Fitting Generalized Bayesian Factor Models with stan

Lucas Deschamps

28/05/2018

Standard caveat : I am not a mathematician nor a statistician, only an ecologist who loves discover and share new bayesian statistical approaches with his somehow naive view of statistics. I hope that these new statistical methods would help to answer topical question in ecology. Feel free to comment and give your impressions :)

Introduction

Multivariate datasets contain a lot of information on their own. Community matrices, representing the abundances or presence of species (in column) at each sites (in row), are typical examples of multivariate datasets of interest in ecology. A lot of procedures tried to exploit the underlying information of those matrices to understand how the world is structured. Some were based on linear algebra, creating new axis maximazing variance (PCA) or preserving chi-squared distances (CA). Other algorithmic approaches aimed to attribute new coordinates to each data point, preserving whatever distances between sites or species by iterations (NMDS). By contrast, Factor Analysis aimed to discover underlying, abstract or unobserved causes of abundance patterns by exploiting the covariances between variables. The basic idea seems intuitive : covariances between variables (in that case, species), might be the consequence of common factors affecting simultaneously different species.

Problematic

Modelling covariances between variables is a difficult task, because the number of parameters to estimate grows quadratically. Modelling the covariance matrix between 7 species implies to estimate 49 parameters, and for 80 species, 6400 parameters! The number of observations needed to achieve such a task would quickly exceed the amount of data an ecologist can harvest in his own professional life... The aim of Factor Analysis is to estimate the underlying causes of the covariance of interest. As such, it allows to reduces drastically the number of dimensions to explore : covariances between 80 species are possibly explained by 4 or 5 underlying environmental gradients, and we virtually do not need more than them to understand how a community matrix is structured. However classical Factor Analysis are not models. As such, they do not allow to discover the process generating data. They do not allow predictions, test of assumptions, comparisons. They cannot afford dataset with complex structure, what multilevel models can do.

Latent variable models : Bayesian factor analysis

The concepts behind factor analysis can however be used in model formulation, in the form of latent variables. In this kind of model, a $n\ x\ s$ multivariate dataset $\boldsymbol{Y}$, containing the abundance of species j at plot i can be decomposed into a $n\ x\ d$ Factor Matrix $\boldsymbol{F}$, containing the score of factor f for each plot i and a $s\ x\ d$ loading matrix $\boldsymbol{L}$ containing the coefficient linking each species to each factor. The multiplication is computed using the transpose of $\boldsymbol{L}$.

$$Y \begin{bmatrix} y_{11} & y_{12} & ... & y_{1s} \\ y_{21} & y_{22} & ... & y_{2s} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n1} & y_{n2} & ... & y_{ns} \end{bmatrix} = F \begin{bmatrix} f_{11} & f_{12} & ... & f_{1d} \\ f_{21} & f_{22} & ... & f_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ f_{n1} & f_{n2} & ... & f_{nd} \end{bmatrix} L' \begin{bmatrix} \lambda_{11} & \lambda_{12} & ... & \lambda_{1s} \\ \lambda_{21} & \lambda_{22} & ... & \lambda_{2s} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{d1} & \lambda_{d2} & ... & \lambda_{ds} \end{bmatrix}$$

But where is the covariance? We said that covariances can be approached by the response of different variables to common latent factors. The response of variables to the latent factor are the loading coefficients : we will use them to estimate the covariances between species.To do that, we need to set the variance of factors to a unit scale. This will allow to isolate the variance of each variables in the loadings and to subsequently compute covariances, as greatly explained in (Tryfos 1998)(http://www.yorku.ca/ptryfos/f1400.pdf). Thus, the factor scores will be defined by a multivariate Normal distribution with means of 0 and a correlation matrix with unit diagonal :

$$F \sim MultiNormal(0, \Sigma)$$
$$diag(\Sigma) = 1$$

Under this condition, the s x s estimated covariance matrix $\boldsymbol{C}$ can be obtained by multiplying the loading matrix by its transpose :

$$C \begin{bmatrix} c_{11} & c_{12} & ... & c_{1s} \\ c_{21} & c_{22} & ... & c_{2s} \\ \vdots & \vdots & \ddots & \vdots \\ c_{s1} & c_{s2} & ... & c_{ss} \end{bmatrix} = L \begin{bmatrix} \lambda_{11} & \lambda_{12} & ... & \lambda_{1d} \\ \lambda_{21} & \lambda_{22} & ... & \lambda_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{s1} & \lambda_{s2} & ... & \lambda_{sd} \end{bmatrix} L' \begin{bmatrix} \lambda_{11} & \lambda_{12} & ... & \lambda_{1s} \\ \lambda_{21} & \lambda_{22} & ... & \lambda_{2s} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{d1} & \lambda_{d2} & ... & \lambda_{ds} \end{bmatrix}$$

Now begin the difficulties. In such a model, both variables (factor scores) and coefficients (loadings) are unknown and have to be estimated. This leads to a model which is not identified : multiple points in the parameter space can lead to the same joint probability distribution. In crude words, there will be more than one combination of parameters which will describe equally the observed data. Hopefully, Farouni (2015) gave us the constraints to allow identifiability (see his blog post to obtain original sources, and also (forum 2017; forum 2015)) :

- Fix the upper diagonal of L to zero
- Restrain the diagonal of L to be positive
- Provide close initial values to the algorithm

The first two constraints will reduce drastically the number of possible solutions, and the second will avoid the algorithm get lost in secondary peaks on the posterior surface.

The complete model

This approach of community modelling is known as Joint Species Distributions Models (Warton et al. 2015; Hui 2016), and aims to reveal patterns in community structure. To limit the influence of global differences in abundances between plots, we will add a hierarchical intercept for each sites. Thoses parameters will be distributed normally with estimated standard-deviation. Because we will deal with abundances as count data, we will describe data as following a poisson distribution. The final model is thus defined as follow :

$$Y_{ij} \sim Poisson(\mu_{ij})$$
$$\mu_{ij} = \alpha + \delta_i + \Upsilon_{ij}$$
$$\Upsilon = FL'$$
$$\delta_i \sim normal(0, \sigma_d)$$

Concerning priors for loadings, I have tried a lot of different distributions and parameters, hierarchical or not (gamma, student-t with 3 degre of freedom, normal, and even a mix of normal and chi-squared, inspired from the barlett decomposition of the wishart distribution (forum 2018)). However, the solution providing the best results is to use cauchy distribution, as Rick Farouni did (Farouni 2015). Unlike him, I did not put hierarchical prior on the diagonal loadings because they are only three, but one could if using suffficiently informative priors. The complete set of priors are as follow, *Vech* being the half-vectorization operator, selecting the lower diagonal loadings of loading matrices.

$$\begin{aligned} \alpha &\sim student(3,0,5) & \sigma_d &\sim HalfCauchy(0,1) \\ \delta_0 &\sim Normal(0,\sigma_d) & \mu_l &\sim Cauchy(0,1) \\ diag(L) &\sim HalfCauchy(0,2.5) & \tau_l &\sim HalfCauchy(0,1) \\ Vech(L) &\sim Cauchy(\mu_l,\tau_l) & \Sigma &\sim LKJ(1) \\ F &\sim MultiNormal(0,\Sigma) \end{aligned}$$

Simulated data

Let's simulate some "ideal" data, based on three factors, 10 species and 100 plots. The steps are : - Sample factor scores from a multivarial normal, with unit diagonal and correlation matrix sampled from a LKJ distribution (eta = 1 give an equal probability between -1 and 1, see ?rlkjcorr or here) - Sample loadings from a multivariate normal, with correlation matrix sampled from an LKJ distribution (eta = 0.5 concentrates the density around correlations of -1 and 1)

- Set the constraints on loadings : zero on upper diagonal and positive diagonal
- Sample an intercept
- Sample a deviation for each plot
- Compute the mean abundance of each species at each plot and simulate observed abundance by sampling from a poisson distribution

```
## Empty the environment
rm(list = ls())
## Set seed for reproducibility
set.seed(42)
library(rethinking) ## Implements multivariate normal and LKJ distributions
library(rstan) ## Makes the link with stan to sample the posterior
library(vegan) ## Contains principal ordinations methods
library(corrplot) ## Correlation plot
library(parallel) ## Will be used to detect the number of cores
```

```
N <- 100 ## Number of plots
S <- 10 ## Number of species
D <- 3 ## Number of factors

# Sample the factors (N x D matrix)
## Sample a correlation matrix for factors (sd = 1)
F_corr <- rlkjcorr(1, D, eta = 1)
diag(F_corr)
```

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

```
## Sample the factor scores from a multivariate normal (mean = 0)
FS <- rmvnorm2(N, Mu = rep(0, length(diag(F_corr)), sigma = diag(F_corr),
                           Rho = F_corr))
```

```r
# Sample the loadings (D x S matrix)
## Sample a matrix from a multivariate normal
L_corr <- rlkjcorr(1, S, eta = 0.5) ## eta = 0.5 concentrate the density around -1 and 1
Lambda <- rmvnorm2(D, Mu = rep(0, length(diag(L_corr))), sigma = rep(0.8, length(diag(L_corr))),
                              Rho = L_corr)
head(Lambda)
```

```
##            [,1]       [,2]       [,3]      [,4]       [,5]        [,6]
## [1,] -1.1589707 -0.2702740 -1.2610899  1.657322  0.2840215 -0.56693651
## [2,]  0.2336548  1.2888008  0.2792652 -1.398169 -1.4714822  0.03439578
## [3,] -0.3937239  0.6857522  1.4282712 -1.726076 -1.8651243 -0.39597793
##            [,7]       [,8]        [,9]      [,10]
## [1,] -0.3895683 -0.5318822  0.06484861 -0.7761556
## [2,] -0.2926818  1.2627511 -1.64374959  0.4498957
## [3,] -1.7826745  1.7635332 -0.21599092  0.9178326
```

```r
Lt <- t(Lambda)
head(Lt)
```

```
##            [,1]        [,2]       [,3]
## [1,] -1.1589707  0.23365484 -0.3937239
## [2,] -0.2702740  1.28880076  0.6857522
## [3,] -1.2610899  0.27926522  1.4282712
## [4,]  1.6573223 -1.39816935 -1.7260764
## [5,]  0.2840215 -1.47148218 -1.8651243
## [6,] -0.5669365  0.03439578 -0.3959779
```

```r
## Force the diag to be positive
for(i in 1:D) Lt[i,i] <- abs(Lt[i,i])
diag(Lt)
```

```
## [1] 1.158971 1.288801 1.428271
```

```r
## Force the upper-diag to zero
L <- Lt
for(i in 1:(D-1)) {
  for(j in (i+1):(D)){
    L[i,j] = 0;
  }
}
head(L)
```

```
##            [,1]        [,2]       [,3]
## [1,]  1.1589707  0.00000000  0.0000000
## [2,] -0.2702740  1.28880076  0.0000000
## [3,] -1.2610899  0.27926522  1.4282712
## [4,]  1.6573223 -1.39816935 -1.7260764
## [5,]  0.2840215 -1.47148218 -1.8651243
## [6,] -0.5669365  0.03439578 -0.3959779
```

```r
# Sample a global intercept
alpha <- rnorm(1)
# Sample a deviation parameter per row (plot)
d0 <- rnorm(N)
# Compute the final matrix of predictor
Mu <- exp(alpha + d0 + FS %*% t(L))
```

```
head(Mu)
```

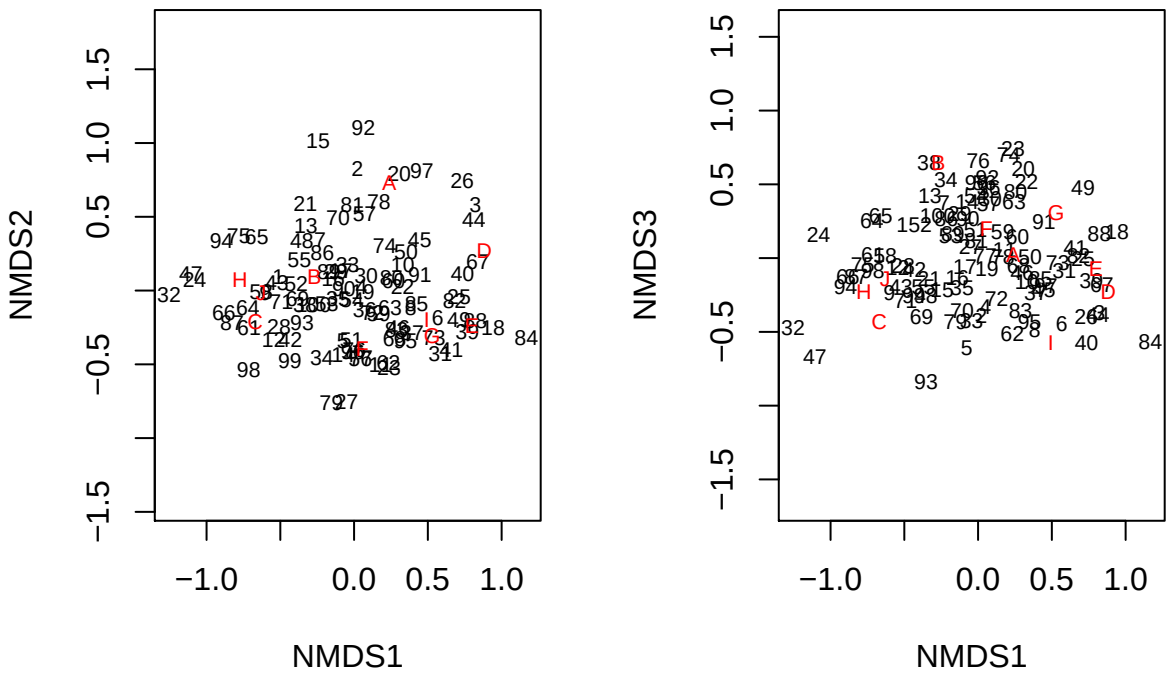
```
##              [,1]       [,2]         [,3]         [,4]        [,5]
## [1,]  2.42253029 19.7774257   4.17267400    0.3269167   0.3430220
## [2,] 15.25708243  0.7861532   0.73090628    4.7889380   0.2509665
## [3,] 46.34546757  0.2946353   0.08343032 1633.9408039  81.3763210
## [4,]  0.98370160  2.7012901   1.08092980    0.6179191   0.7368242
## [5,]  0.06922517  0.1327959 142.92711417    0.0572304   2.1875911
## [6,]  4.17347095  0.6513279   4.16964097   58.1628928 103.4255582
##           [,6]        [,7]       [,8]        [,9]      [,10]
## [1,] 3.1817982  2.17168148 16.5435331   0.2314961  5.3834151
## [2,] 0.2787542  0.06664557  4.6374977   1.4019836  0.9886410
## [3,] 0.9533330  3.31563065  0.1027254  39.5438135  0.2300294
## [4,] 1.4163731  1.66632386  1.6665435   0.4254916  1.3057475
## [5,] 3.6977603  0.82216183  2.9085246  52.5875729 13.2483568
## [6,] 7.1333703 15.35466279  0.5459093 113.2282520  2.8948641
```

```
# Compute the stochastic observations
Y <- matrix(nrow = N, ncol = S)
for(i in 1:N){
  for(j in 1:S){
    Y[i,j] <- rpois(1, Mu[i,j])
  }
}
#summary(Y)
colnames(Y) <- LETTERS[1:S] ## Attribute names to species

pairs(log(Y+1), col = scales::alpha('black', 0.5), pch = 16) ## Scatterplot matrix
```


```
### Examine data structure with NMDS
NMDS <- metaMDS(Y, k = 3, trymax = 100)
```

```
par(mfrow = c(1,2))
plot(NMDS, choices = c(1,2), type = "t")
plot(NMDS, choices = c(1,3), type = "t");par(mfrow = c(1,1))
```



Here is the stan code of the model described above. However, Euclidian Hamiltonian Monte-Carlo algorithms like the No-U-Turn sampler implemented in *stan* might have hard time to sample from the heavy tails of cauchy distributions, especially with as few data as we simulated. I let there the simple code, clearer to read, and potentially efficient if one have a lot of data points. However, we will sample from a better parametrization presented at the end of the post.

The data part contains the integers providing the dimensions of the response matrix, the response matrix and the number of factor chosen.

```
data{
  int N; // sample size
  int S; // number of species
  int D; // number of factors
  int<lower=0> Y[N,S]; // data matrix of order [N,S]
}
```

The transformed data part fix the mean and standard-deviation of factors to 0 and 1, respectively. In this part is also computed the number of lower triangular loadings. The operations in this part are made only once.

```
transformed data{
  int<lower=1> M;
  vector[D] FS_mu; // factor means
  vector<lower=0>[D] FS_sd; // factor standard deviations
  M = D*(S-D)+ D*(D-1)/2;  // number of lower-triangular, non-zero loadings
  for (m in 1:D) {
    FS_mu[m] = 0; //Mean of factors = 0
    FS_sd[m] = 1; //Sd of factors = 1
  }
}
```

In the parameter part, we define every parameter which will be sampled. Hyperparameters are the ones describing hierarchical distributions, such as the standard-deviation of the row deviations.

```
parameters{
  //Parameters
  real alpha; //Global intercept
  vector[N] d0; //Row deviations
  vector[M] L_lower; //Lower diagonal loadings
  vector<lower=0>[D] L_diag; //Positive diagonal elements of loadings
  matrix[N,D] FS; //Factor scores, matrix of order [N,D]
  cholesky_factor_corr[D] Rho; //Correlation matrix between factors
  //Hyperparameters
  real<lower=0> sigma_d; //Sd of the row intercepts
  real mu_low; //Mean of lower diag loadings
  real<lower=0> tau_low; //Scale of lower diag loadings
}
```

The transform parameter block realized the biggest jobs. In this part :

- the final unit diagonal Σ is computed
- the loadings are inserted in the loadings matrix, defined as a cholesky decomposition of a covariance matrix (lower triangular matrix with positive diagonal)
- the predictor for the poisson distribution, μ is computed

```
transformed parameters{
  cholesky_factor_cov[S,D] L; //Final matrix of laodings
  matrix[D,D] Ld; // cholesky decomposition of the covariance matrix between factors
  //Predictors
  matrix[N,S] Ups; //intermediate predictor
  matrix<lower=0>[N,S] Mu; //predictor

  // Correlation matrix of factors
  Ld = diag_pre_multiply(FS_sd, Rho); //Fs_sd fixed to 1, Rho estimated

  {
    int idx2; //Index for the lower diagonal loadings
    idx2 = 0;

    // Constraints to allow identifiability of loadings
     for(i in 1:(D-1)) { for(j in (i+1):(D)){ L[i,j] = 0; } } //0 on upper diagonal
     for(i in 1:D) L[i,i] = L_diag[i]; //Positive values on diagonal
     for(j in 1:D) {
       for(i in (j+1):S) {
         idx2 = idx2+1;
         L[i,j] = L_lower[idx2]; //Insert lower diagonal values in loadings matrix
       }
     }
  }

  // Predictors
  Ups = FS * L';
  for(n in 1:N) Mu[n] = exp(alpha + d0[n] + Ups[n]);

}
```

The model block contains all prior declarations and the computation of the likelihood.

```
model{
  // Hyperpriors
  sigma_d ~ cauchy(0,1); //Sd of the plot deviations
  mu_low ~ cauchy(0,1); //Mu of lower diag loadings
  tau_low ~ cauchy(0,1); //Scales of the lower diag loadings

  // Priors
  alpha ~ student_t(3,0,5); //Weakly informative prior for global intercept
  d0 ~ normal(0,sigma_d); //Regularizing prior for row deviations
  L_diag ~ cauchy(0,2.5); //Weakly informative prior for diagonal loadings
  L_lower ~ cauchy(mu_low, tau_low); //Hierarchical prior for lower diag loadings
  Rho ~ lkj_corr_cholesky(1); //Uninformative prior for Rho

  for(i in 1:N){
    Y[i,] ~ poisson(Mu[i,]); //Likelihood
    FS[i,] ~ multi_normal_cholesky(FS_mu, Ld); //Regularizing prior for factor scores
  }
}
```

The generated quantity block will be usefull to compute the estimated covariance matrix and correlation matrix, compute the log-likelihood for each observation (required for model diagnotics and comparison) and generate stochastic predicted values.

```
generated quantities{
  matrix[S,S] cov_L;
  matrix[S,S] cor_L;
  matrix[N,S] Y_pred;
  matrix[N,S] log_lik1;
  vector[N*S] log_lik;

  cov_L = multiply_lower_tri_self_transpose(L); //Create the covariance matrix

  // Compute the correlation matrix from de covariance matrix
  for(i in 1:S){
    for(j in 1:S){
      cor_L[i,j] = cov_L[i,j]/sqrt(cov_L[i,i]*cov_L[j,j]);
    }
  }

  //Compute the log-likelihood and predictions for each observation
  for(n in 1:N){
    for(s in 1:S){
      log_lik1[n,s] = poisson_lpmf(Y[n,s] | Mu[n,s]);
      Y_pred[n,s] = poisson_rng(Mu[n,s]);
    }
  }
  log_lik = to_vector(log_lik1); //Tranform in a vector usable by loo package
}
```

Now it is time to sample from the posterior... Reparametrized code is downloadable as a .stan file here. It takes more than 15 minutes to fit on my computer, be patient...

```
## Define the data used by stan
D_stan <- list(Y = Y, N = nrow(Y), S = S, D = D)
```

```
## Fit the model
BFS <- stan("Poisson_BFS_reparam.stan", data = D_stan,
            pars = c("Mu", "d0_raw", "L_lower_unif", "L_diag_unif", "sigma_d_unif",
                     "mu_low_unif", "tau_low_unif", "log_lik1"), include = F,
            iter = 2000, init = 0, chains = parallel::detectCores()-1,
            cores = parallel::detectCores()-1,
            control = list(max_treedepth = 12))
```

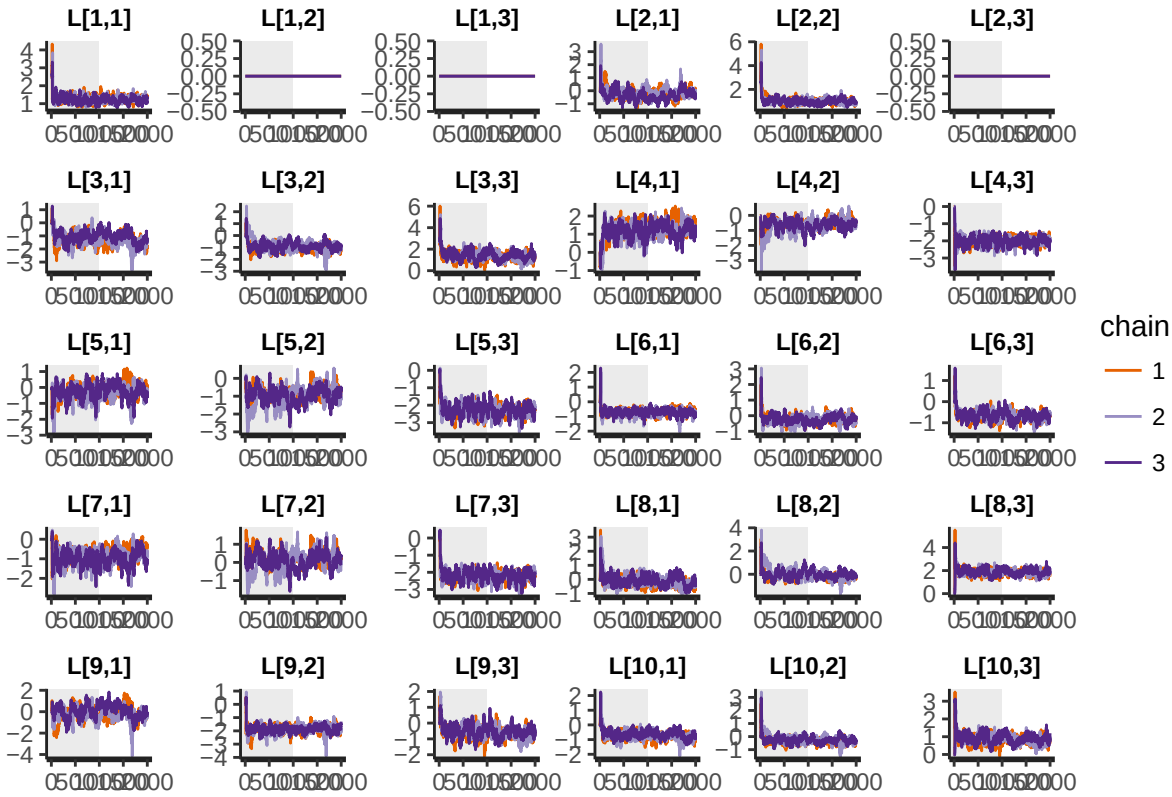
Loadings effective sample sizes are not great, but rhats are corrects for all loadings (<1.2)

```
print(BFS, pars = "L")
```

```
## Inference for Stan model: Poisson_BFS_Reparam.
## 3 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=3000.
##
##          mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## L[1,1]   1.24    0.01 0.15  0.94  1.13  1.23  1.34  1.55   116 1.04
## L[1,2]   0.00    0.00 0.00  0.00  0.00  0.00  0.00  0.00  3000  NaN
## L[1,3]   0.00    0.00 0.00  0.00  0.00  0.00  0.00  0.00  3000  NaN
## L[2,1]  -0.24    0.08 0.35 -0.91 -0.48 -0.23 -0.01  0.46    19 1.17
## L[2,2]   0.99    0.03 0.18  0.64  0.87  0.99  1.09  1.37    41 1.07
## L[2,3]   0.00    0.00 0.00  0.00  0.00  0.00  0.00  0.00  3000  NaN
## L[3,1]  -1.29    0.11 0.47 -2.19 -1.60 -1.28 -0.96 -0.42    19 1.18
## L[3,2]  -0.97    0.05 0.33 -1.63 -1.15 -0.96 -0.78 -0.35    51 1.07
## L[3,3]   1.30    0.05 0.35  0.63  1.07  1.29  1.52  2.01    41 1.07
## L[4,1]   1.33    0.07 0.38  0.55  1.09  1.33  1.58  2.04    27 1.14
## L[4,2]  -0.52    0.05 0.30 -1.12 -0.72 -0.52 -0.32  0.10    37 1.07
## L[4,3]  -2.00    0.02 0.21 -2.45 -2.13 -1.98 -1.86 -1.63    95 1.02
## L[5,1]  -0.16    0.08 0.47 -1.13 -0.45 -0.14  0.13  0.72    31 1.13
## L[5,2]  -0.83    0.07 0.41 -1.58 -1.12 -0.82 -0.56  0.05    37 1.06
## L[5,3]  -2.28    0.03 0.30 -2.95 -2.46 -2.26 -2.07 -1.72    76 1.03
## L[6,1]  -0.69    0.03 0.22 -1.18 -0.80 -0.67 -0.55 -0.32    58 1.07
## L[6,2]  -0.20    0.04 0.25 -0.69 -0.38 -0.18 -0.03  0.29    43 1.07
## L[6,3]  -0.69    0.03 0.19 -1.10 -0.80 -0.68 -0.56 -0.31    55 1.06
## L[7,1]  -0.80    0.07 0.38 -1.65 -1.05 -0.77 -0.53 -0.13    33 1.08
## L[7,2]   0.23    0.08 0.44 -0.60 -0.09  0.24  0.50  1.17    33 1.09
## L[7,3]  -2.19    0.03 0.27 -2.80 -2.34 -2.16 -2.01 -1.75    86 1.03
## L[8,1]  -0.24    0.06 0.32 -0.79 -0.47 -0.25 -0.02  0.42    27 1.10
## L[8,2]  -0.14    0.05 0.29 -0.68 -0.31 -0.15  0.02  0.49    28 1.09
## L[8,3]   1.81    0.03 0.23  1.37  1.65  1.80  1.97  2.30    56 1.04
## L[9,1]  -0.10    0.15 0.64 -1.26 -0.52 -0.13  0.31  1.14    19 1.20
## L[9,2]  -1.82    0.04 0.29 -2.42 -1.99 -1.81 -1.64 -1.31    60 1.01
## L[9,3]  -0.60    0.05 0.33 -1.30 -0.80 -0.61 -0.40  0.10    49 1.06
## L[10,1] -0.75    0.05 0.25 -1.28 -0.91 -0.74 -0.58 -0.28    27 1.12
## L[10,2] -0.32    0.03 0.23 -0.76 -0.45 -0.32 -0.18  0.12    51 1.07
## L[10,3]  0.83    0.03 0.22  0.41  0.69  0.83  0.98  1.29    45 1.06
##
## Samples were drawn using NUTS(diag_e) at Thu May 24 19:16:04 2018.
## 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).
```

And traceplots look good!

```
traceplot(BFS, pars = "L", inc_warmup = T)
```

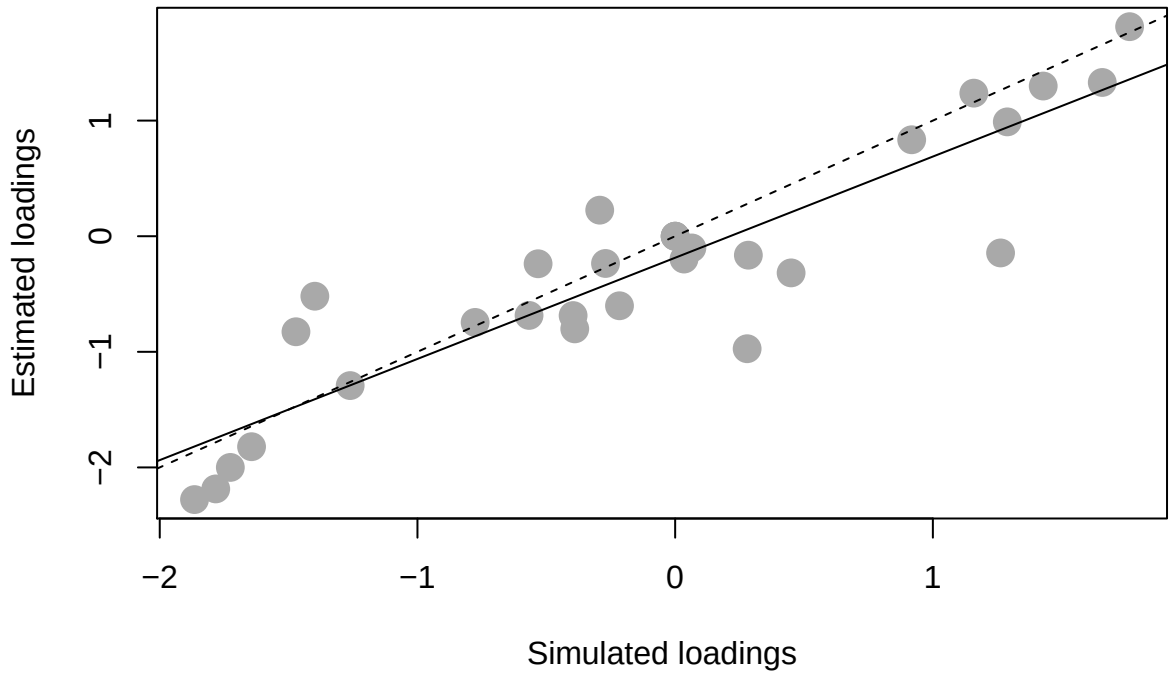


Now, let's see if we were able to recover the loadings!

```
## Compare estimated to simulated loadings
mod_L <- summary(BFS, pars = "L")$summary # Extract the mean loadings values
head(mod_L)
```

```
##              mean    se_mean        sd       2.5%        25%        50%
## L[1,1]  1.2374902 0.01433126 0.1541706  0.9436806  1.1328476  1.2348595
## L[1,2]  0.0000000 0.00000000 0.0000000  0.0000000  0.0000000  0.0000000
## L[1,3]  0.0000000 0.00000000 0.0000000  0.0000000  0.0000000  0.0000000
## L[2,1] -0.2359411 0.08137081 0.3543481 -0.9134629 -0.4789247 -0.2301793
## L[2,2]  0.9889739 0.02790587 0.1784865  0.6403309  0.8741270  0.9887844
## L[2,3]  0.0000000 0.00000000 0.0000000  0.0000000  0.0000000  0.0000000
##                75%     97.5%      n_eff     Rhat
## L[1,1]  1.33682212 1.5482537  115.72683 1.037407
## L[1,2]  0.00000000 0.0000000 3000.00000      NaN
## L[1,3]  0.00000000 0.0000000 3000.00000      NaN
## L[2,1] -0.01248806 0.4640547   18.96370 1.169703
## L[2,2]  1.09318906 1.3674121   40.90909 1.071237
## L[2,3]  0.00000000 0.0000000 3000.00000      NaN
```

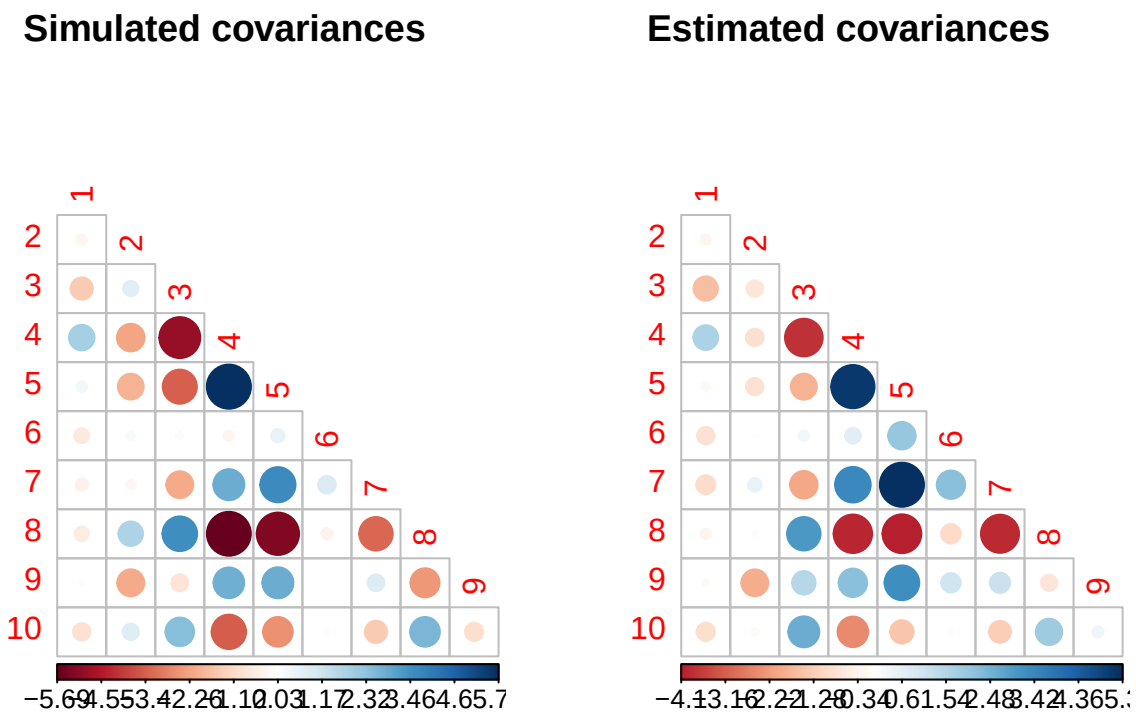
```
plot(mod_L[,1] ~ c(t(L)), cex = 2, pch = 16, col = "dark gray",
     xlab = "Simulated loadings", ylab = "Estimated loadings") # Plot the estimated loadings against th
fit_L <- lm(mod_L[,1] ~ c(t(L))) # Fit a linear model
abline(fit_L)
abline(0,1, lty = 2) # 1:1 line
```



And, more importantly : were we able to recover covariances among species?

```
## Extract the estimated covariance matrix (we will use the median of post warm-up samples)
cov_L <- summary(BFS, pars = "cov_L")$summary
mean_cov_L <- matrix(cov_L[,1], nrow = S, ncol = S)

## Compare the covariance plots
par(mfrow = c(1,2))
corrplot(tcrossprod(L), is.corr = F, diag = F, type = "lower", mar =c(1, 1, 1.1, 1) -0.2,
         title = "Simulated covariances" )
corrplot(mean_cov_L, is.corr = F, diag = F, type = "lower", mar =c(1, 1, 1.1, 1) -0.2,
         title = "Estimated covariances" )
```

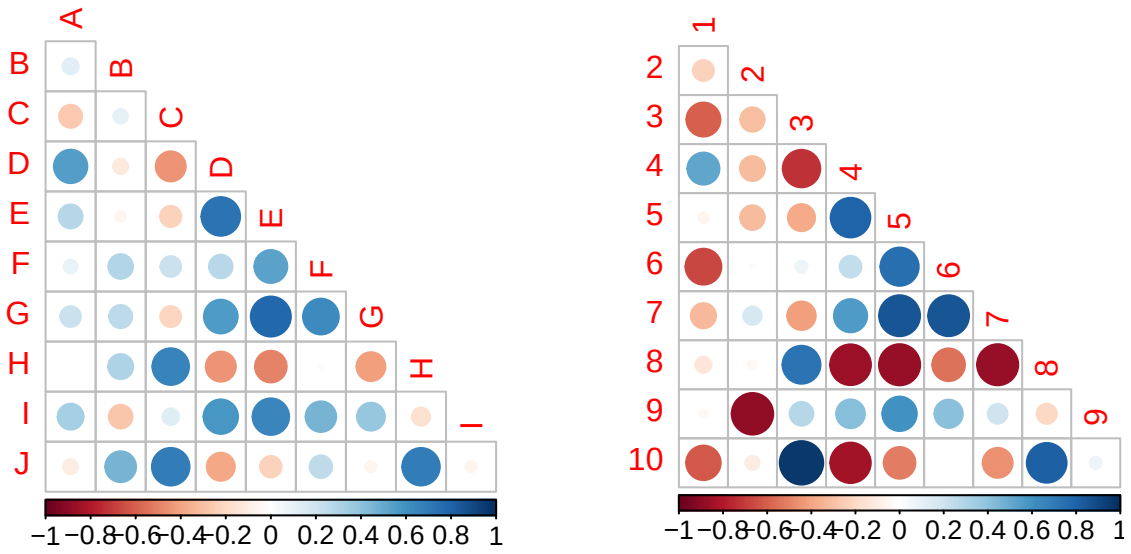



```
par(mfrow = c(1,1))
```

The model seems to be far more sensible to species correlations than the spearman rank based method are!

```
## Extract the estimated correlation matrix (we will use the median of post warm-up samples)
cor_L <- summary(BFS, pars = "cor_L")$summary
mean_cor_L <- matrix(cor_L[,1], nrow = S, ncol = S)

## Compare the correlation plots
par(mfrow = c(1,2))
corrplot(cor(Y, method = "spearman"), diag = F, type = "lower", mar =c(1, 1, 1.1, 1) -0.2,
         title = "Simulated correlations"  )
corrplot(mean_cor_L, diag = F, type = "lower", mar =c(1, 1, 1.1, 1) -0.2,
         title = "Estimated correlations"  )
```

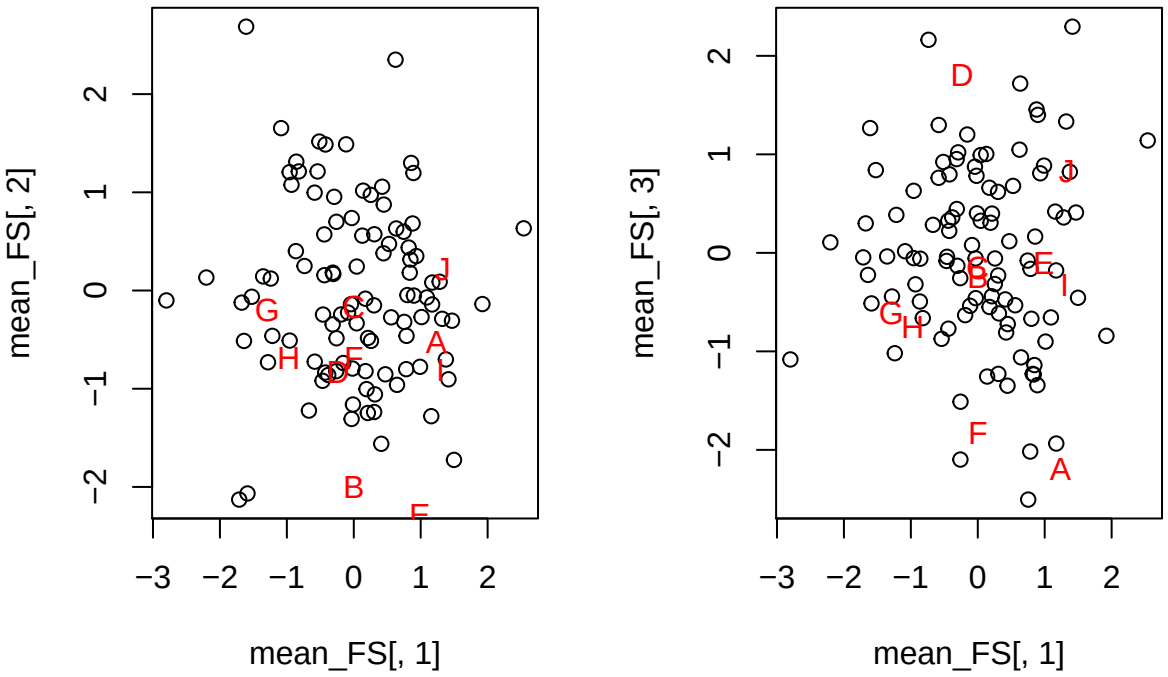



```
par(mfrow = c(1,1))
```

Finally, let's look at the estimated biplot!

```
## Extract the factor scores
mod_FS <- summary(BFS, pars = "FS")$summary
mean_FS <- matrix(mod_FS[,1],nrow = N, ncol = D)
mean_L <- matrix(mod_L[,1], nrow = S, ncol = D)

par(mfrow = c(1,2))
plot(mean_FS[,2] ~ mean_FS[,1]);text(mean_L[,2] ~ mean_L[,1],
                                     labels = LETTERS[1:S], col = "red")
plot(mean_FS[,3] ~ mean_FS[,1]);text(mean_L[,3] ~ mean_L[,1],
                                     labels = LETTERS[1:S], col = "red");par(mfrow = c(1,1))
```



Here is the code for the reparametrized version of the model, allowing really more efficient posterior sampling. Hierarchical and cauchy priors are not declared in the model part, but are computed in the transformed parameter block, by transforming heavy tailed distributions to easy-to-sample distributions.

```
data{
  int N; // sample size
  int S; // number of species
  int D; // number of factors
  int<lower=0> Y[N,S]; // data matrix of order [N,S]
}
transformed data{
  int<lower=1> M;
  vector[D] FS_mu; // factor means
  vector<lower=0>[D] FS_sd; // factor standard deviations
  M = D*(S-D)+ D*(D-1)/2;  // number of lower-triangular, non-zero loadings
  for (m in 1:D) {
    FS_mu[m] = 0; //Mean of factors = 0
    FS_sd[m] = 1; //Sd of factors = 1
  }
}
parameters{
  //Parameters
  real alpha; //Global intercept
  vector[N] d0_raw; //Uncentered row intercepts
  vector<lower=-pi()/2, upper=pi()/2>[M] L_lower_unif; //Uncentered lower diagonal loadings
  vector<lower=0, upper=pi()/2>[D] L_diag_unif; //Uncentered positive diagonal elements of loadings
  matrix[N,D] FS; //Factor scores, matrix of order [N,D]
  cholesky_factor_corr[D] Rho; //Correlation matrix between factors
  //Hyperparameters
  real<lower=0, upper=pi()/2> sigma_d_unif; //Uncentered sd of the row intercepts
  real<lower=-pi()/2, upper=pi()/2> mu_low_unif; //Uncentered mean of lower diag loadings
  real<lower=0, upper=pi()/2> tau_low_unif; //Uncentered scale of lower diag loadings
}
```

```
transformed parameters{
  // Final parameters
  vector[N] d0; //Final row intercepts
  vector[D] L_diag; //Final diagonal loadings
  vector[M] L_lower; // Final lower diagonal loadings
  cholesky_factor_cov[S,D] L; //Final matrix of laodings
  matrix[D,D] Ld; // cholesky decomposition of the covariance matrix between factors
  // Final hyperparameters
  real sigma_d; //Final sd of the row intercepts
  real mu_low; //Final mean of lower diag loadings
  real tau_low; //Final scale of lower diag loadings
  //Predictors
  matrix[N,S] Ups; //intermediate predictor
  matrix<lower=0>[N,S] Mu; //predictor

  // Compute the final hyperparameters
  sigma_d = 0 + 1 * tan(sigma_d_unif); //sigma_d ~ Halfcauchy(0,2.5)
  mu_low = 0 + 1 * tan(mu_low_unif); //mu_low ~ cauchy(0,2.5)
  tau_low = 0 + 1 * tan(tau_low_unif); //sigma_low ~ Halfcauchy(0,2.5)

  //Compute the final parameters
  d0 = 0 + sigma_d * d0_raw; //do ~ Normal(0, sigma_d)
  L_diag = 0 + 2.5 * tan(L_diag_unif); //L_diag ~ Halfcauchy(0, 2.5)
  L_lower = mu_low + tau_low * tan(L_lower_unif); //L_lower ~ cauchy(mu_low, tau_low)

  // Correlation matrix of factors
  Ld = diag_pre_multiply(FS_sd, Rho); //Fs_sd fixed to 1, Rho estimated

  {
    int idx2; //Index for the lower diagonal loadings
    idx2 = 0;

    // Constraints to allow identifiability of loadings
     for(i in 1:(D-1)) { for(j in (i+1):(D)){ L[i,j] = 0; } } //0 on upper diagonal
     for(i in 1:D) L[i,i] = L_diag[i]; //Positive values on diagonal
     for(j in 1:D) {
       for(i in (j+1):S) {
         idx2 = idx2+1;
         L[i,j] = L_lower[idx2]; //Insert lower diagonal values in loadings matrix
       }
     }
  }

  // Predictors
  Ups = FS * L';
  for(n in 1:N) Mu[n] = exp(alpha + d0[n] + Ups[n]);

}
model{
  // Uncentered hyperpriors : the sampling will be automaticaly done as if they were defined on an unif

  // Priors
  alpha ~ student_t(3,0,5); //Weakly informative prior for global intercept
  d0_raw ~ normal(0,1); //Uncentered regularizing prior for row deviations
```

```
  Rho ~ lkj_corr_cholesky(1); //Uninformative prior for Rho

  for(i in 1:N){
    Y[i,] ~ poisson(Mu[i,]);
    FS[i,] ~ multi_normal_cholesky(FS_mu, Ld);
  }
}
generated quantities{
  matrix[S,S] cov_L;
  matrix[S,S] cor_L;
  matrix[N,S] Y_pred;
  matrix[N,S] log_lik1;
  vector[N*S] log_lik;

  cov_L = multiply_lower_tri_self_transpose(L); //Create the covariance matrix

  // Compute the correlation matrix from de covariance matrix
  for(i in 1:S){
    for(j in 1:S){
      cor_L[i,j] = cov_L[i,j]/sqrt(cov_L[i,i]*cov_L[j,j]);
    }
  }

  //Compute the likelihood and predictions for each observation
  for(n in 1:N){
    for(s in 1:S){
      log_lik1[n,s] = poisson_lpmf(Y[n,s] | Mu[n,s]);
      Y_pred[n,s] = poisson_rng(Mu[n,s]);
    }
  }
  log_lik = to_vector(log_lik1); //Tranform in a vector usable by loo package
}
```


Acknowledgement

Many thanks to Vincent Maire, who always supports me in my fancies, Guillaume Blanchet for the time passed to explain the subtilities of his analysis to me, and Marco Rodriguez for the always enriching discussions!

References

Farouni, Rick. 2015. "Fitting a Bayesian Factor Analysis Model in Stan." http://rfarouni.github.io/2015-04-26-fa/.

forum, Stan. 2015. "Bayesian Factor Analysis." https://groups.google.com/forum/{\#}!msg/stan-users/wgGvHTwM2g0/JHeFianO4LQJ.

———. 2017. "Generalized linear latent variable model in Stan." http://discourse.mc-stan.org/t/generalized-linear-latent-variable-model-in-stan/2124.

———. 2018. "Help with factor analysis (latent variable model." http://discourse.mc-stan.org/t/help-with-factor-analysis-latent-variable-model/3923/2.

Hui, Francis K.C. 2016. "boral – Bayesian Ordination and Regression Analysis of Multivariate Abundance

Data in r." *Methods in Ecology and Evolution* 7 (6): 744–50. doi:10.1111/2041-210X.12514.

Tryfos, Peter. 1998. "Factor analysis." In *Methods for Business Analysis and Forecasting : Text and Cases*. Wiley.

Warton, David I., F. Guillaume Blanchet, Robert B. O'Hara, Otso Ovaskainen, Sara Taskinen, Steven C. Walker, and Francis K.C. Hui. 2015. "So Many Variables: Joint Modeling in Community Ecology." *Trends in Ecology and Evolution* 30 (12). Elsevier Ltd: 766–79. doi:10.1016/j.tree.2015.09.007.