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:

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

$$\beta_1 = 0.5, \beta_2 = 1, \beta_3 = 2, \beta_4 = 0, \sigma^2 = 1 \tag{1.2}$$

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

The values of the four predictors are sampled from a standard normal distribution. These values are then multiplied by the regression coefficients to obatin 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 periode 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 0.000127 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 1.27 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: 0.074254 seconds (Warm-up)
##                0.065907 seconds (Sampling)
##                0.140161 seconds (Total)
##
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 2).
##
## Gradient evaluation took 1.3e-05 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 0.13 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: 0.074243 seconds (Warm-up)
##                0.066928 seconds (Sampling)
##                0.141171 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,]  0.35178659 0.8655954 1.142714 0.8983258 0.9083553 0.9562517
##       [2,]  0.06255488 0.8535117 1.049424 0.7477095 0.8104561 1.0594685
##       [3,]  0.31597084 0.7650021 1.340354 0.9362496 1.1261895 1.2277133
##       [4,]  0.06533558 1.0942983 0.951873 0.8730962 0.7628246 1.0079768
##       [5,]  0.28055153 1.0896500 1.150824 0.9820826 0.8198639 1.4869377
##       [6,]  0.40788418 0.9904042 1.312746 0.9577036 0.7945795 1.6462027
##       [7,]  0.54576471 0.8962367 1.204006 0.9703716 1.0254554 1.3898373
##       [8,]  0.28610365 0.8699938 1.110565 0.7824389 0.9131850 0.8861995
##       [9,]  0.18966765 0.8408969 1.076889 0.8853991 0.8885238 1.2108844
##      [10,] -0.04572765 1.0818832 1.255911 0.5915038 1.0709531 0.9675569
```

```
#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 0.00000000 0.0000000 0.00000000
## x1       0.1155161 0.1265817 0.05861261 0.2214549 0.14825388
## x2       0.2743602 0.2539277 0.30650867 0.1766879 0.19380243
## x3       0.3583408 0.2997988 0.30910601 0.3549040 0.30995688
```

```
## x4       0.1549655 0.1350313 0.18054621 0.1127693 0.09532595
## x1 x2    0.4652522 0.4560452 0.42407258 0.4808169 0.41292796
## x1 x3    0.4251260 0.3791724 0.33656876 0.5076723 0.40604211
## x1 x4    0.2878599 0.2785842 0.25263665 0.3548052 0.25903292
## x2 x3    0.5829040 0.5098845 0.56668905 0.4920195 0.46490367
## x2 x4    0.4267990 0.3866899 0.48417204 0.2877273 0.28746299
## x3 x4    0.4514269 0.3819125 0.42701257 0.4159155 0.36089513
## x1 x2 x3 0.7078564 0.6499467 0.63719154 0.7115499 0.61751520
## x1 x2 x4 0.6397839 0.6100421 0.62046964 0.6158245 0.52523325
## x1 x3 x4 0.5327627 0.4761689 0.46519159 0.5866294 0.46992805
## x2 x3 x4 0.6781015 0.5939166 0.68714028 0.5543666 0.51713991
## all      0.8239334 0.7547871 0.77532252 0.7966452 0.68703340
```

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

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

LMG.Vals.T<-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[,i]=obj.Gelman$IJ[,1]
  LMG.Vals.J[,i]=obj.Gelman$IJ[,2]
        LMG.Vals.T[,i]=obj.Gelman$IJ[,3]
}

varnames <- row.names(obj.Gelman$IJ)


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

##       2.5%        50%      97.5%
## 0.06083054 0.14395976 0.24118322
```

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

```
##      2.5%       50%     97.5%
## 0.1447740 0.2572612 0.3774172
```

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

```
##      2.5%       50%     97.5%
## 0.1319720 0.2334742 0.3435360
```

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

```
##       2.5%        50%      97.5%
## 0.06183814 0.13457410 0.23772626
```

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

```
##          2.5%           50%         97.5%
## -0.029947641 -0.017765115 -0.006019912
```

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

```
##          2.5%           50%         97.5%
## -0.025073063 -0.012040967  0.002017762
```

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

```
##       2.5%        50%      97.5%
## 0.06025121 0.07776257 0.09024082
```

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

```
##        2.5%         50%       97.5%
## 0.009420845 0.018006878 0.027944757
```

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

```
##       2.5%        50%      97.5%
## 0.04794251 0.12584522 0.22163041
```

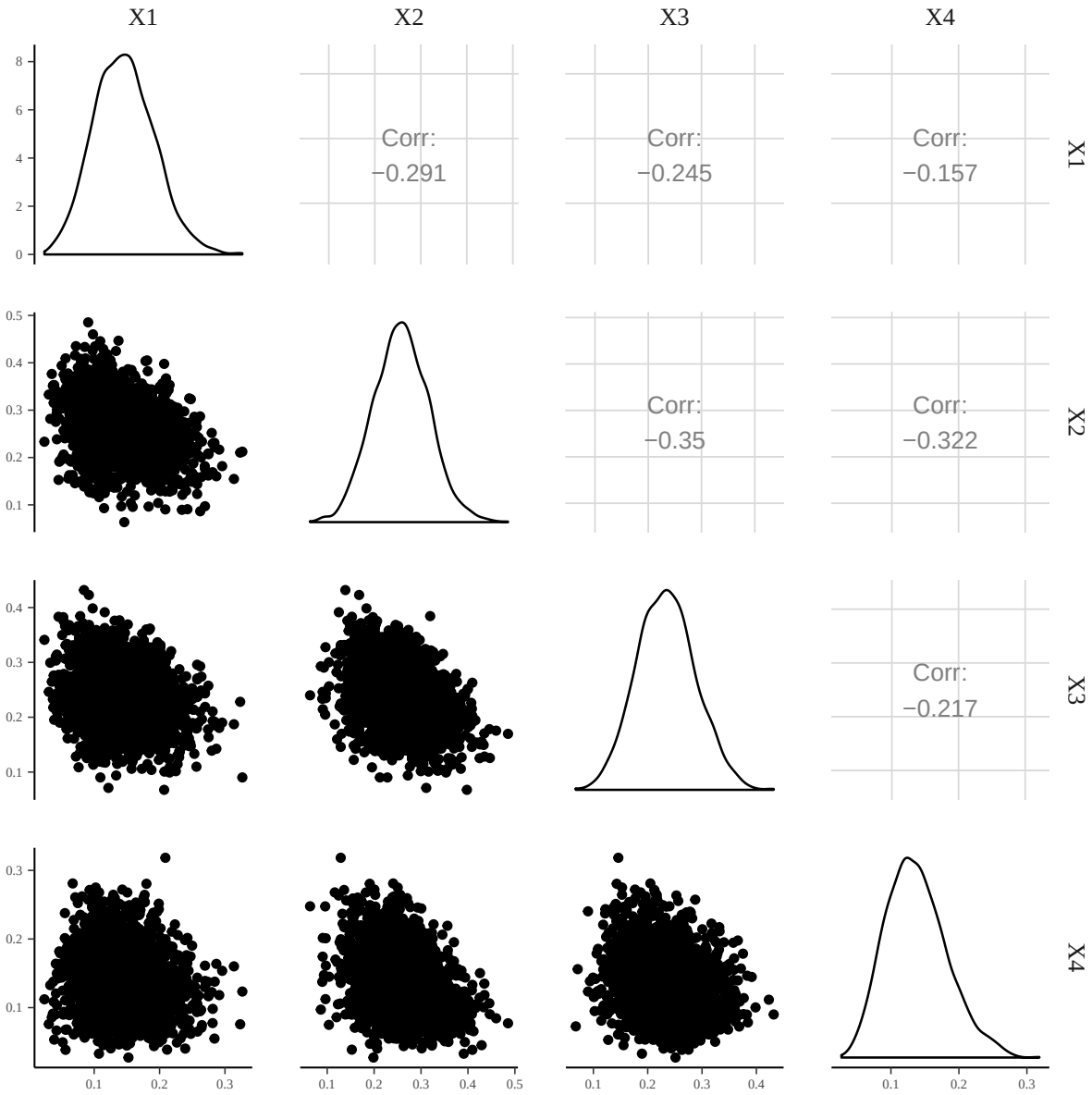
```
quantile(LMG.Vals.T[2,], c(0.025, 0.5, 0.975))
```

```
##      2.5%       50%     97.5%
## 0.1343851 0.2455826 0.3661442
```

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

```
##      2.5%       50%     97.5%
## 0.1936227 0.3106018 0.4305216
```

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

##       2.5%        50%      97.5%
## 0.07340494 0.15240129 0.26038850

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

pairs.chart <- ggpairs(dat, lower = list(list(combo = "facetdensity")), upper = list(list(c
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: 5.207532
## Analysis based on 50 observations
##
## 4 Regressors:
## x1 x2 x3 x4
## Proportion of variance explained by model: 80.7%
## Metrics are not normalized (rela=FALSE).
##
## Relative importance metrics:
##
##          lmg
## x1 0.1682618
## x2 0.1052344
## x3 0.2570110
## x4 0.2765038
##
## Average coefficients for different model sizes:
##
##           1X       2Xs       3Xs       4Xs
## x1 1.0374575 1.0292667 1.0085588 0.9715225
## x2 0.7930673 0.7587584 0.7155455 0.6624706
## x3 1.1681569 1.1317494 1.0891532 1.0440098
## x4 1.2582913 1.2410188 1.2174703 1.1894942
##
##
##  Confidence interval information ( 1000 bootstrap replicates, bty= perc ):
## Relative Contributions with confidence intervals:
##
##                   Lower  Upper
##        percentage 0.95   0.95
## x1.lmg 0.1683     0.0854 0.2798
## x2.lmg 0.1052     0.0494 0.1946
## x3.lmg 0.2570     0.1579 0.3588
## x4.lmg 0.2765     0.1789 0.3920
```

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

Variable	I	J	Total
x1	0.144 (0.061, 0.241)	-0.018 (-0.03, -0.006)	0.126 (0.048, 0.222)
x2	0.257 (0.145, 0.377)	-0.012 (-0.025, 0.002)	0.246 (0.134, 0.366)
x3	0.233 (0.132, 0.344)	0.078 (0.06, 0.09)	0.311 (0.194, 0.431)
x4	0.135 (0.062, 0.238)	0.018 (0.009, 0.028)	0.152 (0.073, 0.26)

Table 1.1: Variance decomposition for stochastic predictors using bootstrap. I = LMG values, J = joint contribution, Total = total explained variance in one-predictor only model

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 exogeinity and conditional independence the posterior distributions of the regression paramters $\boldsymbol{\beta}$ 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 poserior 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: 5.21
## Analysis based on 50 observations
##
## 4 Regressors:
## x1 x2 x3 x4
## Proportion of variance explained by model: 80.7%
```

```
## Metrics are not normalized (rela=FALSE).
##
## Relative importance metrics:
##
##      lmg
## x1 0.168
## x2 0.105
## x3 0.257
## x4 0.277
##
## Average coefficients for different model sizes:
##
##       1X   2Xs   3Xs   4Xs
## x1 1.037 1.029 1.009 0.972
## x2 0.793 0.759 0.716 0.662
## x3 1.168 1.132 1.089 1.044
## x4 1.258 1.241 1.217 1.189
##
##
##  Confidence interval information ( 1000 bootstrap replicates, bty= perc ):
## Relative Contributions with confidence intervals:
##
##                    Lower  Upper
##        percentage 0.95   0.95
## x1.lmg 0.1683     0.0675 0.3082
## x2.lmg 0.1052     0.0259 0.2261
## x3.lmg 0.2570     0.1257 0.3875
## x4.lmg 0.2765     0.1503 0.4095
##
## CAUTION: Bootstrap confidence intervals can be somewhat liberal.
```

```
#-------------------------------------------------------------------------------
```

```
df.rtwos.boot <-rtwos.boot(df[,2:5], post.sample, 100)

n.boot = 100

LMG.Vals.I.boot<-array(0, c(4,dim(df.rtwos.boot)[2], n.boot))
LMG.Vals.J.boot<-array(0, c(4,dim(df.rtwos.boot)[2], n.boot))
LMG.Vals.T.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.I.boot[,i, b]=obj.Gelman$IJ[,1]
        LMG.Vals.J.boot[,i, b]=obj.Gelman$IJ[,2]
        LMG.Vals.T.boot[,i, b]=obj.Gelman$IJ[,3]

}

}

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

##   2.5%    50%  97.5%
## 0.0481 0.1786 0.3276

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

##   2.5%    50%  97.5%
## 0.0736 0.1959 0.3503

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

##   2.5%    50%  97.5%
## 0.0178 0.0527 0.1651

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

##   2.5%    50%  97.5%
## 0.0601 0.1999 0.3977

#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 = FAL

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 0.000217 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 2.17 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: 0.146693 seconds (Warm-up)
##                0.106807 seconds (Sampling)
##                0.2535 seconds (Total)
##
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 2).
##
## Gradient evaluation took 2.9e-05 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 0.29 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: 0.14285 seconds (Warm-up)
##                0.113844 seconds (Sampling)
##                0.256694 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,]     -0.1276 0.982 1.012 1.016 0.998 0.869
##       [2,]     -0.1000 1.012 0.995 0.990 0.994 0.722
##       [3,]     -0.3582 0.995 1.006 0.980 1.013 0.681
##       [4,]     -0.0984 1.011 0.995 1.000 0.999 1.071
##       [5,]     -0.1142 1.007 1.003 0.999 0.998 1.111
##       [6,]     -0.3027 1.009 1.014 0.998 1.000 0.871
##       [7,]     -0.2289 0.986 0.991 1.008 1.001 0.914
##       [8,]     -0.0522 1.014 1.010 0.995 0.996 0.717
##       [9,]     -0.1920 0.990 0.987 0.995 1.005 0.681
##      [10,]     -0.1777 1.006 1.014 1.006 0.993 1.032
```

```
#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.I.boot.p<-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.I.boot.p[,i, b]=obj.Gelman$IJ[,1]
}

}

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

##  2.5%   50% 97.5%
## 0.113 0.248 0.401

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

##  2.5%   50% 97.5%
## 0.107 0.177 0.258

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

##   2.5%    50%  97.5%
## 0.0479 0.0782 0.1288

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

##   2.5%    50%  97.5%
## 0.0989 0.2850 0.3869
```

Variable	I	J	Total
x1	0.179 (0.048, 0.328)	-0.011 (-0.074, 0.086)	0.168 (0.012, 0.38)
x2	0.196 (0.074, 0.35)	-0.041 (-0.098, 0.049)	0.156 (0.013, 0.373)
x3	0.053 (0.018, 0.165)	-0.035 (-0.072, 0.009)	0.015 (3.314×10^{-5}, 0.132)
x4	0.2 (0.06, 0.398)	-0.041 (-0.102, 0.047)	0.16 (0.012, 0.383)

Table 1.2: Variance decomposition. I = LMG values, J = joint contribution, Total = total explained variance in one-predictor only model

In the following example we know that the **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" ,
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 50
##    Unobserved stochastic nodes: 5
##    Total graph size: 82
##
## Initializing model

update( jagsModel , n.iter=nBurnIn )
codaSamples = coda.samples( jagsModel ,




# Convergence diagnostics:
parameterNames = varnames(codaSamples) # get all parameter names


# Examine the posterior distribution:
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.I.covm<-array(0, c(4,dim(df.rtwos)[2], n.boot))
LMG.Vals.J.covm<-array(0, c(4,dim(df.rtwos.boot)[2], n.boot))
LMG.Vals.T.covm<-array(0, c(4,dim(df.rtwos.boot)[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.covm[,i, b]=obj.Gelman$IJ[,1]
                LMG.Vals.J.covm[,i, b]=obj.Gelman$IJ[,2]
                LMG.Vals.T.covm[,i, b]=obj.Gelman$IJ[,3]

        }

}

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

##  2.5%   50% 97.5%
## 0.127 0.219 0.345

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

##  2.5%   50% 97.5%
## 0.159 0.246 0.331

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

##  2.5%   50% 97.5%
## 0.106 0.131 0.238

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

##  2.5%   50% 97.5%
## 0.100 0.184 0.287
```

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

Variable	I	J	Total
x1	0.219 (0.127, 0.345)	-0.016 (-0.062, 0.048)	0.213 (0.069, 0.387)
x2	0.246 (0.159, 0.331)	-0.007 (-0.106, 0.074)	0.23 (0.06, 0.403)
x3	0.131 (0.106, 0.238)	-0.043 (-0.081, 0.005)	0.1 (0.036, 0.242)
x4	0.184 (0.1, 0.287)	-0.037 (-0.09, 0.01)	0.155 (0.023, 0.259)

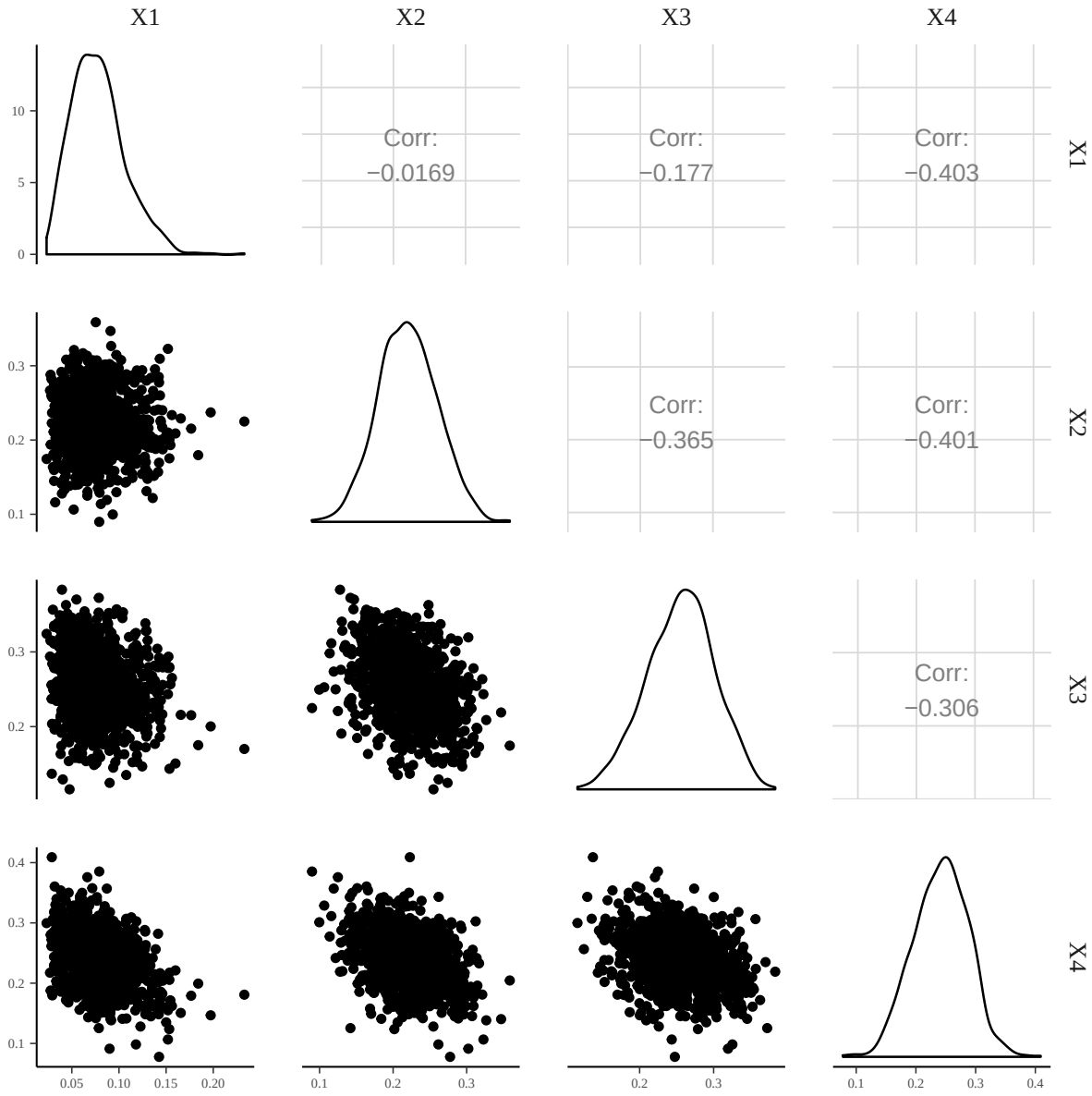
Table 1.3: Variance decomposition. I = LMG values, J = joint contribution, Total = total explained variance in one-predictor only model

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

}

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

##   2.5%    50%  97.5%
## 0.0331 0.0755 0.1425

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

##  2.5%   50% 97.5%
## 0.143 0.219 0.299

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

##  2.5%   50% 97.5%
## 0.163 0.258 0.340

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

##  2.5%   50% 97.5%
## 0.152 0.243 0.328

dat <- data.frame(t(LMG.Vals.I.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  0.5408 -0.0195 -0.0646 -0.1055
## x2 -0.0195  0.9811  0.2331 -0.0374
## x3 -0.0646  0.2331  0.9027  0.2073
## x4 -0.1055 -0.0374  0.2073  1.2234
```

```
#Comparison of sample covariance and shrink covariance matrix

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

```
##              x1         x2          x3         x4
## x1  1.18967229 -0.2288990  0.01461313  0.2334858
```

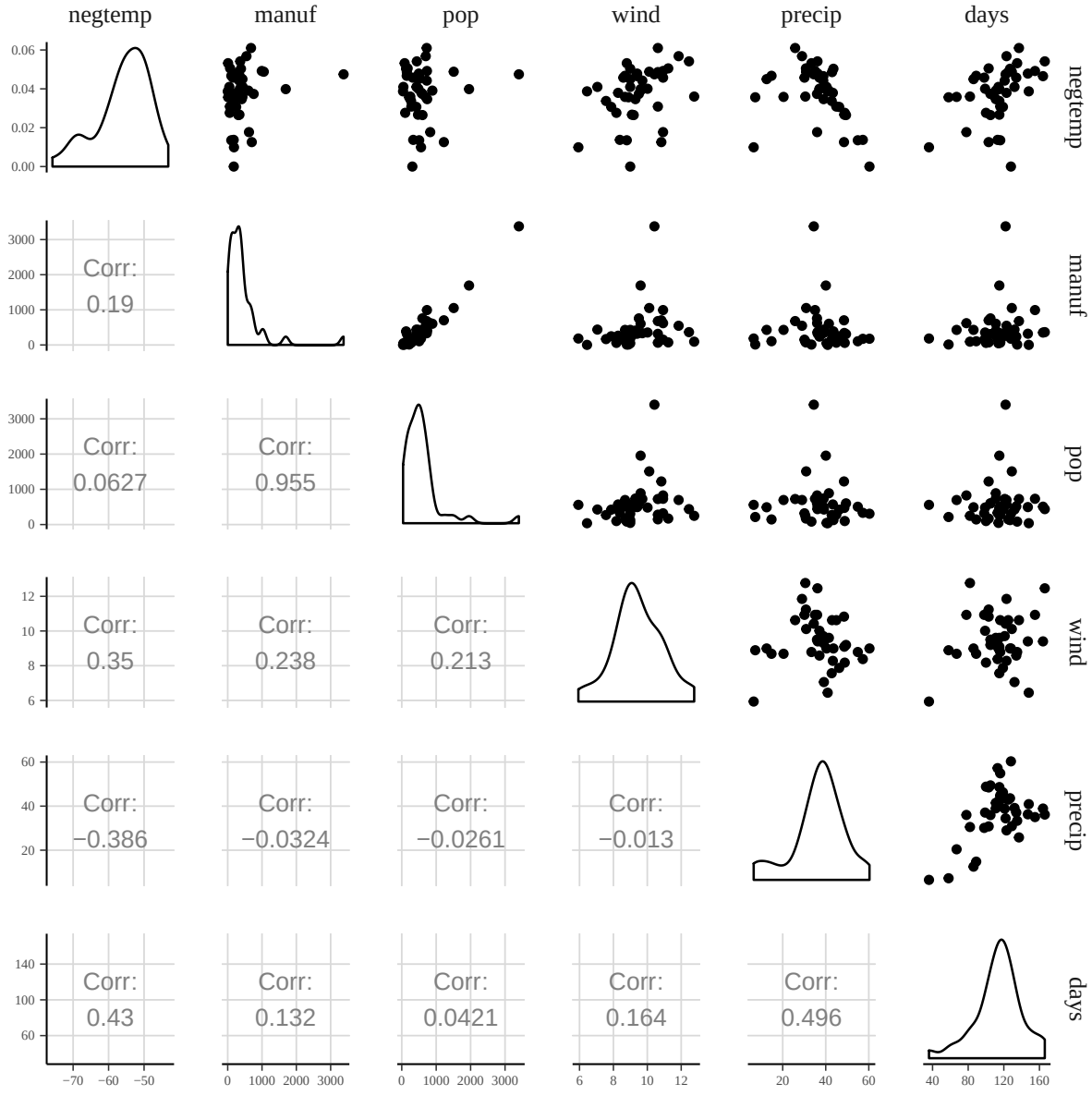
```
## x2 -0.22889895  0.7701068  0.27285759 -0.0514278
## x3  0.01461313  0.2728576  0.71861703 -0.1451502
## x4  0.23348578 -0.0514278 -0.14515015  1.2953711

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

## Estimating optimal shrinkage intensity lambda.var (variance vector): 0.6037
##
## Estimating optimal shrinkage intensity lambda (correlation matrix): 0.4525
##              x1          x2           x3          x4
## x1  1.063018555 -0.12783778  0.008351944  0.11160357
## x2 -0.127837782  0.89676052  0.178027260 -0.02806219
## x3  0.008351944  0.17802726  0.876357065 -0.08105320
## x4  0.111603568 -0.02806219 -0.081053201  1.10490300
## attr(,"lambda")
## [1] 0.4524851
## attr(,"lambda.estimated")
## [1] TRUE
## attr(,"class")
## [1] "shrinkage"
## attr(,"lambda.var")
## [1] 0.6037376
## 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 consits of the SO2 level as the dependent variable and six explanatory variables Two of the explanatory variables are are related to human ecology (pop, manuf) and four others are related to climate (negtemp, wind, precip, days).

Table 1.4: Variable description of air pollution data

Variable Name	Description	Codes/Values
SO2	sulfur dioxide content of air	micrograms 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 percipitation	inches
days	average munber 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.

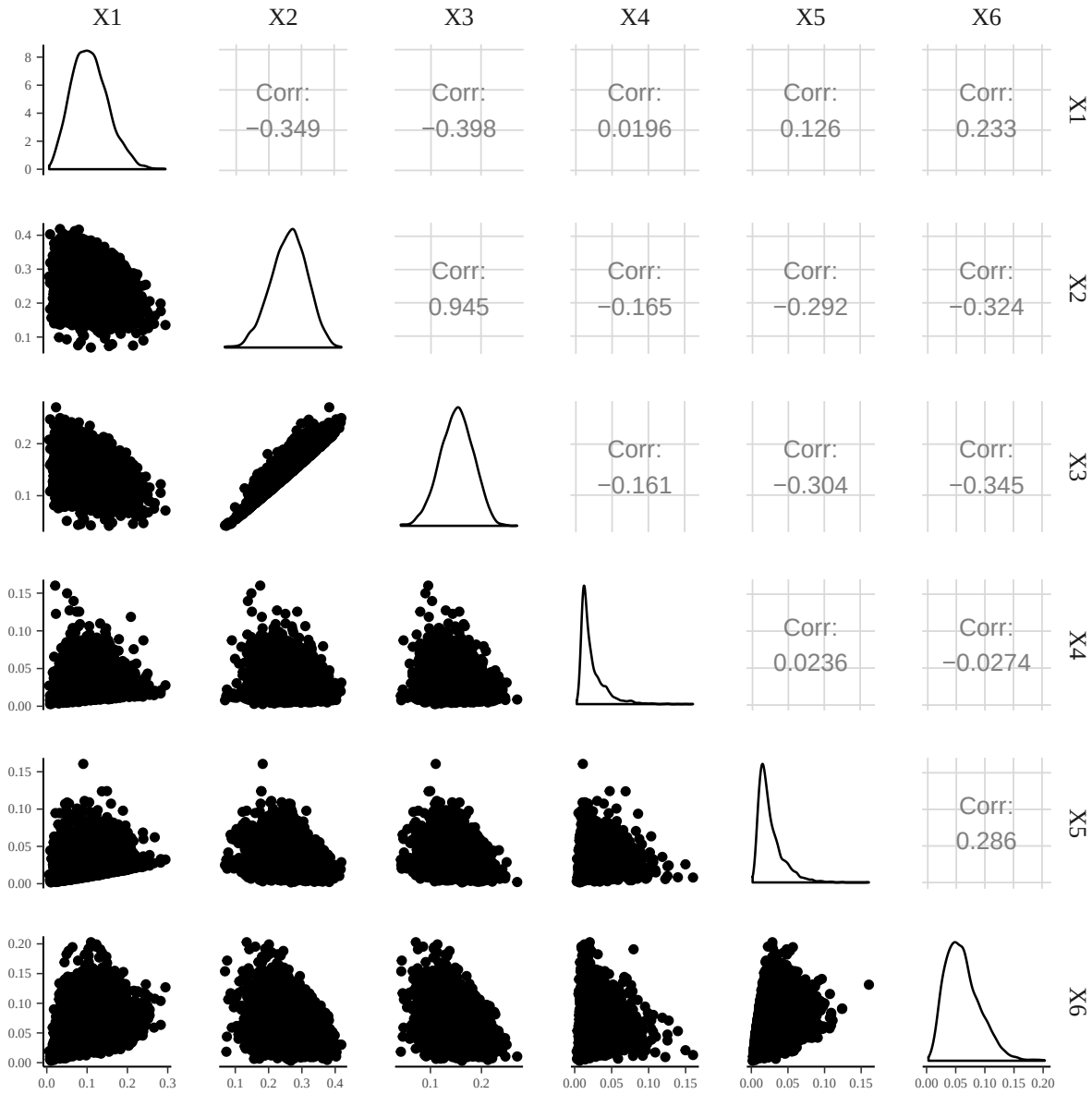
```
usair.lm <- lm(SO2~., data = usair)


#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     0.0355 0.2138
## manuf.lmg   0.2965     0.1900 0.4115
## pop.lmg     0.1661     0.1098 0.2403
## wind.lmg    0.0167     0.0089 0.0728
## precip.lmg  0.0187     0.0081 0.0712
## days.lmg    0.0604     0.0203 0.1389
##
## 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,]   135.83097 1.6205034 0.05559092 -0.024963609 -3.368546
##       [2,]   148.61733 1.9921319 0.05209391 -0.033695296 -3.379457
##       [3,]   134.21933 1.8223345 0.05344065 -0.030226013 -3.321580
##       [4,]   109.72730 0.9034187 0.06408435 -0.036752478 -4.075358
##       [5,]   169.22540 1.8980336 0.07106017 -0.042167151 -3.707558
##       [6,]   148.96289 1.5976024 0.03715969 -0.002440436 -5.353039
##       [7,]   143.43058 1.9582751 0.04677972 -0.023647799 -3.424771
##       [8,]   138.12934 1.9767608 0.04680241 -0.020412746 -3.164154
##       [9,]    31.62113 0.4178926 0.07519204 -0.047230985 -1.559374
##      [10,]    67.76305 0.3463465 0.08937092 -0.059099658 -2.171377
##           parameters
## iterations     precip         days    sigma
```

```
##       [1,] 0.61867890 -0.135979662 13.71905
##       [2,] 0.58114542  0.009231195 13.98130
##       [3,] 0.63406863  0.004519797 13.94855
##       [4,] 0.40657783 -0.096770264 15.49280
##       [5,] 1.09928658 -0.419331341 13.29130
##       [6,] 0.72383580 -0.181001660 17.26865
##       [7,] 0.54949515 -0.026075784 13.07193
##       [8,] 0.55819691 -0.022390552 12.83345
##       [9,] 0.13001486  0.259908026 16.58099
##      [10,] 0.07051633 -0.038521238 14.73657

#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           0.0000000000 0.000000000 0.000000000
## x1             0.1851174789 0.354757878 0.286512432
## x2             0.5159090754 0.246527846 0.342081676
## x3             0.3550144090 0.113969863 0.191841855
## x4             0.0142971366 0.015067873 0.016850648
## x5             0.0002522025 0.001754583 0.006335778
## x6             0.0976191698 0.245652389 0.230862537
## x1 x2          0.6054322912 0.507200513 0.528696856
## x1 x3          0.5099993760 0.445270827 0.450735709
## x1 x4          0.1862057657 0.363098436 0.290265657
## x1 x5          0.2240787525 0.441680841 0.382884967
## x1 x6          0.2050125946 0.425074111 0.363324206
## x2 x3          0.6091586524 0.460226343 0.508703591
## x2 x4          0.5187029528 0.246550343 0.342174538
## x2 x5          0.5174445778 0.249893437 0.351805476
## x2 x6          0.5641635721 0.434853872 0.507672952
## x3 x4          0.3550676397 0.116690195 0.193250427
## x3 x5          0.3560037606 0.116542643 0.200135346
## x3 x6          0.4377402491 0.346152318 0.405710107
## x4 x5          0.0146011543 0.016958947 0.023458917
## x4 x6          0.1024127968 0.247415066 0.233531366
## x5 x6          0.1232918763 0.300851144 0.264298788
## x1 x2 x3       0.6456106231 0.584255292 0.592992648
```

```
## x1 x2 x4       0.6299578857 0.535105745 0.552953372
## x1 x2 x5       0.6337285916 0.584173288 0.611906572
## x1 x2 x6       0.6163720923 0.566495167 0.591070960
## x1 x3 x4       0.5332435173 0.469598807 0.472349588
## x1 x3 x5       0.5494288477 0.532562443 0.547631148
## x1 x3 x6       0.5272929582 0.512954480 0.523845393
## x1 x4 x5       0.2278603389 0.459725409 0.394161398
## x1 x4 x6       0.2062563925 0.434216088 0.367645926
## x1 x5 x6       0.2247710032 0.444241717 0.385372831
## x2 x3 x4       0.6138551538 0.460582565 0.509634363
## x2 x3 x5       0.6111122982 0.464530328 0.519795613
## x2 x3 x6       0.6282292054 0.559388261 0.600408251
## x2 x4 x5       0.5202160480 0.249919029 0.351888289
## x2 x4 x6       0.5712100901 0.437947107 0.512078169
## x2 x5 x6       0.5710755760 0.469718024 0.523015285
## x3 x4 x5       0.3560535555 0.119303587 0.201596553
## x3 x4 x6       0.4406228328 0.346764294 0.407037274
## x3 x5 x6       0.4543353180 0.393715673 0.431450645
## x4 x5 x6       0.1259845839 0.301113141 0.265303596
## x1 x2 x3 x4    0.6648538323 0.604437815 0.610664917
## x1 x2 x3 x5    0.6641871676 0.638957779 0.655086937
## x1 x2 x3 x6    0.6516045505 0.627040581 0.639878133
## x1 x2 x4 x5    0.6665049380 0.626816987 0.650558389
## x1 x2 x4 x6    0.6410854774 0.594866718 0.615773494
## x1 x2 x5 x6    0.6360323883 0.585606086 0.613042441
## x1 x3 x4 x5    0.5829666007 0.572906477 0.585314068
## x1 x3 x4 x6    0.5510641756 0.538350903 0.546506715
## x1 x3 x5 x6    0.5513361785 0.534281256 0.549041548
## x1 x4 x5 x6    0.2292035985 0.460591928 0.395268558
## x2 x3 x4 x5    0.6157810155 0.464875234 0.520697018
## x2 x3 x4 x6    0.6358059358 0.562980536 0.605322254
## x2 x3 x5 x6    0.6294242541 0.573880076 0.604750245
## x2 x4 x5 x6    0.5794657832 0.474944249 0.529029029
## x3 x4 x5 x6    0.4588355009 0.395931365 0.434254757
## x1 x2 x3 x4 x5 0.6903033120 0.671422793 0.685128213
## x1 x2 x3 x4 x6 0.6712607148 0.648350860 0.658656123
## x1 x2 x3 x5 x6 0.6666325206 0.640247140 0.656109414
## x1 x2 x4 x5 x6 0.6728797311 0.626825334 0.650558535
## x1 x3 x4 x5 x6 0.5887423878 0.572960223 0.585334637
## x2 x3 x4 x5 x6 0.6375962715 0.578877182 0.610545875
## all            0.6963888732 0.671452014 0.685134958

# prepare data frame for LMG values
```

```
LMG.Vals.I<-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[,i]=obj.Gelman$IJ[,1]
}

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

##       2.5%        50%      97.5%
## 0.03089074 0.10414155 0.20513745

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

##      2.5%       50%     97.5%
## 0.1492509 0.2632596 0.3615762

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

##       2.5%        50%      97.5%
## 0.08674675 0.15221081 0.21173554

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

##        2.5%         50%       97.5%
## 0.007556802 0.017508171 0.071477479

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

##        2.5%         50%       97.5%
## 0.006626489 0.020800713 0.070602786

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

##       2.5%        50%      97.5%
## 0.01776118 0.05803545 0.13121868

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

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

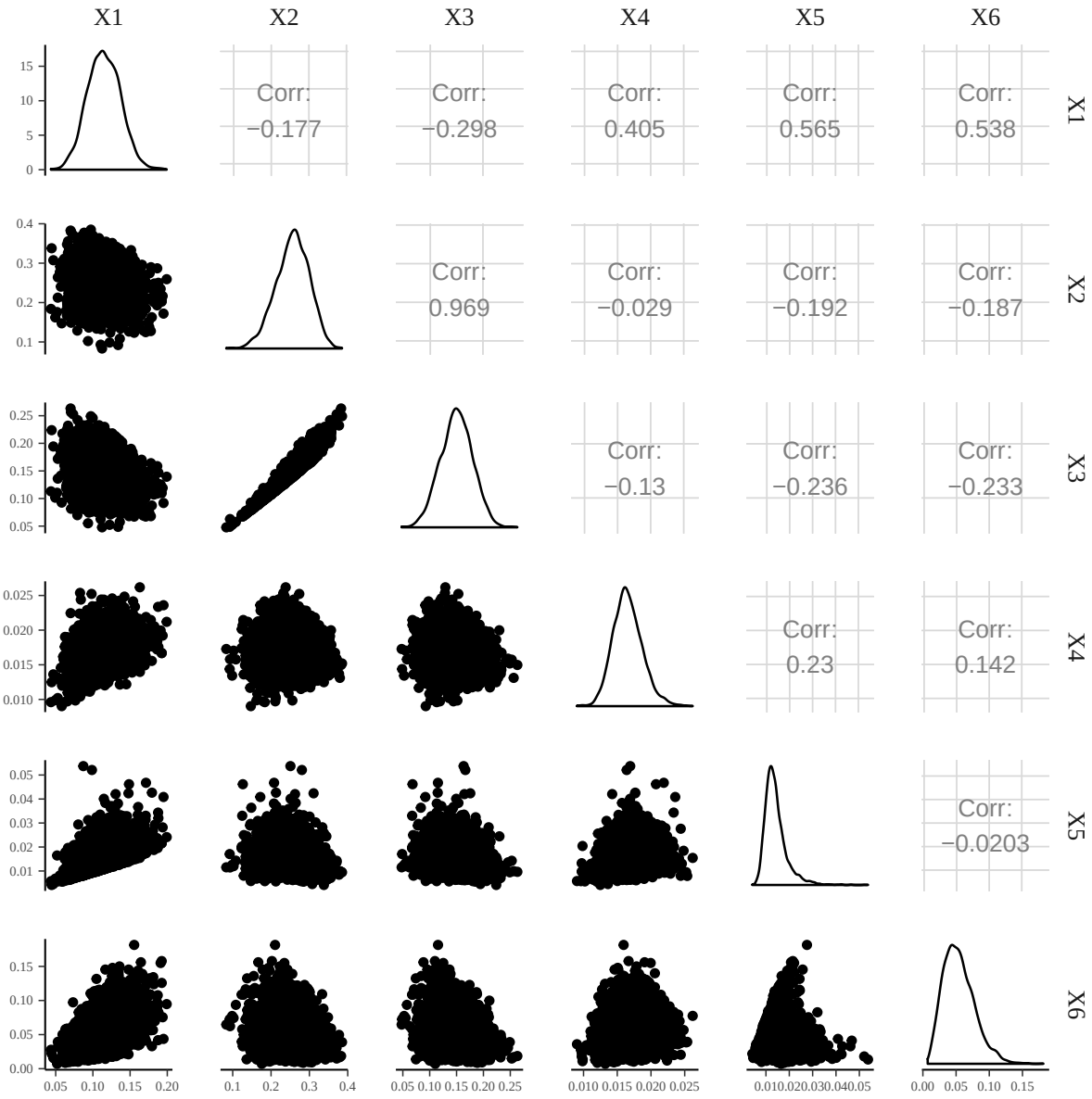



```
#use prior knowledge
my_prior <- normal(location = c(1, 0.05,-0.03,-3, 0, 0), scale = c(0.1, 0.01,0.01,0.1, 0.1, 0.1


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,]    93.15510 0.9598408 0.05076080 -0.02230947 -2.974473
##       [2,]    85.33896 0.9767056 0.04873335 -0.02762700 -2.993646
##       [3,]   115.76574 1.0501914 0.04800659 -0.02754002 -2.960514
##       [4,]   105.13833 1.0937662 0.06237204 -0.03862311 -2.755087
##       [5,]   100.17739 0.9218288 0.06397271 -0.03741000 -3.209142
##       [6,]    94.81891 0.8699693 0.06460313 -0.03974537 -3.058839
##       [7,]   107.25037 1.1169629 0.04871286 -0.02439039 -2.983201
##       [8,]    85.37864 0.8471359 0.05783901 -0.03187420 -2.868974
##       [9,]   104.06738 0.9644024 0.05936805 -0.03575935 -3.024624
##      [10,]    92.09087 1.0109323 0.05776583 -0.03223990 -2.961939
##           parameters
## iterations      precip        days    sigma
##       [1,]  0.14680578  0.01154576 10.73355
##       [2,]  0.30231176  0.07928726 11.73207
##       [3,] -0.06167479 -0.00771524 15.35028
##       [4,]  0.14574409 -0.01068613 13.39733
##       [5,] -0.05972290  0.10561934 13.20576
##       [6,]  0.04507632  0.07356100 13.71082
##       [7,]  0.16243033 -0.03046557 15.03457
##       [8,]  0.13903104  0.09494624 15.20654
##       [9,] -0.06792343  0.05425036 15.97530
##      [10,]  0.27630906  0.04133327 12.82158
```

```
#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.I<-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[,i]=obj.Gelman$IJ[,1]
```

```
}

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

##      2.5%       50%     97.5%
## 0.0725409 0.1143731 0.1584069

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

##      2.5%       50%     97.5%
## 0.1656552 0.2578780 0.3334933

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

##       2.5%        50%      97.5%
## 0.09132594 0.15029884 0.20709797

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

##       2.5%        50%      97.5%
## 0.01266590 0.01641026 0.02130490

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

##        2.5%         50%       97.5%
## 0.007922485 0.013181413 0.026273074

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

##       2.5%        50%      97.5%
## 0.01718799 0.05162126 0.10892648

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

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


X1
X2
X3
X4
X5
X6
Corr: −0.177
Corr: −0.298
Corr: 0.405
Corr: 0.565
Corr: 0.538
Corr: 0.969
Corr: −0.029
Corr: −0.192
Corr: −0.187
Corr: −0.13
Corr: −0.236
Corr: −0.233
Corr: 0.23
Corr: 0.142
Corr: −0.0203