**overview**

gjam models multivariate responses that can be combinations of discrete and continuous variables, where interpretation is needed on the observation scale. It was motivated by the challenges of modeling distribution and abundance of multiple species, so-called joint species distribution models (JSDMs). Because species and other attributes are often recorded on many different scales and with different levels of sampling effort, many analyses are limited to presence-absence, the lowest common denominator. Combining species and other attributes is challenging because some species groups are counted. Some may be continuous cover values or basal area. Some may be recorded in ordinal bins, such as ‘rare’, ‘moderate’, and ‘abundant’. Others may be presence-absence. Some are composition data, either fractional (continuous on (0, 1)) or counts (e.g., molecular and fossil pollen data). Attributes such as body condition, infection status, and herbivore damage are often included in field data. gjam accommodate multifarious observations.

To allow transparent interpretation gjam avoids non-linear link functions. This is a departure from generalized linear models (GLMs) and most hierarchical Bayes models.

The integration of discrete and continuous data on the observed scales makes use of *censoring*. Censoring extends a model for continuous variables across censored intervals. Continuous observations are uncensored. Censored observations are discrete and can depend on sample effort.

Censoring is used with the *effort* for an observation to combine continuous and discrete variables with appropriate weight. In count data, effort is determined by the size of the sample plot, search time, or both. It is comparable to the offset in GLMs. In count composition data (e.g., microbiome, fossil pollen), effort is the total count taken over all species. In gjam, discrete observations can be viewed as censored versions of an underlying continuous space.

gjam generates an object of class "gjam", allowing it to appropriate the summary and print functions in R. To avoid conflicts with other packages, gjam function names begin with "gjam". gjam uses the RcppArmadillo linear algebra library with the Rcpp interface library for R/C++.

**model summary**

The basic model is detailed in Clark et al. (2017) and summarized at the end of this vignette (see **reference notes**). An observation consists of two vectors (xi,yi)ni=1, where xi is a length-Q design vector (intercept and predictors) and yi is a length-S vector of response variables, each potentially measured in different ways. yi may include both continuous and discrete variables. There is a latent vector wi that represents response variables all on a continuous scale. The vector wi has the joint distribution wi∼MVN(μi,Σ), where μi is the length-S mean vector, and Σ, is an S×S covariance matrix.

As a data-generating mechanism the model can be thought of like this: There is a vector of continuous responses wi generated from mean vector μi and covariance Σ (Fig. 1a). A partition pis=(−∞,…,∞) segments the real line into intervals, some of which are censored and others not. Each interval is defined by two values, (pis,k,pis,k+1]. For a value of wis that falls within a censored interval k the observed yis is assigned to discrete interval zis=k. For a value of wis that falls in an uncensored interval yis is assigned wis.

Of course, data present us with the inverse problem: the observed yis are continuous or discrete, with known or unknown partition (pis,k,pis,k+1] (Fig. 1b). Depending on how the data are observed, we must impute the elements of n×S matrix W that lie within censored intervals. Unknown elements of P will also be imputed in order to estimate B and Σ.

A diagram of a function

AI-generated content may be incorrect.

**Censoring in gjam.** As a data-generating model (a), a realization wis that lies within a censored interval is translated by the partition pis to discrete yis. The distribution of data (bars at left) is induced by the latent scale and the partition. For inference (b), observed discrete yis takes values on the latent scale from a truncated distribution.

**data types**

The different types of data that can be included in the model are summarized in Table 1, assigned to the character variable typeNames that is included in the modelList passed to gjam:

**Table 1. Partition for each data type**

| **typeNames** | **Type** | **Obs values** | **Default partition** | **Comments** |
| --- | --- | --- | --- | --- |
| 'CON' | continuous, uncensored | (−∞,∞) | none | e.g., centered, standardized |
| 'CA' | continuous abundance | [0,∞) | (−∞,0,∞) | with zeros |
| 'DA' | discrete abundance | {0,1,2,…} | (−∞,12Ei,32Ei,…,maxs(yis)−1/2Ei,∞)1 | e.g., count data |
| 'PA' | presence- absence | {0,1} | (−∞,0,∞) | unit variance scale |
| 'OC' | ordinal counts | {0,1,2,…,K} | (−∞,0,estimates,∞) | unit variance scale, imputed partition |
| 'FC' | fractional composition | [0,1] | (−∞,0,1,∞) | relative abundance |
| 'CC' | count composition | {0,1,2,…} | (−∞,12Ei,32Ei,…,1−12Ei,∞)1 | relative abundance counts |
| 'CAT' | categorical | {0,1} | (−∞,maxk(0,wis,k),∞)2 | unit variance, multiple levels |

1 For 'DA' and 'CC' data the second element of the partition is not zero, but rather depends on effort. There is thus zero-inflation. The default partition for each data type can be changed with the function gjamCensorY (see **specifying censored intervals**).  
2 For 'CAT' data species s has Ks−1 non-reference categories. The category with the largest wis,k is the ‘1’, all others are zeros.

**effort and weight of discrete data**

The partition for a discrete interval k depends on effort for sample i

(pi,k,pi,k+1]=(k−1/2Ei,k+1/2Ei]

Effort affects the partition and, thus, the weight of each observation; wide intervals allow large variance, and vice versa. For **discrete abundance** ('DA') data on plots of a given area, large plots contribute more weight than small plots. Because plots have different areas one might choose to model wis on a ‘per-area’ scale (density) rather than a ‘per-plot’ scale. If so, plot area becomes the ‘effort’. Here is a table of variables for the case where counts represent the same density of trees per area, but have different effort due to different plot areas:

| **count**yis=zis | **plot area**Ei | **density**wis | **bin**k | **density**pik |
| --- | --- | --- | --- | --- |
| 10 | 0.1 ha | 100 ha−1 | 11 | (95, 105] |
| 100 | 1.0 ha | 100 ha−1 | 101 | (99.5, 100.5] |

The wide partition on the 0.1-ha plot admits large variance around the observation of 10 trees per 0.1 ha plot. Wide variance on an observation decreases its contribution to the fit. Conversely, the narrow partition on the 1.0-ha plot constrains density to a narrow interval around the observed value.

For **composition count** ('CC') data effort is represented by the total count. For 0<yis<Ei the variable 0<wis<1, i.e., the composition scale. Using the same partition as previously the table for two observations that represent the fraction 0.10 with different effort (e.g., total reads in PCR data) looks like this:

| **count**yis=zis | **total count**Ei | **fraction**wis | **bin**k | **fraction**pik |
| --- | --- | --- | --- | --- |
| 10 | 100 | 0.1 | 11 | (0.095, 0.105] |
| 10,000 | 100,000 | 0.1 | 10,001 | (0.099995, 0.100005] |

Again, on the composition scale [0,1], weight of the observation is determined by the partition width and, in turn, effort.

**using gjam**

It’s easiest to start with the examples from gjam help pages. This section provides additional examples and explanation.

**simulated data**

Simulated data are used to check that the algorithm can recover true parameter values and predict data, including underlying latent variables. To illustrate I simulate a sample of size n=500 for S=10 species and Q=4 predictors. To indicate that all species are *continuous abundance* data I specify typeNames as 'CA'. CA data are continuous above zero, with point mass at zero.

library(gjam)

f <- gjamSimData(n = 500, S = 10, Q = 4, typeNames = 'CA')

summary(f)

The object f includes elements needed to analyze the simulated data set. f$typeNames is a length-S character vector. The formula follows standard R syntax. It does not start with y ~, because gjam is multivariate. The multivariate response is supplied as a n×S matrix or data.frame ydata. Here is the formula for this example:

f$formula

The simulated parameter values are returned from gjamSimData in the list f$trueValues, shown in Table 2, with the corresponding names of estimates from function gjam, which I discuss below:

**Table 2. Variable names and dimensions in simulation and fitting**

| **model** | **$trueValues**1 | **$parameters**2 | **$chains**2 | **dimensions** |
| --- | --- | --- | --- | --- |
| B3Q×S | beta | betaMu | bgibbs | W |
| B3u,Q×S | - | betaMuUn | bgibbsUn | W/X |
| B~3Q1×S | - | betaStandXmu | bFacGibbs | dimensionless |
| ΣS×S | sigma | sigMu | sgibbs | WsWs′ |
| RS×S | corSpec | corMu |  | correlation |
| f4Q1 | - | fMu | fSensGibbs | dimensionless |
| F4Q1×Q1 | - | fmatrix |  | dimensionless |
| ES×S | - | ematrix | - | dimensionless |
| P5 | cuts | cutMu | cgibbs | correlation |
| K6 | - | - | kgibbs | dimensionless |
| σ2 6 | - | - | sigErrGibbs | W2 |
| α7Q×M | - | betaTraitMu | agibbs | W/X |
| Ω7M×M | - | sigmaTraitMu | mgibbs | WmUm′ 8 |

1 simulated object from gjamSimData.

2 fitted object from gjam.

3 coefficients are Bu: unstandardized; B: standardized for X; B~: standardized for X, correlation scale for W and factor levels relative to the mean for each factor (see section *factors in*X\*\*).

4 sensitivities based on unstandardized B and covariance scales Σ.

5 Only when ydata includes ordinal types "OC".

6 Only with dimension reduction, reductList is included in modelList.

7 Only for trait analysis, traitList is included in modelList (Trait vignette).

The dimension for response Y is W×E, where W is the latent variable scale, and E is sample effort. When effort E = 1, then Y and W have the same dimension.

The matrix F contains the covariance between predictors in X in terms of the responses they elicit from Y. The diagonal vector f=diag(F) is the sensitivity of the entire response matrix to each predictor in X.

The matrix E is the correlation among species in terms of their responses to X. Relationships to outputs are discussed in the **Reference notes**.

Simulated data are typical of real data in that there is a large fraction of zeros.

par(bty = 'n', mfrow = c(1,2), family='')

h <- hist(c(-1,f$y), nclass = 50, plot = F)

plot( h$counts, h$mids,type = 's' )

plot( f$w,f$y,cex = .2 )

Here is a short Gibbs sampler to estimate parameters and fit the data. The function gjam needs the formula for the model, the data.frame xdata, which includes the predictors, the response matrix or data.frame ydata, and a modelList specifying number of Gibbs steps ng, the burnin, and typeNames.

ml <- list( ng = 1000, burnin = 100, typeNames = f$typeNames )

out <- gjam( f$formula, f$xdata, f$ydata, modelList = ml )

summary(out)

The print function acts like summary

print(out)

Among the objects to consider initially are the design matrix out$inputs$xUnstand, response matrix out$inputs$y, and the MCMC out$chains with these names and sizes:

summary(out$chains)

$chains is a list of matrices, each with ng rows and as many columns as needed to hold parameter estimates. For example, each row of $chains$bgibbs is a length-QS vector of values for the Q×S matrix B, standardized for X. In other words, a coefficient Bqs has the units of ws. $chains$sgibbs holds either the S(S+1)/2 unique values of Σ or the N×r unique values of the dimension reduced covariance matrix. A summary of the chains is given in Table 2.

Additional summaries are available in the list inputs:

summary(out$inputs)

The matrix classBySpec shows the number of observations in each interval. For this example of continuous data censored at zero, the two labels are k∈{0,1} corresponding to the intervals (ps,0,ps,1]=(−∞,0] and (ps,1,ps,2)=(0,∞). The length-(K+1) partition vector is the same for all species, p=(−∞,0,∞). Here is classBySpec for this example:

out$inputs$classBySpec

The first interval is censored (all values of yis = 0). The second interval is not censored (yis=wis).

The fitted coefficients in $parameters, as summarized in Table 2. For example, here is posterior mean estimate of unstandardized coefficients Bu,

out$parameters$betaMu

Here are posterior summaries,

out$parameters$betaMu *# S by M coefficient matrix unstandardized*

out$parameters$betaSe *# S by M coefficient SE*

out$parameters$betaStandXmu *# S by M standardized for X*

out$parameters$betaStandXWmu *# (S-F) by M standardized for W/X, centered factors*

out$parameters$betaTable *# SM by stats posterior summary*

out$parameters$betaStandXTable *# SM by stats posterior summary*

out$parameters$betaStandXWTable *# (S-F)M by stats posterior summary*

out$parameters$sensBeta *# sensitivity to response variables*

out$parameters$sensTable *# sensitivity to predictor variables*

out$parameters$sigMu *# S by S covariance matrix omega*

out$parameters$sigSe *# S by S covariance std errors*

Again, check Table 2 for names of all fitted coefficients.

The data are also predicted in gjam, summarized by predictive means and standard errors. These are contained in n×Q matrices $prediction$xpredMu and $prediction$xpredSd and n×S matrices $prediction$ypredMu and $prediction$ypredSd. These are in-sample predictions.

The output can be viewed with the function gjamPlot:

f <- gjamSimData( n = 500, S = 10, typeNames = 'CA' )

ml <- list( ng = 1000, burnin = 200, typeNames = f$typeNames )

out <- gjam( f$formula, f$xdata, f$ydata, modelList = ml )

pl <- list( trueValues = f$trueValues, GRIDPLOTS = T )

gjamPlot( output = out, plotPars = pl )

gjamPlot creates a number of plots comparing true and estimated parameters (for simulated data).

par( bty = 'n', mfrow = c(1,3) )

plot( f$trueValues$beta, out$parameters$betaMu, cex = .2 )

plot( f$trueValues$corSpec, out$parameters$corMu, cex = .2 )

plot( f$y,out$prediction$ypredMu, cex = .2 )

To process the output beyond what is provided in gjamPlot I can work directly with the chains.

**my data**

gjam uses the standard R syntax in the formula that I would use with functions like lm and glm. Because gjam uses inverse prediction to summarize large multivariate output, it is important to abide by this syntax. For example, to analyze a model with quadratic and interaction terms, I might simply construct my own design matrix with these columns included, i.e., side-stepping the standard syntax for these effects that can be specified in formula. This would be fine for model fitting. However, without specifying this in the formula there is no way for gjam to know that these columns are in fact non-linear transformations of other columns. Without this knowledge there is no way to properly predict them. The prediction that gjam would return would include silly variable combinations.

To illustrate proper model specification I use a few lines from the data.frame of predictors in the forestTraits data set:

library(repmis)

d <- "https://github.com/jimclarkatduke/gjam/blob/master/forestTraits.RData?raw=True"

source\_data(d)

xdata <- forestTraits$xdata[,c(1,2,7,8)]

Here are a few rows:

xdata[1:5,]

Here is a simple model specification with as.formula() that includes only main effects:

formula <- as.formula( ~ temp + deficit + soil )

The design matrix x that is generated in gjam has an intercept, two covariates, and four columns for the multilevel factor soil:

## (Intercept) temp deficit soilreference

## 1 1 1.22 0.04 1

## 2 1 0.18 0.21 1

## 3 1 -0.94 0.20 0

## 4 1 0.64 0.82 1

## 5 1 0.82 -0.18 1

To include interactions between temp and soil I use the symbol ‘\*’:

formula <- as.formula( ~ temp\*soil )

Here is the design matrix that results from this formula with interaction terms indicated by the symbol ':'

## (Intercept) temp soilreference temp:soilreference

## 1 1 1.22 1 1.22

## 2 1 0.18 1 0.18

## 3 1 -0.94 0 0.00

## 4 1 0.64 1 0.64

## 5 1 0.82 1 0.82

For a quadratic term I use the R function I():

formula <- as.formula( ~ temp + I(temp^2) + deficit )

Here is the design matrix with linear and quadratic terms:

## (Intercept) temp I(temp^2) deficit

## 1 1 1.22 1.4884 0.04

## 2 1 0.18 0.0324 0.21

## 3 1 -0.94 0.8836 0.20

## 4 1 0.64 0.4096 0.82

## 5 1 0.82 0.6724 -0.18

Here is a quadratic response surface for temp and deficit:

formula <- as.formula( ~ temp\*deficit + I(temp^2) + I(deficit^2) )

Here is the design matrix with all combinations:

## (Intercept) temp deficit I(temp^2) I(deficit^2) temp:deficit

## 1 1 1.22 0.04 1.4884 0.0016 0.0488

## 2 1 0.18 0.21 0.0324 0.0441 0.0378

## 3 1 -0.94 0.20 0.8836 0.0400 -0.1880

## 4 1 0.64 0.82 0.4096 0.6724 0.5248

## 5 1 0.82 -0.18 0.6724 0.0324 -0.1476

These are examples of the formula options available in gjam. Using them will allow for proper inverse prediction of x. To optimize MCMC, gjam does not predict x for higher order polynomials–they are rarely used, being both hard to interpret and a cause of unstable predictions. To accelerate MCMC I can set PREDICTX = F in the modelList.

I can use this model to analyze a tree data set. For my data set I use the tree data contained in forestTraits. It is stored in de-zeroed (sparse matrix) format, so I extract it with the function gjamReZero. Here are dimensions and the upper left corner of the response matrix Y,

y <- gjamReZero(forestTraits$treesDeZero) *# extract y*

treeYdata <- gjamTrimY(y,10)$y *# at least 10 plots*

dim(treeYdata)

treeYdata[1:5,1:6]

In code that follows I treat responses as discrete counts, typeNames = 'DA'. Because of the large number of columns (98) I speed things up by calling for dimension reduction, passed as N×r=20×8:

ml <- list( ng = 2500, burnin = 500, typeNames = 'DA',

reductList = list(r = 8, N = 20) )

form <- as.formula( ~ temp\*deficit + I(temp^2) + I(deficit^2) )

out <- gjam(form, xdata = xdata, ydata = treeYdata, modelList = ml)

specNames <- colnames(treeYdata)

specColor <- rep('black',ncol(treeYdata))

specColor[ c(grep('quer',specNames),grep('cary',specNames)) ] <- '#d95f02'

specColor[ c(grep('acer',specNames),grep('frax',specNames)) ] <- '#1b9e77'

specColor[ c(grep('abie',specNames),grep('pice',specNames)) ] <- '#377eb8'

gjamPlot( output = out, plotPars = list(GRIDPLOTS=T, specColor = specColor) )

Additional information on variable types and their treatment in gjam is included later in this document and in the other gjam vignettes.

**sensitivity to predictors**

The sensitivity of an individual response variable s to an individual predictor q is given by the coefficient βqs. This granularity is useful, but it is often desirable to have a sensitivity that applies to the full response matrix. That sensitivity is given by

f=diag(BΣ−1B′)

(Brynjarsdottir and Gelfand. 2014, Clark et al. 2017). These coefficients are evaluated on the scale that is standardized scale for X and correlation scale Y–they are dimensionless. In the notation of the Appendix this is Br. A plot of these values is displayed when I call gjamPlot.

I can also evaluate sensitivity for a species group g as

fg=diag(BgΣ−1gB′g)

where Bg includes columns for the species in group g, and Σg is the covariance matrix for group g conditional on remaining species in the model.

In the help page for the function gjamSensitivity is this example comparing sensitivity for a simulated data set with multiple data types against a group that includes only the composition count ('CC') data. Note that the latter is supplied identified by group, which is a character string of column names in ydata:

cols <- c( '#1f78b4', '#33a02c' )

types <- c('DA','DA','OC','OC','OC','OC','CC','CC','CC','CC','CC','CA','CA','PA','PA')

f <- gjamSimData(S = length(types), typeNames = types)

ml <- list(ng = 500, burnin = 50, typeNames = f$typeNames)

out <- gjam(f$formula, f$xdata, f$ydata, modelList = ml)

ynames <- colnames(f$y)

group <- ynames[types == 'CC']

full <- gjamSensitivity(out)

cc <- gjamSensitivity(out, group)

ylim <- range( c(full, cc) )

nt <- ncol(full)

boxplot( full, boxwex = 0.2, at = 1:nt - .15, col = cols[1], log='y',

ylim = ylim, xaxt = 'n', xlab = 'Predictors', ylab='Sensitivity')

boxplot( cc, boxwex = 0.2, at = 1:nt + .15, col = cols[2], add=T,

xaxt = 'n')

axis( 1, at=1:nt,labels=colnames(full) )

legend( 'bottomright',c('full response','CC data'),

text.col = cols, bty='n' )

Again, the scale is dimensionless. Here is a comparison between two groups:

group <- ynames[types == 'CA']

ca <- gjamSensitivity(out, group)

ylim <- range( rbind(cc,ca) )

nt <- ncol(full)

boxplot( ca, boxwex = 0.2, at = 1:nt - .15, col = cols[1], log='y',

xaxt = 'n', ylim = ylim, xlab = 'Predictors', ylab='Sensitivity')

boxplot( cc, boxwex = 0.2, at = 1:nt + .15, col = cols[2], add=T,

xaxt = 'n')

axis(1,at=1:nt,labels=colnames(full))

legend('bottomright',c('CA data','CC data'),

text.col = cols, bty = 'n' )

**plotting output**

In the foregoing example arguments passed to gjamPlot in the list plotPars included SMALLPLOTS = F (do not compress margins and axes), GRIDPLOTS = T (draw grid diagrams as heat maps for parameter values and predictions). In this section I summarize plots generated by gjamPlot.

By default, plots are rendered to the screen. I enter ‘return’ to render the next plot. Faster execution obtains if I write plots directly to pdf files, with SAVEPLOTS = T. I can specify a folder this way:

plotPars <- list( GRIDPLOTS=T, SAVEPLOTS = T, outfolder = 'plots' )

In all plots, posterior distributions and predictions are shown as 68% (boxes) and 95% (whiskers) intervals, respectively. Here are the plots in alphabetical order by file name:

| **Name** | **Comments** |
| --- | --- |
| betaAll | Posterior B |
| beta\_(variable) | Posterior distributions, one file per variable |
| betaChains | Example MCMC chains for B (has it converged?) |
| clusterDataE | Cluster analysis of raw data and E matrix |
| clusterGridB | Cluster and grid plot of E and B |
| clusterGridE | Cluster and grid plot of E |
| clusterGridR | Cluster and grid plot of R |
| corChains | Example MCMC chains for R |
| dimRed | Dimension reduction (see vignette) for Σ matrix |
| gridF\_B | Grid plot of sensitivity F and B, ordered by clustering F |
| gridR\_E | Grid plot of R and E ordered by clustering R |
| gridR | Grid plot of R, ordered by cluster analysis. |
| gridY\_E | Grid plot of correlation for data Y and E, ordered by clustering cor(Y) |
| gridTraitB | If traits are predicted, see gjam vignette on traits. |
| ordination | PCA of E matrix, including eigenvalues (cumulative) |
| partition | If ordinal responses, posterior distribution of P |
| richness | Predictive distribution with distribution of data (histogram) |
| sensitivity | Overall sensitivity f by predictor |
| traits | If traits are predicted, see gjam vignette on traits. |
| traitPred | If traits are predicted, see gjam vignette on traits. |
| trueVsPars | If simulated data and trueValues included in plotPars |
| xPred | Inverse predictive distribution of of X |
| xPredFactors | Inverse predictive distribution of factor levels |
| yPred | Predicted Y, in-sample (blue bars), out-of-sample (dots), and distribution of data (histogram) |
| yPredAll | If PLOTALLY all response predictions shown |

If the plotPars list passed to gjamPlot specifies GRIDPLOTS = T, then grid and cluster plots are generated as gridded values for B, Σ and R. Gridplots of matrix R show conditional and marginal dependence in white and grey. In plots of E marginal independence is shown in grey, but conditional independence is not shown, as the matrix does not have an inverse (Clark et al. 2017).

The sensitivity matrix F is shown together in a plot with individual species responses B.

The plot in which the model residual correlation R and the response correlation B are compared are ordered by their similiarity in the R. If the two contain similar structure, then it will be evident in this comparison. There is no reason to expect them to be similar.

For large S the labels are not shown on the graphs, they would be too small. The order of species and the cluster groups to which they belong are returned in fit$clusterOrder and fit$clusterIndex.

**flexibility in gjam**

**heterogeneous effort**

Here is an example with discrete abundance data, now with heterogeneous sample effort. Heterogeneous effort applies where vegetation plots have different areas, animal survey data have variable search times, or catch returns from fishing vessels have different gear and/or trawling times. Here I simulate a list containing the columns and the effort that applies to those columns, shown for 50 observations:

S <- 5

n <- 50

ef <- list( columns = 1:S, values = round(runif(n,.5,5),1) )

f <- gjamSimData(n, S, typeNames = 'DA', effort = ef)

ef

If ef$values consists of a length-n vector, then gjam assumes each value applies to all columns in the corresponding row specified in the vector ef$columns. This is the case shown above and would apply when effort is plot area, search time, sample volume, and so forth. Alternatively, values can be supplied as a matrix, having the same dimensions as ydata. As a matrix, ef$values can have elements that differ by observation and species. For example, camera trap data detect large animals at greater distances than small animals (column differences), and each camera can be deployed for different lengths of time (row differences). For simulation purposes gjamSimData only accepts a vector. However, for fitting with gjam effort$values can be supplied as a matrix with as many columns as are listed in effort$columns.

Because observations are discrete the continuous latent variables wis are censored. Unlike the previous continuous example, observations yis now assume only discrete values:

plot( f$w, f$y, cex = .2, xlab = 'Latent w scale', ylab = 'Observed y' )

The large scatter reflects the variable effort represented by each observation. Incorporating the effort scale gives this plot:

plot(f$w\*ef$values, f$y, cex = .2, xlab = 'Latent w time effort', ylab = 'Observed y')

The heterogeneous effort affects the weight of each observation in model fitting. The effort is entered in modelList. Increase the number of iterations and look at plots:

S <- 10

n <- 1500

ef <- list( columns = 1:S, values = round(runif(n,.5,5),1) )

f <- gjamSimData(n, S, typeNames = 'DA', effort = ef)

ml <- list(ng = 1000, burnin = 250, typeNames = f$typeNames, effort = ef)

out <- gjam(f$formula, f$xdata, f$ydata, modelList = ml)

pl <- list( trueValues = f$trueValues )

gjamPlot(output = out, plotPars = pl)

Use summary(out) to see a summary of effort.

**specifying censored intervals**

To analyze data that are censored at specific intervals, I specify the censored values and the intervals to which those values apply. In the help page for gjamCensorY I discuss a vector of values and a corresponding matrix of upper and lower bounds in two rows and one column for each element in values. For example, if an observer stops counting wildebeest or sea lions at, say, 100 animals, I can set this as a censored interval:

*# assumes there is a data matrix ydata*

upper <- 100

intv <- matrix(c(upper,Inf),2)

rownames(intv) <- c('lower','upper')

tmp <- gjamCensorY(values = upper, intervals = intv, y = f$ydata, type='DA')

There are two objects returned by gjamCensor. The list $censor holds two lists, the $columns, indicating which columns in $y are censored, and the $partition, giving the values and bounds for the partition. Because I did not specify whichcol in my call to gjamCensorY, censoring defaults to all columns in ydata. The object $y matches the mode and dimensions of the input y (a matrix or data.frame). This new version of the response matrix replaces any values in ydata that fall within a censored interval with the censored value. This feature is useful when observers are inconsistent on the assignment of intervals or where an analyst distrusts the precision reported in data. For example, if counts range to thousands, but it is known that counts beyond 100 are inaccurate, then all counts above 100 are censored and, thus, uncertain.

Censoring can be applied differently to each response variable. For example, chemical constitents reported as concentrations in a sample, sometimes reach non-zero minimum values, taken as detection limits for that instrument and that constituent (detection limits can differ for each constituent). In this case, there is a list for each column in ydata. I can generate this list with a loop, where the censored interval for each column j spans from -Inf to min(ydata[,j]),

miny <- apply(f$ydata, 2, min, na.rm=T) *#minimum value by column*

censor <- numeric(0)

p <- matrix(0, 3, dimnames=list(c("values","lower","upper"), NULL))

**for**(j **in** 1:ncol(f$ydata)){

p[1:3] <- c(miny[j], -Inf, miny[j])

jlist <- list("columns" = j, "partition" = p)

censor <- c(censor, list(jlist))

names(censor)[j] <- 'CA'

}

This list censor can be passed directly to gjam in the list modelList.

**prior distribution on coefficients**

Informative prior distributions in regression models are rare, perhaps partly because it is hard to assign both magnitude (e.g., large or small prior mean value) and weight (large or small prior variance) without obscuring the relative contributions of prior and data. Prior distributions for regression coefficients are typically Gaussian, having support on (−∞,∞). In many cases, the sign of the effect is known, but the magnitude is not. Ad hoc experimentation with prior mean and variance can, at best, only insure that ‘most’ of the posterior distribution is positive or negative. Yet, it can be important to use prior information, especially in the multivariate setting, where covariances between responses can result in estimates where the sign of a coefficient effect makes no sense.

The knowledge of the ‘direction’ of the effect can be readily implmented with uniform priors truncated at zero, having the advantage that the posterior distribution that preserves the shape of the likelihood, but is restricted to positive or negative values (Clark et al. 2013).

The prior distribution for B is either non-informative (if unspecified) or truncated by limits provided in the list betaPrior. The betaPrior list contains the two matrices lo and hi. The rows of these matrices have rownames that match explanatory variables in the formula and colnames in xdata. In the example that follows I fit a model for FIA data to winter temperature temp, climatic deficit, and their interaction. For this example, I use a prior distribution having positive effects of warm winters and negative effects of climate deficit. This prior is set up with the function gjamPriorTemplate.

The prior distribution is also the place to exclude specific predictor/response combinations, by setting them equal to NA in lo or hi. Here is an example of informative priors on some coefficients:

xdata <- forestTraits$xdata

formula <- as.formula(~ temp\*deficit)

snames <- colnames(treeYdata)

*# warm winter*

hot <- c("liquStyr","liriTuli","pinuEchi","pinuElli","pinuPalu","pinuTaed",

"querImbr","querLaur","querLyra","querMich","querMueh","querNigr",

"querPhel","querVirg") *# arbitrary spp, positive winter temp*

nh <- length(hot)

lo <- vector('list', nh)

names(lo) <- paste('temp',hot,sep='\_')

**for**(j **in** 1:nh)lo[[j]] <- 0

*# humid climate (negative deficit)*

humid <- c("abieBals", "betuAlle", "querNigr", "querPhel") *#again, arbitrary*

nh <- length(humid)

hi <- vector('list', nh)

names(hi) <- paste('deficit',humid,sep='\_')

**for**(j **in** 1:nh)hi[[j]] <- 0

bp <- gjamPriorTemplate(formula, xdata, ydata = treeYdata, lo = lo, hi=hi)

rl <- list(N = 10, r = 5)

ml <- list(ng=1000, burnin=200, typeNames = 'CA', betaPrior = bp, reductList=rl)

out <- gjam(formula, xdata, ydata = treeYdata, modelList = ml)

sc <- rep('grey',ncol(treeYdata))

sc[snames %in% hot] <- '#ff7f00' *# highlight informative priors*

sc[snames %in% humid] <- '#1f78b4'

pl <- list( GRIDPLOTS = T, specColor = sc)

gjamPlot(output = out, plotPars = pl)

The combination of lo and hi set the limits for posterior draws from the truncated multivariate normal distribution. The help page for gjamPriorTemplate provides an example with informative priors specified for individual predictor-response combinations.

**species-factor levels that do not occur**

There can be times when specific species-predictor combinations could be omitted. For example, some species may never occur on some soil types. If I want to estimate β conditioned on some elements being zero, I could do the following:

formula <- as.formula(~ temp + soil)

*# find species-soil type combinations that occur less than 5 times*

y0 <- treeYdata

y0[ y0 > 0 ] <- 1

soil <- rep( as.character(xdata$soil), ncol(y0) )

soil <- paste('soil', soil, sep='') *# soil is a factor, so full name begins with variable name*

spec <- rep( colnames(y0), each = nrow(y0) )

specBySoil <- tapply( as.vector(y0), list(spec, soil), sum )

wna <- which( specBySoil < 5, arr.ind = T ) *# only if at least 5 observations*

*# assign NA to excluded coefficients using species-variable names*

nh <- nrow(wna)

lo <- vector('list', nh)

names(lo) <- paste( rownames(specBySoil)[wna[,1]], colnames(specBySoil)[wna[,2]], sep='\_' )

hi <- lo

**for**(j **in** 1:nh)lo[[j]] <- NA

hi <- lo

*# setup prior and fit model*

bp <- gjamPriorTemplate(formula, xdata, ydata = treeYdata, lo = lo, hi=hi)

rl <- list(N = 10, r = 5)

ml <- list(ng=1000, burnin=200, typeNames = 'CA', betaPrior = bp, reductList=rl)

out <- gjam(formula, xdata, ydata = treeYdata, modelList = ml)

The values in out$parameters$betaMu will show these omitted values as NA.

**sample effort in count data**

Discrete count ('DA') data have effort associated with search time, plot area, and so forth. When the effort values are measured in units that result in large differences between data types sampled with different efforts, model fitting deteriorates. The modeling scale is W=Y/E, where Y are counts, and E is effort. Units that make E large, can make W vanishingly small. When counts per unit effort span different orders of magnitude for different data types, consider changing the units on effort to bring them more in alignment with one another. Model predictions for ground beetles (pitfall traps) and small mammals (live traps) in our NEON analysis improved by shifting from trap-days to trap-months.

**sample effort in composition data**

Composition count ('CC') data have heterogenous effort due to different numbers of counts for each sample. For example, in microbiome data, the number of reads per sample can range from 102 to 106. Typically, the number of reads does not depend on total abundance. It is generally agreed that only relative differences are important. gjam knows that the effort in CC data is the total count for the sample, so effort does not need to be specified. Here is an example with simulated data:

f <- gjamSimData( S = 8, typeNames = 'CC' )

ml <- list( ng = 2000, burnin = 500, typeNames = f$typeNames )

out <- gjam( f$formula, f$xdata, f$ydata, modelList = ml )

gjamPlot( output = out, plotPars = list( trueValues = f$trueValues) )

For comparison, here is an example with fractional composition, where there is no effort:

f <- gjamSimData( S = 10, typeNames = 'FC' )

ml <- list( ng = 2000, burnin = 500, typeNames = f$typeNames )

out <- gjam( f$formula, f$xdata, f$ydata, modelList = ml )

gjamPlot( output = out, plotPars = list( trueValues = f$trueValues ) )

**the partition in ordinal data**

Ordinal count ('OC') data are collected where abundance must be evaluated rapidly or precise measurements are difficult. Because there is no absolute scale the partition must be inferred. Here is an example with 10 species:

f <- gjamSimData( typeNames = 'OC' )

ml <- list( ng = 2000, burnin = 500, typeNames = f$typeNames )

out <- gjam( f$formula, f$xdata, f$ydata, modelList = ml )

A simple plot of the posterior mean values of cutMu shows the estimates with true values from simulation:

par( mfrow=c(1,1), bty = 'n' )

keep <- strsplit(colnames(out$parameters$cutMu),'C-') *#only saved columns*

keep <- matrix(as.numeric(unlist(keep)), ncol = 2, byrow = T)[,2]

plot( f$trueValues$cuts[,keep], out$parameters$cutMu, xlab = 'True partition',

ylab = 'Estimated partition')

Here are plots:

pl <- list(trueValues = f$trueValues)

gjamPlot(output = out, plotPars = pl)

**categorical data**

Categorical data have levels within groups. The levels are unordered. The columns in ydata that hold categorical responses are declared as typeNames = "CAT". In observation vector yi there are elements for one less than the number of factor levels. Suppose that observations are obtained on attributes of individual plants, each plant being an observation. The group leaf type might have four levels broadleaf decidious bd, needleleaf decidious nd, broadleaf evergreen be, and needleaf evergreen ne. A second group xylem anatomy might have three levels diffuse porous dp, ring porous rp, and tracheid tr. In both cases I assign the last class to be a reference class, other. Ten rows of the response matrix data might look like this:

## leaf xylem

## 1 bd dp

## 2 be dp

## 3 bd rp

## 4 nd other

## 5 be rp

## 6 bd rp

## 7 other rp

This data.frame ydata becomes this response matrix y:

## leaf\_bd leaf\_nd leaf\_be leaf\_other xylem\_dp xylem\_rp xylem\_other

## [1,] 1 0 0 0 1 0 0

## [2,] 0 0 1 0 1 0 0

## [3,] 1 0 0 0 0 1 0

## [4,] 0 1 0 0 0 0 1

## [5,] 0 0 1 0 0 1 0

## [6,] 1 0 0 0 0 1 0

## [7,] 0 0 0 1 0 1 0

gjam expands the two groups into four and three columns in y, respectively. As for composition data there is one redundant column for each group. Here is an example with simulated data, having two categorical groups:

types <- c( 'CAT', 'CAT' )

f <- gjamSimData( n = 3000, S = length(types), typeNames = types )

ml <- list( ng = 1500, burnin = 500, typeNames = f$typeNames, PREDICTX = F )

out <- gjam( f$formula, xdata = f$xdata, ydata = f$ydata, modelList = ml )

gjamPlot( out, plotPars = list(trueValues = f$trueValues, PLOTALLY = T) )

**combinations of data types**

One of the advantages of gjam is that it combines data of many types. Here is an example showing joint analysis of 12 species represented by five data types, specified by column:

types <- c('OC','OC','OC','OC','CC','CC','CC','CC','CC','CA','CA','PA','PA' )

f <- gjamSimData( S = length(types), Q = 5, typeNames = types )

ml <- list( ng = 2000, burnin = 500, typeNames = f$typeNames )

out <- gjam( f$formula, f$xdata, f$ydata, modelList = ml )

tmp <- data.frame( f$typeNames, out$inputs$classBySpec[,1:10] )

print( tmp )

I have displayed the first 10 columns of classBySpec from the inputs of out, with their typeNames. The ordinal count ('OC') data occupy lower intervals. The width of each interval in OC data depends on the estimate of the partition in cutMu.

The composition count ('CC') data occupy a broader range of intervals. Because CC data are only relative, there is information on only S−1 species. One species is selected as other. The other class can be a collection of rare species (Clark et al. 2017).

Both continuous abundance ('CA') and presence-absence ('PA') data have two intervals. For CA data only the first interval is censored, the zeros (see above). For PA data both interval are censored; it is a multivariate probit.

Here are some plots for analysis of this model:

gjamPlot( output = out, plotPars = list(trueValues = f$trueValues, GRIDPLOTS = T) )

**random effects**

In addition to random effects on species used in dimension reduction, gjam allows for random groups. Examples could be observer effects for bird point counts or plot effects, where there are multiple observations per plot. Just like fixed effects, random effects should have replication. In other words, if I want to estimate an observer effect, I should have multiple observations for each observer. For plot effects, I want replication within plots.

To specify random groups, I provide the name of the column in xdata that holds the group indicator. This can be entered as an integer, a factor level, or a character variable.

modelList$random <- 'columnNameInXdata'

This column will be the basis for the random groups. Any rare groups (less than a few observations) will be aggregated into a single rareGroups category, again, to insure replication.

In the gjam output, here are objects related to random effects in output$parameters:

| **object** | **dimension** | **description** |
| --- | --- | --- |
| randByGroupMu | S by G | random groups, mean |
| randByGroupSe | S by G | SE |
| groupIndex | n | group index for observations |

**missing data, out-of-sample prediction**

gjam identifies missing values in xdata and ydata and models them as part of the posterior distribution. These are located at $missing$xmiss and $missing$ymiss. The estimates for missing X are $missing$xmissMu and $missing$xmissSe. The estimates for missing Y are $missing$ymissMu and $missing$ymissSe.

To simulate missing data use nmiss to indicate the number of missing values. The actual value will be less than nmiss:

f <- gjamSimData(typeNames = 'OC', nmiss = 20)

which(is.na(f$xdata), arr.ind = T)

Note that missing values are assumed to occur in random rows and columns, but not in column one, which is the intercept and known. No further action is needed for model fitting, as gjam knows to treat these as missing data.

Out-of-sample prediction of Y is not part of the posterior distribution. For model fitting, holdouts can be specified randomly in modelList with holdoutN (the number of plots to be held out at random) or with holdoutIndex (observation numbers, i.e., row numbers in x and y). The latter can be useful when a comparison of predictions is desired for different models using the same plots as holdouts.

When observations are held out, gjam provides out-of-sample prediction for both x[holdoutIndex,] and y[holdoutIndex,]. The holdouts are not included in the fitting of B,Σ, or P. For prediction of y[holdoutIndex,], the values of x[holdoutIndex,] are known, and sampling for w[holdoutIndex,] is done with multivariate normal distribution, without censoring. This is done because the censoring depends on y[holdoutIndex,], which taken to be unknown. This sample of w[holdoutIndex,] becomes a prediction for y[holdoutIndex,] using the partition (Figure 1a).

For inverse prediction of x[holdoutIndex,] the values of y[holdoutIndex,] are known. This represents the situation where a sample of the community is available, and the investigator would like to predict the environment of origin.

Here is an example with simulated data having both missing values and holdout observations:

f <- gjamSimData( typeNames = 'CA', nmiss = 20 )

ml <- list( ng = 2000, burnin = 500, typeNames = f$typeNames, holdoutN = 50 )

out <- gjam( f$formula, f$xdata, f$ydata, modelList = ml )

par( mfrow=c(1,3), bty = 'n' )

xMu <- out$prediction$xpredMu *# inverse prediction of x*

xSd <- out$prediction$xpredSd

yMu <- out$prediction$ypredMu *# predicted y*

hold <- out$modelList$holdoutIndex *# holdout observations (rows)*

plot(out$inputs$xUnstand[hold,-1],xMu[hold,-1], cex=.2, xlab='True', ylab='Predictive mean')

title('holdouts in x')

abline( 0, 1, lty=2 )

plot(out$inputs$y[hold,], yMu[hold,], cex=.2, xlab='True', ylab='')

title('holdouts in y')

abline( 0, 1, lty=2 )

xmiss <- out$missing$xmiss *# locations of missing x*

xmissMu <- out$missing$xmissMu

xmissSe <- out$missing$xmissSe

xmean <- apply(f$xdata,2,mean,na.rm=T)[xmiss[,2]] *# column means for missing values*

plot(xmean, xmissMu, xlab='Variable mean', ylab='Missing estimate') *#posterior estimates*

segments(xmean, xmissMu - 1.96\*xmissSe, xmean, xmissMu + 1.96\*xmissSe) *#approx 95% CI*

title('missing x')

Note that there are no ‘true’ values in the simulation for missing x–the last graph at right just shows estimates relative to mean values for respective variables.

**prediction with heterogenous effort**

Out-of-sample prediction can not only be done by holding out samples in gjam. It can also be done post-fitting, with the function gjamPredict. In this second case, a prediction grid is passed together with the fitted object generated by gjam. Here is an example with counts (DA) and continuous data (CA). I simulate, fit, and predict these data with heterogeneous sample effort:

sc <- 3 *# no. CA responses*

sd <- 10 *# no. DA responses*

tn <- c( rep('CA',sc),rep('DA',sd) ) *# combine CA and DA obs*

S <- length(tn)

n <- 500

emat <- matrix( runif(n,.5,5), n, sd ) *# simulated DA effort*

eff <- list( columns = c((sc+1):S), values = emat )

f <- gjamSimData( n = n, typeNames = tn, effort = eff )

ml <- list( ng = 2000, burnin = 500, typeNames = f$typeNames, effort = f$effort )

out <- gjam( f$formula, f$xdata, f$ydata, modelList = ml )

gjamPredict( out, y2plot = colnames(f$ydata)[tn == 'DA'] ) *# predict DA data*

The prediction plot fits the data well, because it assumes the same effort. However, I might wish to predict data with a standard level of effort, say ‘1’. This new effort is taken in the same units as was used to fit the data, e.g., plot area, time observed, and so on. I use the same design matrix, but specify this new effort. Here I first predict the data with the actual effort, followed by the new effort of 1

cols <- c( '#1b9e77','#d95f02' )

new <- list( xdata = f$xdata, effort=eff, nsim = 500 ) *# effort unchanged*

p1 <- gjamPredict( output = out, newdata = new )

plot(f$y[,tn == 'DA'], p1$sdList$yMu[,tn == 'DA'],

xlab = 'Observed', ylab = 'Predicted', cex=.2, col = cols[1] )

abline( 0, 1, lty = 2 )

new$effort$values <- eff$values\*0 + 1 *# predict for effort = 1*

p2 <- gjamPredict( output = out, newdata = new )

points( f$y[,tn == 'DA'], p2$sdList$yMu[,tn == 'DA'], col= cols[2], cex=.2 )

legend( 'topleft', c('Actual effort', 'Effort = 1'), text.col = cols, bty='n' )

The orange dots show what the model would predict had effort on all observations been equal to 1.

**conditional prediction**

gjam can predict a subset of columns in y conditional on other columns using the function gjamPredict. Here is an example using the model fitted in the previous section. Consider model prediction in the case where the second plant species is absent, and the first species is at its mean value. In other words, for these plant species abundances, what is the effect of model predictors? I compare it to predictions where I first condition on the observed values for the first two plant species. I do not specify a new version of xdata, but rather include the columns of y to condition on:

cols <- c( '#1b9e77','#d95f02','#e7298a' )

condCols <- 1:3

pCols <- c(1:S)[-condCols]

new <- list(ydataCond = f$y[,condCols], nsim=200) *# cond on observed CA data*

p1 <- gjamPredict(output = out, newdata = new)$sdList$yMu[,pCols]

new <- list(ydataCond = f$y[,condCols]\*0, nsim=200) *# spp 1, 2 absent*

p2 <- gjamPredict(output = out, newdata = new)$sdList$yMu[,pCols]

obs <- f$y[,pCols]

plot( jitter(obs), p2, xlab='Observed', ylab = 'Predicted', cex=.1,

ylim=range(c(p1,p2)), col = cols[1] )

points( jitter(obs), out$prediction$ypredMu[,pCols], col=cols[2], cex=.1)

points( jitter(obs), p1, col=cols[3], cex=.1)

abline( 0, 1, lty=2 )

legend( 'topleft',

c('Conditioned on absent spp 1:3', 'Unconditional', 'Conditioned on observed spp 1:3'),

text.col = cols, bty='n')

In the first case, I held the values for columns 1 and 2 at the values observed. This conditioning on observed values changes the predictions of other variables. In the second case, I conditioned on the specific values of y mentioned above. Both differ from the unconditional prediction.

When there are large covariances in the estimates of Σ the conditional predictions can differ dramatically from anything observed. In fact, if I condition on values of y that are well outside the data predictions will make no sense at all. Conversely, if covariances in Σ are near zero conditional distributions will not look much different from unconditional predictions. With dimension reduction we have only a crude estimate of Σ, so conditional prediction was be judged accordingly.

Here is a second example, where I ask how knowledge of some species affects ability to predict others.

library(repmis)

d <- "https://github.com/jimclarkatduke/gjam/blob/master/forestTraits.RData?raw=True"

source\_data(d)

xdata <- forestTraits$xdata *# n X Q*

ydata <- gjamReZero( forestTraits$treesDeZero ) *# n X S*

ydata <- ydata[,colnames(ydata) != 'other']

ydata <- gjamTrimY( ydata, 200, OTHER=F )$y

Here is a model fitted to the forest "DA" data:

ml <- list( ng = 2500, burnin = 1000, typeNames = 'DA',

PREDICTX = F )

formula <- as.formula(~ temp + deficit\*moisture)

output <- gjam( formula, xdata, ydata, modelList = ml )

*# prediction*

newdata <- list( nsim=100 )

tmp <- gjamPredict( output, newdata=newdata )

full <- tmp$sdList$yMu

Here I condition on half of the species and predict the other half:

cols <- c( '#1b9e77','#d95f02' )

cnames <- sample( colnames(ydata), S/2)

wc <- match(cnames, colnames(ydata))

newdata <- list(ydataCond = ydata[,cnames], nsim=100)

tmp <- gjamPredict(output, newdata = newdata)

condy <- tmp$sdList$yMu

plot(ydata[,-wc], full[,-wc], ylim = c(range(ydata)),

xlab = 'Observed', ylab = 'Predicted', col = cols[1], cex = .3)

abline( 0, 1, lty=2 )

points( ydata[,-wc], condy[,-wc], col = cols[2], cex = .3)

legend( 'topleft', c('Unconditional', 'Conditional on half of species'),

text.col = cols[1:2], bty='n' )

Again, the conditional prediction can differ from the unconditional one, due to the covariances between species.

**conditional parameters**

The parameters used for conditional prediction can be obtained with the function gjamConditionalParameters. The coeffficients come from a partition of the response vector as w′=[u,v] with coefficient and covariance matrices

B=[BuBv],Σ=[ΣuuΣvuΣuvΣvv]

Write the conditional distribution for responses in u conditioned on values set for the others in v,

u|vACP∼MVN(Cx+Av,P)=ΣuvΣ−1vv=Bu−ABv=Σuu−AΣvu

gjamConditionalParameters takes the fitted gjam object and returns these three matrices, each as a matrix of posterior means and standard errors and as a table with 95% credible intervals. The vector conditionOn holds the column names in ydata for the variables that I want to condition on, i.e., in the subvector v,

gjamConditionalParameters( output, conditionOn = c('acerSacc','betuAlle','faguGran','querRubr') )

Matrix A is like a regression matrix on species. It is a Su×Sv matrix holding the effects of each response in conditionOn (v) on the other columns (u).

Matrix C are regression coefficients on predictors in x, after extracting the contribution from other species. It is a Su×Q matrix.

Matrix P is the Su×Su conditional covariance.

**presence-only data with effort**

If I have a model for effort, then incidence data can have a likelihood, i.e., a probability assigned to observations that are aggregated into groups of known effort. I cannot model absence for the aggregate data unless I know how much effort was devoted to searching for it. Effort is widely known to have large spatio-temporal and taxonomic bias.

If I know the effort for a region, even in terms of a model (e.g., distance from universities and museums, from rivers or trails, numbers of a species already in collections), I can treat aggregate data as counts. If I do not know effort, out of desperation I might use the total numbers of all species counted in a region as a crude index of effort. The help page for function gjamPoints2Grid provides examples to aggregate incidence data with (x, y) locations to a lattice.

If effort is known I can supply a prediction grid predGrid for the known effort map and aggregate incidence to that map. I can then model the data are 'DA' data, specifying effort as in the example above.

If effort is unknown, I can model the data as composition data, 'CC'. Again, this is a desperate measure, because there are many reasons why even the total for all species at a lattice point might not represent relative effort.

**dimension reduction**

Microbiome, genetic, and hyperspectral satelitte data are examples of observations characterized by a large number of response variables S (e.g., species); we refer to such data sets as *‘big-S’*. Covariance Σ has dimension S×S, with S(S+1)/2 unique elements, the *S* diagonal elements plus 1/2 of the off-diagonal elements. For example, a data set with S=100 has 5050 unique elements in Σ. It must be inverted, an order(S3) operation. Even in cases where Σ can be inverted the number of observations may not be sufficient to accurately estimate the large number of parameters in the model. In gjam, *big-S* is handled by generating a low-order approximation of Σ. The rank of Σ is reduced by finding structure, essentially groups of responses that respond similarly. (Taylor-Rodriguez et al. 2017).

The interpretation of a reduced model warrants a few words. If we replace Σ with a much smaller number of estimates, we cannot insist that we can know the covariance between every species. If Σ does not contain structure that can be adequately summarized with fewer estimates, then we have at best a version of the model that soaks up some of the dependence structure that is important for estimating B. On the other hand Σ may contain substantial structure that can be captured by a small number of estimates. We first point out that an analysis of *big-S* data sets need not include every species that might be recorded in a data set and how gjam functions can be used to trim large data sets. We then describe dimension reduction in gjam.

A species *s* that bears no relationship to any of the predictors in X (all Bs small) or to other species *s’* (all Σs′,s small) will not be ‘explained’ by the model. Such species will contribute little to the model fit, while degrading performance. Consider either of two options for reducing the number of species in the model, *trimming* and *aggregation*.

**Trim** species that are not of interest, that will not affect the fit, or both.

**Aggregation** can be based on a number of criteria, such as phylogenetic similarity (e.g., members of the same genus), by functional similarity (e.g., a feeding guild, C3 vs C4 plants), and so forth. Rare species can be aggregated into a single group. For example, Clark et al. (2014) include 96 tree species that occur on a minimum of 50 forest inventory plots in eastern North America. The remaining species can be gathered into a single class. When this option is used the name *‘other’* is assigned to this class in the plots-by-species matrix ydata. Including this class is important where species compete, such as forest trees. It can also be used as a reference category for composition data, summarized below

In gjam, the total number of covariance parameter estimates is reduced to N×r, where r<N<<S. The integer N represents the potential number of response groups. The integer r is the dimensionality of each group. In other words, large *N* means more groups, and large *r* increases the flexibility of those *N* groups.

Dimension reduction is invoked in one of two ways. **The first way** is automatic, when i) a data set includes more species than can be fitted given sample size *n* or when ii) *S* is too large irrespective of *n*.

**A second way** to invoke dimension reduction is to specify it in modelList, through the list reductList. Here is an example using simulated data, where the number of species is twice the number of observations.

f <- gjamSimData( n = 100, S = 200, typeNames='CA' )

ml <- list( ng = 2000, burnin = 500, typeNames = f$typeNames,

reductList = list(r = 15, N = 25), PREDICTX = F )

out <- gjam( f$formula, f$xdata, f$ydata, modelList = ml )

gjamPlot( output = out, plotPars = list(trueValues = f$trueValues) )

The full matrix is not stored, so gjam needs time to construct versions of it as needed. The setting PREDICTX = F can be included in modelList to speed up computation, when prediction of inputs is not of interest.

The massive reduction in rank of the covariance matrix means that the we cannot estimate the ‘true’ version of Σ, particularly given the fact that the simulator does not generate a structured Σ. These appear as highly structured GRIDPLOTS for the posterior mean estimates of the correlation matrix R. However, we can still obtain estimates of B and predictions of Y that are close to true values.

To override the automatic dimension reduction for a large response matrix, specify modelList$REDUCT <- FALSE.

***big-S* composition data: fungal endophytes**

Microbiome data are often *big-S, small-n*; with thousands of response variables, columns in Y (e.g., OTUs). They are also composition count ('CC') data, discrete counts, but not related to absolute abundance; they are meaningful in a relative sense. Because data only inform about relative abundance, there is information for only S−1 species. If there are thousands of OTUs, most of which are rare and thus not explained by the model, consider aggregating the many rare types into a single other class.

Fungal endophytes were sequenced on host tree seedlings (Hersh et al. 2016). In the data set fungEnd there is a compressed version of responses yDeZero containing OTU counts, a data.frame xdata containing predictors, and status, a vector of host responses, 0 for morbid, 1 for no signs of morbidity. Several histograms show the overwhelming numbers of zeros. Here we extract the data, stored in de-zeroed format, and generate some plots:

library(repmis)

d <- "https://github.com/jimclarkatduke/gjam/blob/master/fungEnd.RData?raw=True"

source\_data(d)

xdata <- fungEnd$xdata

otu <- gjamReZero(fungEnd$yDeZero)

status <- fungEnd$status

par( mfrow=c(1,3), bty='n' )

hist( status, main='Host condition (morbid = 0)', ylab = 'Host obs' )

hist( otu, nclass=100, ylab = "Reads", main = "OTU's" )

nobs <- gjamTrimY( otu, minObs = 1, OTHER = F )$nobs

hist( nobs/ncol(otu), nclass=100, xlab = 'Fraction of observations',

ylab = 'Frequency', main='Incidence' )

The graph on the left shows that two thirds of host seedlings are in the morbid condition. The middle graph shows that an individual OTU may be counted > 40,000 times in an obseration, but the vast majority of observations (96%) are zero. The graph on the right shows the frequency for the fraction of observations in which each OTU occurs. The most common OTU occurs in 23% of observations, but most occur only once or twice.

The model will provide no information on the rarest taxa. Here we trim otu to include only OTUs that occur in > 100 observations. The rarest OTUs are aggregated into the last column of y with the column name other:

tmp <- gjamTrimY(otu, minObs = 100)

y <- tmp$y

dim(fungEnd$y) *# all OTUs*

dim(y) *# trimmed data*

tail(colnames(y)) *# 'other' class added*

The full response matrix includes the OTU composition counts and the host status in column 1:

ydata <- cbind(status, y) *# host status is also a response*

S <- ncol(ydata)

typeNames <- rep('CC',S) *# composition count data*

typeNames[1] <- 'PA' *# binary host status*

The interactions in the model involve two factors poly (two levels, polyculture vs monoculture) and host (eight factor levels, one for each host species). I assign acerRubr as the reference class for host,

xdata$host <- relevel(xdata$host,'acerRubr')

For this example we specify up to N=20 clusters with r=3 columns each. Here is an analysis of host seedling and polyculture effect on combined host morbidity status and the microbiome composition:

ml <- list( ng = 2000, burnin = 500, typeNames = typeNames,

reductList = list(r = 8, N = 15) )

output <- gjam(~ host\*poly, xdata, ydata, modelList = ml)

Here is output:

S <- ncol(ydata)

specColor <- rep('blue',S)

specColor[1] <- '#b2182b' *# highlight host status*

fit <- gjamPlot( output, plotPars = list(specColor = specColor, GRIDPLOTS = T,

SIGONLY = T) )

fit$eComs[1:5,]

Check the chains for convergence. Those coefficients that do not converge are poorly represented in factorial factor combinations.

Again, the low dimensional version of covariance Σ is expected to perform best when there is structure in the data. The responses in matrix E, returned in fit$ematrix, classify OTUs in three main groups, contained in fit$eComs.

**interactions and indirect effects**

A plot of main effects, interactions, and indirect effects is used in this example to show contributions to host status, the response variable status. The effects on host status of on responses is available as a table with standard errors and credible intervals:

beta <- output$parameters$betaTable

ws <- grep( 'status\_',rownames(beta) ) *# find coefficients for status*

beta[ws,]

Following the intercept are rows showing main effects. These are followed by interaction terms. A quick visual of coefficients having credible intervals that exclude zero is here:

ws <- which(beta$sig95 == '\*')

beta[ws,]

with - and + indicating negative and postive values.

Here I use the function gamIIE to create the object fit1, a list of these main, interaction, and indirect effects. I specify not to include the response variable other as an indirect effect on status, because we want to focus on the effects of microbes that have been assigned to known taxonomic groups.

I specify the values for main effects that are involved in the interactions between poly and host. Each factor has one less column in the design matrix x than factor levels. poly has two classes in xdata, one each for monoculture and polyculture, so there is one poly column in x. host has eight species in xdata, so there are seven columns in x. Recall that we assigned acerRubr to be the reference class, so there is no column for it in x.

The vector of predictor values xvector passed to gjamIIE has the same elements and names as columns in x. For this reason it is easiest to simply assign it a row in x, then change the values. The only values that influence interactions are those that are involved in interaction terms, as specified in formula.

For the following plots I copy the first row of x. In the first are main effects of all predictors on status. In the second plot is interactions with poly set to 1. In the third plot are indirect effects of the microbes:

x <- output$inputs$xUnstand

xvector <- x[1,]\*0

names(xvector) <- colnames(x)

xvector['hostfraxAmer'] <- 1

xvector['polypoly'] <- 1

fit1 <- gjamIIE(output, xvector, omitY = 'other')

par(mfrow=c(1,3), bty='n', oma = c(1,2,0,0),

mar = c(3,3,3,1), tcl = -0.5, mgp = c(3,1,0))

gjamIIEplot(fit1, response = 'status', effectMu = 'direct',

effectSd = 'direct', legLoc = 'bottomright', ylim=c(-10,10))

title('Direct effect by host')

gjamIIEplot(fit1, response = 'status', effectMu = 'int', effectSd = 'int',

legLoc = 'topright', ylim=c(-10,10))

title('Interactions with polyculture')

gjamIIEplot(fit1, response = 'status', effectMu = 'ind', effectSd = 'ind',

legLoc = 'topright', ylim=c(-5,5))

title('Indirect effect of microbiome')

The plot at left is the direct effect, which includes both the main effects plus interactions and plotted relative to the mean over all hosts. The interaction contribution at center is the effect of each host, when grown in polyculture (poly = ref1) and of polyculture when the host = 'fraxAmer.

The indirect effects bring with them the main effects and interaction effects on each microbial taxon. In this example the indirect effects are noisy, showing large 95% intervals.

**joint trait modeling**

Because it accommodates different data types gjam can be used to model ecological traits by either of two approaches ([Clark 2016](https://esajournals.onlinelibrary.wiley.com/doi/full/10.1002/ecy.1453)). One approach uses community weighted mean/mode (CWMM) trait values for a plot i as a response vector ui, where each trait has a corresponding data type designation in typeNames. I discuss this approach first. I then summarize the second approach, predictive trait modeling.

**trait response model (TRM)**

There are n observations of M traits to be explained by Q−1 predictors in design matrix X. The Trait Response Model (TRM) in [Clark 2016](https://esajournals.onlinelibrary.wiley.com/doi/full/10.1002/ecy.1453) is

wi∼MVN(A′xi,Ω)

where wi is a length-M vector of expected CWMM values, A is the Q×M matrix of coefficients, and Ω is the M×M residual covariance (diagram below). After describing the setup and model fitting I show how gjam summarizes the estimates and predictions.

A rectangular rectangles with black text

AI-generated content may be incorrect.

**Trait response model** showing the sizes of matrices for a sample containing *n* observations, *M* traits, and *Q* predictors.

**input data**

Data contained in forestTraits include predictors in xdata, a character vector of data types in traitTypes, and treesDeZero, which contains tree biomass in de-zeroed format. Here the data are loaded, re-zeroed with gjamReZero:

library(repmis)

d <- "https://github.com/jimclarkatduke/gjam/blob/master/forestTraits.RData?raw=True"

source\_data(d)

xdata <- forestTraits$xdata *# n X Q*

types <- forestTraits$traitTypes *# 12 trait types*

sbyt <- forestTraits$specByTrait *# S X 12*

pbys <- gjamReZero(forestTraits$treesDeZero) *# n X S*

pbys <- gjamTrimY(pbys,5)$y *# at least 5 plots*

sbyt <- sbyt[match(colnames(pbys),rownames(sbyt)),] *# trait matrix matches ydata*

identical(rownames(sbyt),colnames(pbys))

The matrix pbys holds biomass values for species. The first six columns of sbyt are centered and standardized. The three ordinal classes are integer values, but do not represent an absolute scale (see below). The three groups of categorical variables in data.frame sbyt have different numbers of levels shown here:

table(sbyt$leaf) *# four levels*

table(sbyt$xylem) *# diffuse/tracheid vs ring-porous*

table(sbyt$repro) *# two levels*

These species traits are translated into community-weighted means and modes (CWMM) by the function gjamSpec2Trait:

tmp <- gjamSpec2Trait(pbys, sbyt, types)

tTypes <- tmp$traitTypes *# M = 15 values*

u <- tmp$plotByCWM *# n X M*

censor <- tmp$censor *# (0, 1) censoring, two-level CAT's*

specByTrait <- tmp$specByTrait *# S X M*

M <- ncol(u)

n <- nrow(u)

cbind(colnames(u),tTypes) *# M trait names and types*

**traits by species**

Note the change in data types by comparing types for individuals of a species with tTypes for CWMM values at the plot scale. At the plot scale tTypes has M=15 values, because the leaf 'CAT' group in types includes four levels, which are expanded to four 'FC' columns in u. The two-level groups 'xylem' and 'repro' are transformed to censored continuous values on (0, 1) and thus each occupy a single column in u.

As discussed in [Clark (2016)](https://esajournals.onlinelibrary.wiley.com/doi/full/10.1002/ecy.1453) the interpretation of CWMM values in u is not the same as the interpretation of species-level traits assigned in forestTraits$specByTrait. Let T′ be a species-by-traits matrix specByTrait, constructed as CWMM values in function gjamSpec2Trait. The row names of specByTrait match the column names for the n×S species abundance matrix plotByTrees. The latter is referenced to individuals of a species.

The plot-by-trait matrix u is referenced to a location, i.e., one row in matrix u. It is a CWMM, with values derived from measurements on individual trees, but combined to produce a weighted value for each location. Ordinal traits (shade, drought, flood) are community weighted modes, because ordinal scores cannot be averaged. The CWMM value for a plot may not be the same data type as the trait measured on an individual tree sbyt. Here is a table of 15 columns in u:

| **trait** | **typeName** | **partition** | **comment** |
| --- | --- | --- | --- |
| gmPerSeed | CON | (−∞,∞) | centered, standardized |
| maxHt | CON | " | " |
| leafN | CON | " | " |
| leafP | CON | " | " |
| SLA | CON | " | " |
| woodSG | CON | " | " |
| shade | OC | (−∞,0,ps1,ps2,ps3,ps4,∞) | five tolerance bins |
| drought | OC | " | " |
| flood | OC | " | " |
| leaf\_broaddeciduous | FC | (−∞,0,1,∞) | categorical traits become FC data as CWMs |
| leaf\_broadevergreen | FC | " | " |
| leaf\_needleevergreen | FC | " | " |
| leaf\_other | FC | " | " |
| repro\_monoecious | CA | (−∞,0,1,∞) | two categories become continuous (censored) |
| xylem\_ring | CA | " | " |

The first six CON variables are continuous, centered, and standardized, as is often done in trait studies. In gjam CON is the only data type that is not assumed to be censored at zero.

The three OC variables are ordinal classes, lacking an absolute scale–the partition must be estimated.

The four fractional composition FC columns are the levels of the single CAT variable leaf, expanded by the function gjamSpec2Trait.

The last two traits in u are fractions with two classes, only one of which is included here. They are censored at both 0 and 1, the intervals (−∞,0) and (1,∞). This censoring can be generated using gjamCensorY:

censorList <- gjamCensorY(values = c(0,1), intervals = cbind( c(-Inf,0),c(1,Inf) ),

y = u, whichcol = c(13:14))$censor

This censoring was already done with gjamSpec2Trait, which knows to treat 'CAT' data with only two levels as censored 'CA' data. In this case the values = c(0,1) indicates that zeros and ones in the data indicate censoring. The intervals matrix gives their ranges.

**factors in this example**

Multilevel factors in xdata require some interpretation. If you have not worked with multilevel factors, refer to the R help page for factor. The interpretation of coefficients for multilevel factors depends on the reference level used to construct a *contrasts* matrix. Standard models in R assign contrasts that may not assume the reference level that is desired. Moreover, if the reference is unspecified, it depends on on the order of observations and variables in the data.

In xdata the variable soil is a multilevel factor, which includes soil types that are both common and have potentially strong effects. Here are the first few rows of xdata:

I used the name reference for a soil type to aggregate types that are rare. Factor levels that rarely occur cannot be estimated in the model.

The R function relevel allows definition of a reference level. In this case I want to compare levels to the reference soil type reference:

xdata$soil <- relevel(xdata$soil,'reference')

To avoid confusion, contrasts can be inspected as output$modelSummary$contrasts. If the reference class is all zeros and other classes are zeros and ones, then the intercept is the reference class.

**TRM analysis**

Here is an analysis of the data, with 20 holdout plots. Predictors in xdata are winter temperature (temp), local moisture, climatic moisture deficit and soil. As discussed above, the variable soil is a multi-level factor.

ml <- list(ng = 3000, burnin = 500, typeNames = tTypes, holdoutN = 20,

censor=censor, notStandard = c('u1','u2','u3'))

out <- gjam(~ temp + stdage + moisture\*deficit + deficit\*soil,

xdata = xdata, ydata = u, modelList = ml)

tnames <- colnames(u)

sc <- rep('black', M) *# highlight types*

wo <- which(tnames %in% c("leafN","leafP","SLA") ) *# foliar traits*

wf <- grep("leaf",tnames) *# leaf habit*

wc <- which(tnames %in% c("woodSG","diffuse","ring") ) *# wood anatomy*

sc[wc] <- 'brown'

sc[wf] <- 'darkblue'

sc[wo] <- 'darkgreen'

pl <- list(GRIDPLOTS = TRUE, PLOTALLY = T, specColor = sc)

gjamPlot(output = out, plotPars = pl)

summary(out)

The model fit is interpreted in the same way as other gjam analyses. Note that specColor is used to highlight different types of traits in the posterior plots for values in coefficient matrix A. Parameter estimates are contained in parameters,

out$parameters$betaMu *# Q by M coefficient matrix alpha*

out$parameters$betaStandXmu *# Q by M standardized for X*

out$parameters$betaStandXWmu *# (Q-F) by M standardized for W/X, centered factors*

out$parameters$betaTable *# QM by stats posterior summary*

out$parameters$sigMu *# M by M covariance matrix omega*

out$parameters$sigSe *# M by M covariance std errors*

The output list contains a large number of diagnostics explained in help pages.  
The matrices out$parameters$cutMu and out$parameters$cutSe hold the estimates for the partion for ordinal ('OC') variables shade, drought, and flood tolerance. Each has three fitted cutpoints, because the first is fixed, with three estimates to partition the remaining four intervals.

**interactions and indirect effects**

Consider the interactions and indirect effects for this model. If there are no interactions in the formula passed to gjam, then there will be no interactions to estimate with the function gjamIIE (there will still be indirect effects, discussed below). If there are interactions in the formula, I must specify the values for main effects that are involved in these interactions to be used for estimating their effects on predictions. For example, consider a model containing the interaction between predictors q and q′,

E[ys]=⋯+βq,sxq+βq′,sxq′+βqq′,sxqxq′+⋯

The ‘effect’ of predictor xq on ys is the derivative

dysdxq=βq,s+βqq′,sxq′

which depends not on xq, but rather on xq′. So if I want to know how interactions affect the response I have to decide on values for all of the predictors that are involved in interactions. These values are passed to gjamIIE in xvector. The default has sdScaleX = F, which means that effects can be compared on the basis of variation in X.

In this example interactions involve moisture, deficit, and the multi-level factor soil, as specified in the formula passed to gjam. The first row of the design matrix is used with moisture and deficit set to -1 or +1 standard deviation to compare dry and wet sites in a dry climate:

xdrydry <- xwetdry <- out$inputs$xUnstand[1,]

xdrydry['moisture'] <- xdrydry['deficit'] <- -1

xwetdry['moisture'] <- 1

xwetdry['deficit'] <- -1

The first observation is from the reference soil level reference, so all other soil classes are zero. Here is a plot of main effects and interactions for deciduous and evergreen traits:

par(mfrow=c(2,2), bty='n', mar=c(1,3,1,1), oma = c(0,0,0,0),

mar = c(3,2,2,1), tcl = -0.5, mgp = c(3,1,0), family='')

fit1 <- gjamIIE(output = out, xvector = xdrydry)

fit2 <- gjamIIE(output = out, xvector = xwetdry)

gjamIIEplot(fit1, response = 'leafbroaddeciduous',

effectMu = c('main','int'), effectSd = c('main','int'),

legLoc = 'bottomleft', ylim=c(-.31,.3))

title('deciduous')

gjamIIEplot(fit1, response = 'leafneedleevergreen',

effectMu = c('main','int'), effectSd = c('main','int'),

legLoc = 'bottomleft', ylim=c(-.3,.3))

title('evergreen')

gjamIIEplot(fit2, response = 'leafbroaddeciduous',

effectMu = c('main','int'), effectSd = c('main','int'),

legLoc = 'bottomleft', ylim=c(-.3,.3))

gjamIIEplot(fit2, response = 'leafneedleevergreen',

effectMu = c('main','int'), effectSd = c('main','int'),

legLoc = 'bottomleft', ylim=c(-.3,.3))

The main effects plotted in the graphs do not depend on the values in xvector. Although this observation is taken from the reference soil, the plot shows the main effects that would be obtained if it were on the different soils included in the model. The interactions show how the effect of each predictor is modified by interactions with other variables. Again, the interactions from each predictor do not depend on values for the predictor itself, but rather on the other variables with which it interacts. For example, the interaction effect of soilUltKan on the broaddeciduous trait is positive on dry sites in dry climates (top left). Combined with a negative main effect, this means that deciduous trees tend to be more abundance on moist sites in this soil type. Its main effect on leafneedleevergreen is positive, but less so on moist sites in dry climates (bottom right).

The indirect effects come from the effects of responses. This example shows indirect effects for foliar N and P that come through broaddeciduous leaf habit:

xvector <- out$inputs$xUnstand[1,]

par(mfrow=c(2,1), bty='n', mar=c(1,1,1,1), oma = c(0,0,0,0),

mar = c(3,2,2,1), tcl = -0.5, mgp = c(3,1,0), family='')

omitY <- colnames(u)[colnames(u) != 'leafbroaddeciduous'] *# omit all but deciduous*

fit <- gjamIIE(out, xvector)

gjamIIEplot(fit, response = 'leafP', effectMu = c('main','ind'),

effectSd = c('main','ind'), legLoc = 'topright', ylim=c(-.6,.6))

title('foliar P')

gjamIIEplot(fit, response = 'leafN', effectMu = c('main','ind'),

effectSd = c('main','ind'), legLoc = 'bottomright', ylim=c(-.6,.6))

title('foliar N')

There will always be indirect effects, because they come through the covariance matrix.

**predictive trait model (PTM)**

The PTM models species abundance data, then predicts traits. This approach has a number of advantages over TRM discussed in [Clark 2016](https://esajournals.onlinelibrary.wiley.com/doi/full/10.1002/ecy.1453). The response is the n×S matrix Y, which could be counts, biomass, and so forth.

**start with species abundance**

PTM is based on species abundance, but includes trait data for CWMM prediction. On the latent scale the observation is represented by a composition ('FC' or 'CC') vector,

wi∼MVN(B′xi,Σ)

where B is the Q×S matrix of coefficients, and Σ is the S×S residual covariance. A predictive distribution on the trait scale is obtained as a *variable change*,

A=BT

Ω=T′ΣT

ui=T′wi

where T is a S×M matrix of trait values for each species, A is the Q×M matrix of coefficients, and Ω is the M×M residual covariance (diagram below).

A rectangular object with a rectangular object in the middle

AI-generated content may be incorrect.

**The predictive trait model** fits species data and predicts traits using the species-by-trait matrix **T**, contained in the object specbyTrait.

The PTM begins by fitting pbys, followed by predicting plotByTraits. This requires a traitList, which defines the objects needed for prediction. The species are weights, so they should be modeled as composition data, eight 'FC' (rows sum to 1) or 'CC'. Here the model is fitted with dimension reduction:

tl <- list(plotByTrait = u, traitTypes = tTypes, specByTrait = specByTrait)

rl <- list(r = 8, N = 25)

ml <- list(ng = 2000, burnin = 500, typeNames = 'CC', holdoutN = 20,

traitList = tl, reductList = rl)

out <- gjam(~ temp + stdage + deficit\*soil, xdata = xdata,

ydata = pbys, modelList = ml)

S <- nrow(specByTrait)

sc <- rep('black', S)

wr <- which(specByTrait[,'ring'] == 1) *# ring porous*

wb <- which(specByTrait[,'leafneedleevergreen'] == 1) *# needle-leaf evergreen*

wd <- which(specByTrait[,'leafneedledeciduous'] == 1) *# needle-leaf deciduous*

ws <- which(specByTrait[,'shade'] >= 4) *# shade tolerant*

sc[wr] <- '#8c510a'

sc[ws] <- '#3288bd'

sc[wb] <- '#003c30'

sc[wd] <- '#80cdc1'

M <- ncol(specByTrait)

tc <- rep('black', M)

names(tc) <- colnames(specByTrait)

tc[ 'ring' ] <- '#8c510a'

tc[ 'leafneedleevergreen' ] <- '#3288bd'

tc[ 'leafneedledeciduous' ] <- '#003c30'

tc[ 'shade' ] <- '#80cdc1'

par(family = '')

pl <- list(GRIDPLOTS=T,

specColor = sc, traitColor = tc, ncluster = 5)

fit <- gjamPlot(output = out, pl)

Output is interpreted as previously, now with coefficients B and covariance Σ. gjamPlot generates an additional plot with trait predictions. Parameter values are here:

out$parameters$betaTraitXMu *# Q by M coefficient matrix alpha, standardized for X*

out$parameters$betaTraitXWmu *# Q by M standardized for X/W*

out$parameters$betaTraitXTable *# QM by stats posterior summary*

out$parameters$betaTraitXWTable *# QM by stats summary for X/W*

out$parameters$varTraitMu *# M by M trait residual covariance, standardized for X*

out$parameters$varTraitTable *# M^2 by stats summary, standardized for X/W*

Trait predictive distributions are summarized here:

out$prediction$tMu[1:5,] *# n by M predictive means*

out$prediction$tSe[1:5,] *# n by M predictive std errors*

Additional quantities can be predicted from the output using the MCMC output in the list out$chains.

It is worth checking for combinations of responses and factor levels that might be missing from the data. That is an issue here, because each species has a limited geographic distribution, as do the soil types. The object out$inputs$factorBeta$missFacSpec lists the missing predictor-response combinations. This list could be the basis for consolidating factor levels.

**trouble shooting in PTM**

When using gjam in predictive trait mode remember the following:

1. typeNames for ydata data should be composition, either CC or FC
2. nrow(plotByTrait) must equal nrow(ydata)
3. ncol(plotByTrait) must equal length(traitTypes)
4. ncol(plotByTrait) must equal length(traitTypes)
5. rownames(specByTrait) must match colnames(ydata)

**selecting variables**

In a univariate model, there is one response and perhaps a number of potential predictor variables from which to choose. Variable selection focuses on X. In a multivariate model, the overall fit and predictive capacity depends not only on what’s in X, but also on what’s in Y. A different combination of variables in X will provide the best “fit” for each combination of variables in Y. Given that many species may be rare, and rare types will not be explained by the model, there will be decisions about what variables to include on both sides of the likelihood.

These considerations mean that simple rules like ‘lowest DIC’ are not sensible. Do I want the model that best explains the subset of response variables having the most ‘signal’? Not necessarily, because many less abundant types may be of interest. Alternatively, the best model for responses ranging from rare to abundant will depend on precisely which of the rare types are included. As discussed in the section on **dimension reduction**, rare types also affect the coefficients for abundant types through covariance Σ. I often want to consider a range of variable combinations. Here are a few objects available in gjam output that can provide some guidance for selecting variables.

| **output or .pdf plot file** | **explanation** |
| --- | --- |
| output$fit$DIC | Low is good |
| output$inputs$designTable | Redundant predictors based on high correlation and/or high VIF (> 10 to 20)? |
| output$inputs$factorBeta$missFacSpec | Missing predictor-response combinations listed here–consider consolidating factor levels or removing predictors and/or responses. |
| output$parameters$sensTable, sensitivity.pdf | High values account for most variation across all responses. |
| betaAll.pdf | Do CIs include zero for all responses? |
| betaChains.pdf, output$chains$bgibbs | Any not converged? |
| yPredAll.pdf | Generated if PLOTALLY = T in plotPars; remove some responses that are not well predicted? |
| yPred.pdf | Model predicts responses in-sample? |
| xPred.pdf, xPredFactors.pdf | Model inversely predicts inputs? |

To cull rare types in Y see the help page for gjamTrimY. To evaluate the model with out-of-sample prediction see the section **missing data, out-of-sample prediction**.

**trouble shooting**

A joint model for data sets with many response variables can be unstable for several reasons. Because the model can accommodate large, multivariate data sets, there is temptation to throw everything in and see what happens. gjam is vulnerable due to the fact that columns in ydata have different scales and, thus, can range over orders of magnitude. It’s best to start small, gain a feel for the data and how it translates to estimates of many coefficients and covariances. More species and predictors can be added without changing the model. The opposite approach of throwing in everything is asking for trouble and is unlikely to generate insight.

If a model won’t execute, start by simplifying it. Use a small number of predictors and response variables. It’s easy to add complexity later. If execution fails there are several options.

**Check**X**and**Y. Most errors come from the data. If there are many species (large Y), some may occur once or not at all. This is easy to check with the function gjamTrimY. If I suspect trouble with the design matrix X, I can make sure it is full rank, e.g.,

f <- gjamSimData(S = 5, typeNames = 'CA')

x <- model.matrix(f$formula, f$xdata)

qr(x)$rank

The rank must equal the number of columns in x.

**If I am simulating data**, first try it again. The simulator aims to generate data that will actually work, more challenging than would be the case for a univariate simulation of a single data type. Simulated data are random, so try again.

**If the fit is bad**, it could be noisy data (there is no ‘signal’), but there are other things to check. Again, insure that all columns in ydata include at least some non-zero values. One would not expect a univariate model to fit a data set where the response is all zeros. However, when there are many columns in ydata, the fact that some are never or rarely observed can be overlooked. The functions hist, colSums, and, for discrete data, table, can be used. The function gjamTrimY(ydata, m) can be used to limit ydata to only those columns with at least m non-zero observations.

**Large differences in scale** in ydata can contribute to instability. Unlike xdata, where the design matrix is standardized, ydata is not rescaled. It is used as-is, because the user may want effort on a specific scale. However, the algorithm is most stable when responses in ydata do not span widely different ranges. For continuous data ("CA", "CON"), consider changing the units in ydata from, say, g to kg or from g ml−1 to g l−1.

For discrete counts ("DA") consider changing the units for effort, e.g., m2 versus ha or hours versus days. Recall, the dimension for the latent W is Y/E. If I count hundreds or thousands of animals per hour, I could consider specifying effort in minutes, thus reducing the scale for W by a factor of 1/60. Rescaling is not relevant for "CC", where modeling is done on the [0,1] scale.

Unlike experiments, where attention is paid to balanced design, observational data often involve **factors**, for which only some species occur in all factor combinations. This inadequate distribution of data is compounded when those factors are included in interaction terms. Consider ways to eliminate factor levels/combinations that cannot be estimated from the data.

If a simulation fails due to a cholesky error, then there is a problem inverting Σ. Any of these error messages might arise, even when using dimension reduction:

error: chol(): decomposition failed

error: inv\_sympd(): matrix is singular or not positive definite

Error in invWbyRcpp(sigmaerror, otherpar$Z[otherpar$K, ]) : inv\_sympd(): matrix is singular or not positive definite

Consider either reducing the number of columns in ydata using gjamTrimY.

If execution is stopped with the message overfitted covariance, then try reducing N and r in reductList.

**reference notes**

**model summary**

An observation consists of environmental variables and species attributes, {xi,yi}, i=1,...,n. The vector xi contains predictors xiq:q=1,...,Q. The vector yi contains attributes (responses), such as species abundance, presence-absence, and so forth, yis:s=1,...,S. The effort Eis invested to obtain the observation of response s at location i can affect the observation. The combinations of continuous and discrete measurements in observed yi motivate the three elements of gjam.

A length-S vector wi∈RS represents response yi in continuous space. This continuous space allows for the dependence structure with a covariance matrix. An element wis can be known (e.g., continuous response yis) or unknown (e.g., discrete responses).

A length-S vector of integers zi represents yi in discrete space. Each observed yis is assigned to an interval zis∈{0,...,Kis}. The number of intervals Kis can differ between observations and between species, because each species can be observed in different ways.

The partition of continuous space at points pis,z∈P defines discrete intervals zis. Two values (pis,k,pis,k+1] bound the kth interval of s in observation i. Intervals are contiguous and provide support over the real line (−∞,∞). For discrete observations, k is a censored interval, and wis is a latent variable. The set of censored intervals is C. The partition set P can include both known (discrete counts, including composition data) and unknown (ordinal, categorical) points.

An observation y maps to w,

yis={ wis zis,continuouswis∈(pzis,pzis+1]discrete

Effort Eis affects the partition for discrete data. For the simple case where there is no error in the assignment of discrete intervals, zi is known, and the model for wi is

wi|xi,yi,Ei∼MVN(μi,Σ)×∏s=1SIis

μi=B′xi

Iis=∏k∈CII(yis=k)is,k(1−Iis,k)I(yis≠k)

where Iis=I(pzis<wis<pzis+1], C is the set of discrete intervals, B is a Q×S matrix of coefficients, and Σ is a S×S covariance matrix. There is a correlation matrix associated with Σ,

Rs,s′=Σs,s′Σs,sΣs′,s′−−−−−−−√

For **presence-absence** (binary) data, pis=(−∞,0,∞). This is equivalent to Chib and Greenberg’s (2008) model, which could be written Iis=I(wis>0)yisI(wis⩽0)1−yis.

For a continous variable with point mass at zero, **continuous abundance**, this is a multivariate Tobit model, with Iis=I(wis=yis)I(yis>0)I(wis⩽0)I(yis=0). This is the same partition used for the probit model, the difference being that the positive values in the Tobit are uncensored.

**Categorical** responses fit within the same framework. Each categorical response occupies as many columns in Y as there are independent levels in response s, levels being k=1,...,Ks−1. For example, if randomly sampled plots are scored by one of five cover types, then there are four columns in Y for the response s. The four columns can have at most one 1. If all four columns are 0, then the reference level is observed. The observed level has the largest value of wis,k (Table 1). This is similar to Zhang et al.’s (2008) model for categorical data.

For **ordinal counts** gjam is Lawrence et al.’s (2008) model having the partition pis=(−∞,0,pis,2,pis,3,...,pis,K,∞), where all but the first two and the last elements must be inferred. The partition must be inferred, because the ordinal scale is only relative.

Like categorical data, **composition** data have one reference class. For this and other **discrete count** data, the partition for observation i can be defined to account for sample effort (next section).

**more on parameter dimensions**

Unlike a univariate model that has one Y per observation or multivariate models where all Y’s have the same dimension, gjam has Y’s on multiple dimensions. So there are two sets of dimensions to consider, the dimensions for the X’s and those for the Y’s. To avoid more notation I just refer to the dimension of a coefficient in the unstandardized coefficient matrix Bu as W/X (Table 2). Except for responses that have no dimension (presence-absence, ordinal, categorical) W has the same dimension as Y (or Y/E, if effort is specified). gjam returns unstandardized coefficients in tables parameters$betaMuUn, parameters$betaSeUn, and in MCMC chains$bgibbsUn, because they are not prone to be misinterpreted by a user who has supplied unstandardized predictors in xdata. I refer to the coefficient matrix as Bu and corresponding unstandardized design matrix as Xu. Each row of $chains$bgibbsUn is one draw from the Q×S matrix Bu.

Models are commonly fitted to covariates X that are ‘standardized’ for mean and variance. Standardization can stabilize posterior simulation. It is desirable when coefficients are needed in standard deviations. Inside gjam design matrix X, and thus B, are standardized, thus having dimension W, not W/X. Of course, for variables in xdata supplied in standarized form, B=Bu. See **Algorithm summary**. Variables returned by gjam, including MCMC chains in $chains$bgibbs and posterior means and standard errors $parameters$betaMu and $parameters$betaSe are standardized.

The third set of scales comes from the correlation scale for the W’s. The correlation scale can be useful when considering responses that have different dimensions. In addition to Bu I provide parameters on the correlation scale. This correlation scale is Br=BD−1/2, where D=diag(Σ). If X is standardized, Br is dimensionless. The MCMC chains in $chains$bFacGibbs and the estimates in $parameterTable$fBetaMu and $parameterTable$fBetaSd are standardized for X (standard deviation scale) and for W (correlation scale). They are dimensionless.

For sensitivity over all species and for comparisons between predictors I provide standardized sensitivity in a length-Q vector f. The sensitivity matrix to X across the full model is given by

F=BΣ−1B′

Note that F takes Br, not B, and not Bu. The sensitivity matrix F is $parameters$fmatrix. The sensitivity vector f=diag(F) is vector $parameters$fMu. Details are given in Clark et al. (2017).

**more on factors in**X

The coefficient matrix B~ is useful when there are **factors** in the model. Factor treatment in gjam follows the convention where a reference level is taken as the overall intercept, and remaining coefficients are added to the intercept. The reference level can be controlled with the R function relevel. This approach makes sense in the ANOVA context, where an experiment has a control level to which other treatment levels are to be compared. A ‘significant’ level is different from the reference (e.g., control), but we are not told about its relationship to other levels. The coefficients in $parameters$betaMu and $parameters$betaSd are reported this way. Should it be needed, the contrasts matrix for this design is returned as a list for all factors in the model as $inputs$factorBeta$contrast and as a single contrasts matrix for the entire model as $inputs$factorBeta$eCont.

This standard structure is not the best way to compare factors in many ecological data sets, where a factor might represent soil type, cover type, fishing gear, and so on. In all of these cases there is no ‘control’ treatment. Here it is more useful to know how all levels relate to the mean taken across factor levels.

To provide more informative comparisons across factor levels and species I introduce a Q1×S recontrast matrix that translates B, with intercept, to all factor levels, without intercept. Consider a three-level factor with levels a, b, c, the first being the reference class. There is a matrix G

## intercept b c

## a 1 -1 -1

## b 1 1 0

## c 1 0 1

and a matrix H,

## a b c

## intercept 1 0 0

## b -1 1 0

## c -1 0 1

With L′=G−1, the recontrasted coefficients are

B~=LB

The rows of B~ correspond to all factor levels. The intercept does not appear, because it has been distributed across factor levels. The corresponding design matrix is

X~=XH

If there are multiple factors then Q1>Q, because the intercept expands to the reference classes for each factor. L is provided as $inputs$factorBeta$lCont.

With factors, the sensitivity matrix reported in $parameters$fmatrixis

F=B~Σ−1B~′

The response matrix in $parameters$ematrix is

E=B~V~B~′

where V~=cov(XH).

**algorithm summary**

Model fitting is done by Gibbs sampling. The design matrix X is centered and standardized. Parameters B and Σ are sampled directly,

1. Σ|W,B
2. B|Σ,W
3. For unknown partition (ordinal variables) the partition is sampled, P|Z,W
4. For ordinal, presence-absence, and categorical data, latent variables are drawn on the correlation scale, W|R,α,P, where R=D−1/2ΣD−1/2, α=D−1/2B, P=D−1/2P, and D=diag(Σ).  
   For other variables that are discrete or censored, latent variables are drawn on the covariance scale, W|Σ,B,P.

Parameters in $chains$bgibbsUn are returned on the original scale Xu. Let Xu be the uncentered/unstandardized version of X. Parameters are returned as Bu=(X′uXu)−1X′uXB. Likewise, $x is returned on the original scale, i.e., it is Xu.

Dimension reduction represents the covariance matrix as Σ=AA′+σ2I. The S×r matrix A generates a random effect with prior distribution ai∼A×MVN(0r,Ir). The (posterior) estimate of ai depends on observed yi. For in-sample prediction, we have wi∼MVN(μi+ai,σ2I). For out-of-sample prediction the random effect is marginalized wi∼MVN(μi,Σ). For variables on the correlation scale, both the mean and the covariance are on the correlation scale.

Below are mean and variance for prediction sampling, where μi=B′xi. Values are (mean, covariance/correlation). Most importantly, for prediction, there is no partition, because yi is unknown and, thus, wi is unconstrained by observation:

| **.** | **covariance scale** | **correlation scale** |
| --- | --- | --- |
| Not dimension reduction: |  |  |
| in- | μi,Σ | D−1/2μi,R |
| out- | μi,Σ | D−1/2μi,R |
| Dimension reduction: | . |  |
| in- | μi+ai | D−1/2(μi+ai),Is |
| out- | μi,Σ | D−1/2μi,R |

Inverse prediction of input variables provides sensitivity analysis (Clark et al. 2011, 2014). Columns in X that are linear (not involved in interactions, polynomial terms, or factors) are sampled directly from the inverted model. Others are sampled by Metropolis. Sampling is described in the Supplement file to Clark et al. (2017).

[website](http://sites.nicholas.duke.edu/clarklab/code/)

**acknowledgements**

For valuable feedback on the model and computation I thank Bene Bachelot, Alan Gelfand, Diana Nemergut, Erin Schliep, Bijan Seyednasrollah, Daniel Taylor-Rodriquez, Brad Tomasek, Phillip Turner, and Stacy Zhang. Daniel Taylor-Rodriguez implemented the dimension reduction. I thank the members of NSF’s *SAMSI* program on *Ecological Statistics* and my class *Bayesian Analysis of Environmental Data* at Duke University. Funding came from NSF’s Macrosystems Biology and EAGER programs.

**references**

Brynjarsdottir, J. and A.E. Gelfand. 2014. Collective sensitivity analysis for ecological regression models with multivariate response. *Journal of Biological, Environmental, and Agricultural Statistics* 19, 481-502.

Chib, S and E Greenberg. 1998. Analysis of multivariate probit models. *Biometrika* 85, 347-361.

Clark, JS 2016. Why species tell us more about traits than traits tell us about species, *Ecology* 97, 1979-1993.

Clark, JS, DM Bell, MH Hersh, and L Nichols. 2011. Climate change vulnerability of forest biodiversity: climate and resource tracking of demographic rates. *Global Change Biology* 17, 1834-1849.

Clark, JS, DM Bell, M Kwit, A Powell, and K Zhu. 2013. Dynamic inverse prediction and sensitivity analysis with high-dimensional responses: application to climate-change vulnerability of biodiversity. *Journal of Biological, Environmental, and Agricultural Statistics* 18, 376-404.

Clark, JS, AE Gelfand, CW Woodall, and K Zhu. 2014. More than the sum of the parts: forest climate response from joint species distribution models. *Ecological Applications* 24, 990-999

Clark, JS, D Nemergut, B Seyednasrollah, P Turner, and S Zhang. 2017. Generalized joint attribute modeling for biodiversity analysis: Median-zero, multivariate, multifarious data, *Ecological Monographs* 87, 34–56.

Lawrence, E, D Bingham, C Liu, and V N Nair (2008) Bayesian inference for multivariate ordinal data using parameter expansion. *Technometrics* 50, 182-191.

Taylor-Rodriguez, D, K Kaufeld, E Schliep, JS Clark, and AE Gelfand, 2017. Joint Species distribution modeling: dimension reduction using Dirichlet processes, *Bayesian Analysis* in press.

Zhang, X, WJ Boscardin, and TR Belin. 2008. Bayesian analysis of multivariate nominal measures using multivariate multinomial probit models. *Computational Statistics and Data Analysis* 52, 3697-3708.