0.15095056 0.20686893  
  
quantile(LMG.Vals[4,], c(0.025, 0.5, 0.975))  
  
##      2.5%      50%     97.5%  
## 0.01262413 0.01637030 0.02123865  
  
quantile(LMG.Vals[5,], c(0.025, 0.5, 0.975))  
  
##      2.5%      50%     97.5%  
## 0.007893674 0.013095342 0.025689887  
  
quantile(LMG.Vals[6,], c(0.025, 0.5, 0.975))  
  
##      2.5%      50%     97.5%  
## 0.01733809 0.05146921 0.10703968  
  
#Visualization  
dat <- data.frame(t(LMG.Vals))  
  
pairs.chart <- ggpairs(dat, lower = list(list(combo = "facetdensity")), upper = list(list(conti  
pairs.chart
```

