gjamTime: Generalized Joint Attribute Modeling for Dynamic Data

James S. Clark

2020-02-28

Contents

location for html file: http://rpubs.com/jimclark/551105

Here are some functions to simulate data and graph food webs that are sourced from the gjam github site:

```
library(gjam)
library(devtools)
```

```
## Loading required package: usethis
```

```
d <- "https://github.com/jimclarkatduke/gjam/blob/master/gjamTimeFunctions.R?raw=True"
source_url(d)
```

```
## SHA-1 hash of file is 2d618850d9f76b0b330c91ad5491aed0c9ee8034
```

#gjamTime for community dynamics

This is a state-space version of gjam that estimates parameters for community interactions where there are multiple groups interacting over time. A group could be a location, such as the BBS data for each route or NEON for each site. It can be viewed as a generalized Lotka-Volterra model with immigration and emigration, connected to data as discussed for gjam Clark et al. 2017.

#Simulated data for diagnostics

A simulation begins with a specification of the number of species, number of sites (or time series), the mean number of time increments (it will vary stochastically), and the observation effort, which is the effort variable of gjam.

```
S     <- 6        # no. species
nsite <- 10       # no. time series
ntime <- 100      # mean no. of time steps in a series
obsEffort <- 1    # full census
```

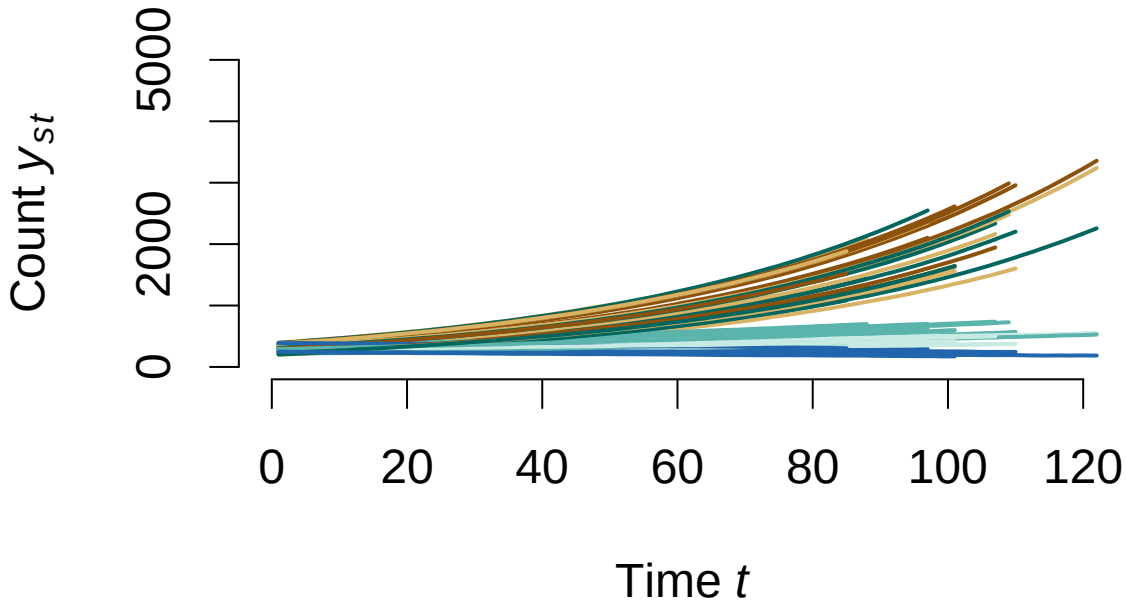
##First example: an AR model

I start with the simple case of an AR(1) model for $S = 6$ species. Here I specify a design for simulation, including `termR = TRUE`, which refers to the AR term:

```
termB <- FALSE    # include immigration/emigration term XB
termR <- TRUE     # include DI population growth term VL
termA <- FALSE    # include DD spp interaction term UA
```

Here is a simulation using the function `gjamSimTime`, which is sourced at the beginning of this vignette from github. I have set `Q = 0` to indicate that there are no predictors in the model–it includes only the growth rate parameters:

```
seed <- 999
set.seed( seed )
tmp    <- gjamSimTime(S, Q = 0, nsite, ntime, termB, termR, termA,
                      obsEffort = obsEffort, PLOT = T)
```




```
xdata  <- tmp$xdata
ydata  <- tmp$ydata
edata  <- tmp$edata
groups <- tmp$groups
times  <- tmp$times
trueValues <- tmp$trueValues
formula    <- tmp$formula
```

Most of these objects are familiar to gjam users. The `list trueValues` holds the parameter values used to generate the data.

The plot generated by `gjamSimTime` identifies the multiple series from each site by colors.

##Missing values in time series data

The function `gjamFillMissingTimes` provides some options for initializing the missing values for analysis. It takes as input the three observation objects `xdata`, `ydata`, and `edata` and returns new versions of them with added rows for missing times.

First, it will insert one observation at the beginning of the sequence for each group. This initial "time zero" is the prior for the observation at time 1.

Second, `gjamFillMissingTimes` inserts a placeholder for missing sample times. The time steps for each group are sequential integer values held in the column `timeCol`. For variables in `xdata` that change at each time step, these can either be filled by the user (preferable) or left as `NA` to be imputed as missing data. Of course, if there is too much missingness, the fit will deteriorate.

Third, `gjamFillMissingTimes` will insert values for `ydata` and `edata` to match those added to `xdata`. These can be `NA` (`FILLMEANS = FALSE`), in which case they will be imputed with an automatically assigned weak prior (small effort). Alternatively, they can be filled with the mean value for their group, with the strength of the prior controlled by the user-specified `missingEffort`. These values can be inspected from the effort matrix `edata` that is returned by `gjamFillMissingTimes`.

Recall, the effort `matrix edata` controls the weight of the observations.

The argument `groupVars` indentifies those columns in `xdata` that are fixed for the group, so they can be filled in by `gjamFillMissingTimes`. In other words, `groupVars` do not change over time. These are count data, so the `typeNames = 'DA'`, i.e., 'discrete abundance'.

```
timeCol   <- 'times'
groupCol  <- 'groups'
groupVars <- c( 'groups' )
tmp <- gjamFillMissingTimes(xdata, ydata, edata, groupCol, timeCol,
                            FILLMEANS = T, groupVars = groupVars,
                            typeNames = 'DA', missingEffort = .1)
xdata  <- tmp$xdata
ydata  <- tmp$ydata
edata  <- tmp$edata
tlist  <- tmp$timeList
snames <- colnames(ydata)

effort <- list(columns = 1:S, values = edata)
```

`tlist` is a `list` that holds bookkeeping objects used for vectorized operations. I would now replace the missing observations in `xdata` with known values for those groups and years, or I could leave them as `NA` in which case they will be imputed as in gjam.

The `list effort` is explained in `help(gjam)`.

##Prior distributions for coefficient matrices

Here is code to set up priors. `rhoPrior` is a list indicating `lo` and `hi` values for the growth rate, which is change per time increment. Again, growth rate `rho` only includes an `intercept`. The density-independent growth rate is given a wide prior values of $\pm 30\%$ per time increment:

```
rhoPrior  <- list(lo = list(intercept = -.3),
                  hi = list(intercept = .3) )
print(rhoPrior)
```

```
## $lo
## $lo$intercept
## [1] -0.3
##
##
## $hi
## $hi$intercept
## [1] 0.3
```

The prior parameter values are organized in a `list` by the `function gjamTimePrior`. It needs a `priorList` which includes `formulaRho` and `rhoPrior`. In this simulator `rhoPrior` is restricted to an intercept, which corresponds to the 'formula ~ ":

```
priorList <- list( formulaRho = as.formula(~ 1), rhoPrior = rhoPrior)
tmp <- gjamTimePrior( xdata, ydata, edata, priorList)
timeList <- mergeList(tlist, tmp)
```

In the last line I appended the `list tmp` generated by `gjamTimePrior` to my `timeList`.

Here are model fitting and plots using gjam:

```
modelList <- list(typeNames = 'DA', ng = 10000, burnin = 5000,
                  timeList = timeList, effort = effort)
outputAR <- gjam(formula, xdata=xdata, ydata=ydata, modelList=modelList)
```

Here are plots, which I've opted to deposit as `.pdf` files (`SAVEPLOTS = T`) in a folder `outFolder = gjamOutputAR`:

```
outFolder <- 'gjamOutputAR'
plotPars  <- list(PLOTALLY=T, trueValues = trueValues,
                  SAVEPLOTS = T, outFolder = outFolder)
gjamPlot(outputAR, plotPars)
save(outputAR, file = paste( outFolder, '/output.rdata', sep=''))
```

Here are MCMC chains:

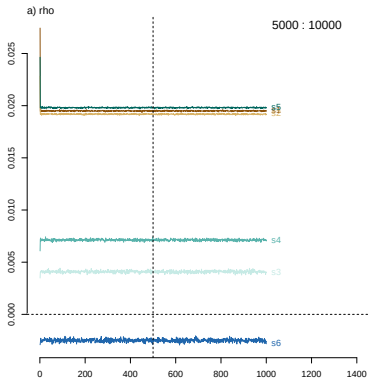


Figure 1: MCMC chains for rho in the AR model converge quickly.

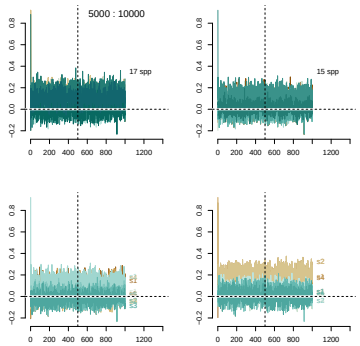


Figure 2: MCMC chains for correlation matrix.

The growth rates are recovered from the fitted model:

The fitted model predicts the data:

##AR with movement responding to environment

This example couples density-independent growth (`termR`) with environmental variation in movement (`termB`). The `logical` variable `termA` is again `FALSE`, because this model still does not include species interactions:

```
S      <- 6         # no. species
nsite <- 10         # no. time series
ntime <- 50         # mean no. of time steps in a series
obsEffort <- 1      # full census
```

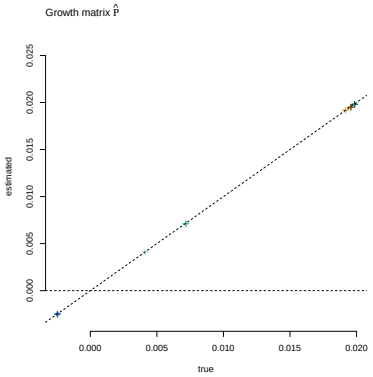


Figure 3: Parameter recovery for the AR model.

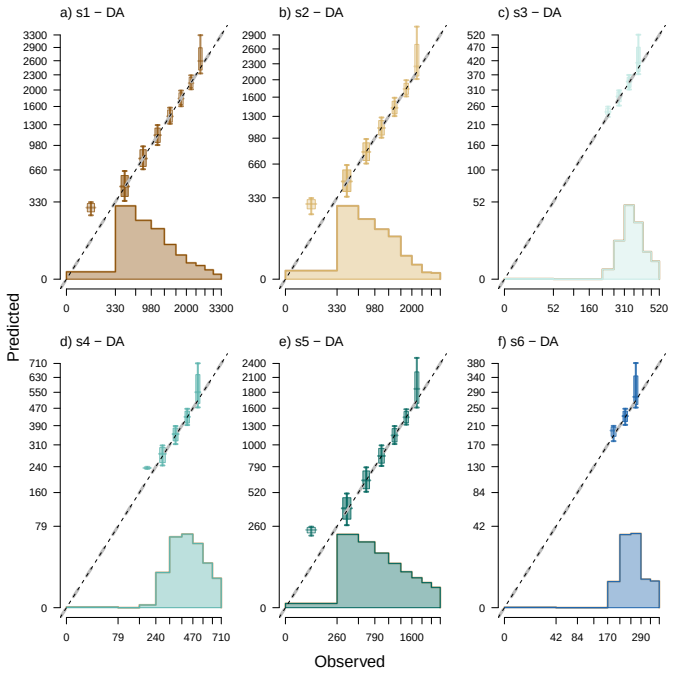
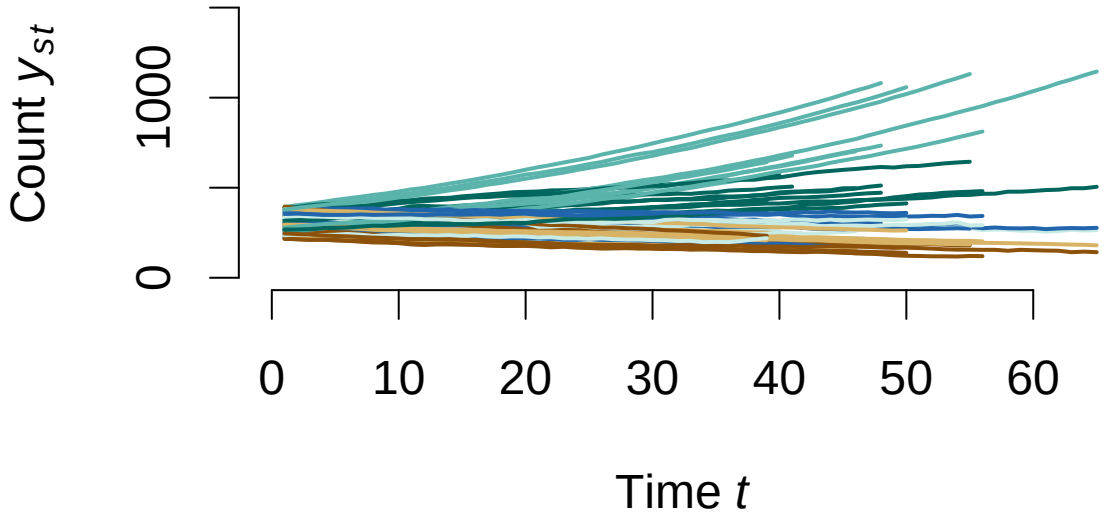


Figure 4: Data prediction for the AR model.

```
termB <- TRUE     # include immigration/emigration term XB
termR <- TRUE     # include DI population growth term VL
termA <- FALSE    # include DD spp interaction term UA
```

Simulated data include `Q = 3` predictors that will be named `"intercept", "x2", "x3"`.

```
seed <- 999
set.seed( seed )
tmp <- gjamSimTime(S, Q = 3, nsite, ntime, termB, termR, termA,
                   obsEffort = obsEffort, PLOT = T)
```




```
xdata  <- tmp$xdata
ydata  <- tmp$ydata
edata  <- tmp$edata
groups <- tmp$groups
times  <- tmp$times
trueValues <- tmp$trueValues
formula    <- tmp$formula
```

Here I define groups and times, and I fill missing values:

```
timeCol   <- 'times'
groupCol  <- 'groups'
groupVars <- c( 'groups' )
tmp <- gjamFillMissingTimes(xdata, ydata, edata, groupCol, timeCol,
                            FILLMEANS = T, groupVars = groupVars,
                            typeNames = 'DA', missingEffort = .1)
xdata  <- tmp$xdata
ydata  <- tmp$ydata
edata  <- tmp$edata
tlist  <- tmp$timeList
snames <- colnames(ydata)
effort <- list(columns = 1:S, values = edata)
```

Here are prior parameter values. The `list betaPrior` holds `lo` and `hi` values for `intercept` and `x3` (uniformative). The variable `x2` is unspecified, so it will be assigned bounds the exclude extreme values.

```
betaPrior <- list(lo = list(intercept = -Inf, x3 = -Inf),
                  hi = list(intercept = Inf, x3 = Inf) )
rhoPrior  <- list(lo = list(intercept = -.3), hi = list(intercept = .3) )
```

```
priorList <- list( formulaBeta = formula, formulaRho = as.formula(~ 1),
                   betaPrior = betaPrior, rhoPrior = rhoPrior)
tmp <- gjamTimePrior( xdata, ydata, edata, priorList)

timeList <- mergeList(tlist, tmp)
```

Here is model fitting:

```
modelList <- list(typeNames = 'DA', ng = 10000, burnin = 5000,
                  timeList = timeList, effort = effort)
outputARX <- gjam(formula, xdata=xdata, ydata=ydata, modelList=modelList)
```

Here are some plots that will be saved in a folder:

```
plotPars <- list(PLOTALLY=T, trueValues = trueValues, SAVEPLOTS = T,
                 outFolder = 'gjamOutputARX')
gjamPlot(outputARX, plotPars)
save(outputARX, file = 'gjamOutputARX/output.rdata')
```

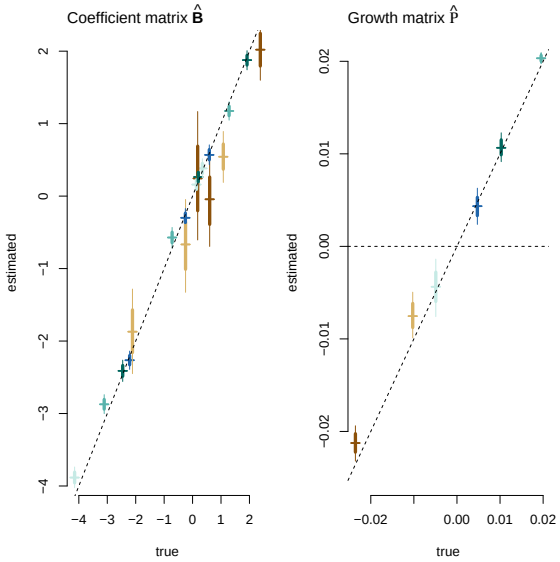


Figure 5: Parameter recovery for the AR model with movement.

#A small food web

A dynamic community is simulated by specifying species pairs that are predator-prey (`predPrey`), that do not affect one another (`zeroAlpha`), or that compete (the default). The function `foodWebDiagram` draws the food web:

```
S     <- 6     # no. species
nsite <- 10    # no. time series
ntime <- 100
obsEffort <- 1 # full census

termB <- FALSE    # include immigration/emigration term XB
termR <- TRUE     # include DI population growth term VL
termA <- TRUE     # include DD spp interaction term UA

predPrey  <- rbind( c(1, 3), c(1, 4), c(2, 3), c(2, 4) )   # second is prey of first
```

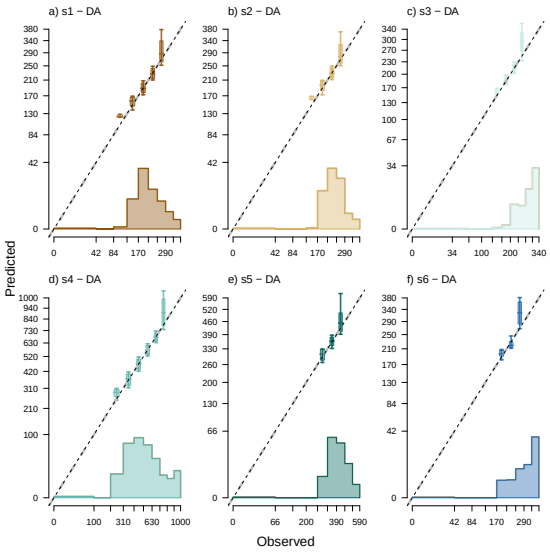


Figure 6: Data prediction.

```
zeroAlpha <- rbind( c(1, 2), c(2, 1),                    # second does not affect first
                    c(1, 5), c(5, 1),
                    c(2, 5), c(5, 2),
                    c(1, 6), c(6, 1),
                    c(2, 6), c(6, 2))

foodWebDiagram(S, predPrey = predPrey, zeroAlpha = zeroAlpha)
```

```
## Loading required package: DiagrammeR
```

In this diagram, red arrows are negative (predation), blue are positive (prey effect on predator), and brown arrows are (negative) competition. Here is a simulation:

```
seed <- 999
set.seed( seed )
tmp <- gjamSimTime(S, Q = 0, nsite, ntime, termB, termR, termA,
                   obsEffort = obsEffort, predPrey, zeroAlpha, PLOT = T)
xdata  <- tmp$xdata
ydata  <- tmp$ydata
edata  <- tmp$edata
groups <- tmp$groups
times  <- tmp$times
trueA  <- tmp$trueValues
formula <- tmp$formula

print( tmp$wdata )   # show eigenvalues and equilibrium abundances
```

This stable community has eigenvalues for `alpha` with all negative parts and equilibrium abundances that are positive. Here I fill missing values:

```
timeCol   <- 'times'
groupCol  <- 'groups'
groupVars <- c( 'groups' )
tmp <- gjamFillMissingTimes(xdata, ydata, edata, groupCol, timeCol,
```

```
                              FILLMEANS = T, groupVars = groupVars,
                              typeNames = 'DA', missingEffort = .1)
xdata  <- tmp$xdata
ydata  <- tmp$ydata
edata  <- tmp$edata
tlist  <- tmp$timeList
snames <- colnames(ydata)
effort <- list(columns = 1:S, values = edata)
```

I set wide prior on DI population growth rate (up to 30% increase per year) and interaction coefficients. In `alphaSign` I specify the sign of interactions in for the interaction matrix α as (-1, 0, 1) to indicate negative, no interaction, and positive, respectively.

```
rhoPrior <- list(lo = list(intercept = 0), hi = list(intercept = .3) )

alphaSign  <- matrix(-1, S, S)
colnames(alphaSign) <- rownames(alphaSign) <- colnames(ydata)
alphaSign[ predPrey ]  <- 1
alphaSign[ zeroAlpha ] <- 0

priorList <- list( formulaRho = as.formula(~ 1),
                   rhoPrior = rhoPrior, alphaSign = alphaSign)

tmp <- gjamTimePrior( xdata, ydata, edata, priorList )

timeList <- mergeList(tlist, tmp)
```

Here is model fitting:

```
modelList <- list(typeNames = 'DA', ng = 10000, burnin = 5000,
           timeList = timeList, effort = effort)

outputA <- gjam(formula, xdata=xdata, ydata=ydata, modelList=modelList)
```

Here are plots:

```
plotPars <- list(PLOTALLY=T, trueValues = trueA, SAVEPLOTS = T,
                 outFolder = 'gjamOutputA')
gjamPlot(outputA, plotPars)
save(outputA, file = 'gjamOutputA/output.rdata')
```

Here are plots:

Mean estimates for the interaction matrix α are available in `outputA$parameters$alphaMu`. Standard errors are in `outputA$parameters$alphaSe`. Eigenvalues are in `outputA$parameters$alphaEigen`.

#A food web with movement in a noisy environment

Building on the same model, I now introduce environmental effects through movement. Here is a simulation for `Q = 3` predictors on movement, DI growth, and DD:

```
Q     <- 3     # no. predictors
termB <- T    # include immigration/emigration term XB
termR <- T    # include DI population growth term VL
termA <- T    # include DD spp interaction term UA

set.seed( seed )
tmp <- gjamSimTime(S, Q = Q, nsite, ntime, termB, termR, termA,
```

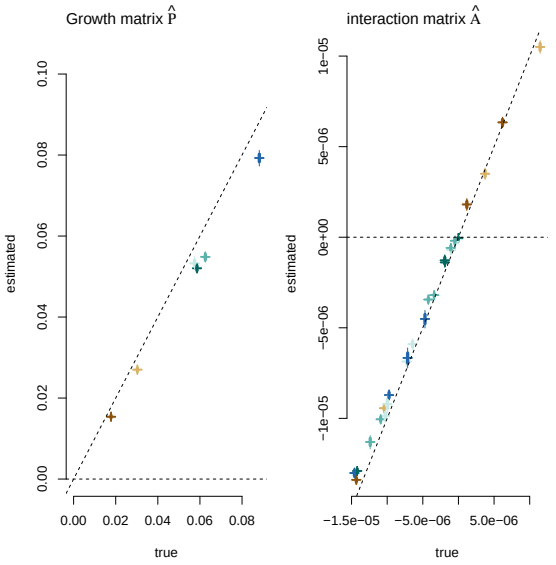


Figure 7: Parameter recovery for the AR model.

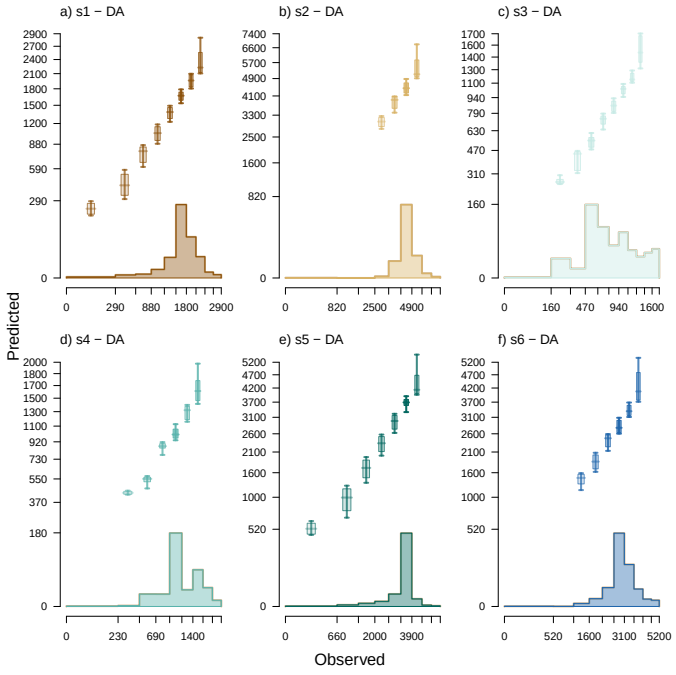


Figure 8: Data prediction for the AR model.

```
                  obsEffort = obsEffort, predPrey,
                  zeroAlpha, PLOT = T)
xdata  <- tmp$xdata
ydata  <- tmp$ydata
edata  <- tmp$edata
groups <- tmp$groups
times  <- tmp$times
trueB  <- tmp$trueValues
formula <- tmp$formula

legend('topright', snames, text.col = colF(S), ncol = 2, bty='n')

timeCol   <- 'times'
groupCol  <- 'groups'
groupVars <- c( 'groups' )
tmp <- gjamFillMissingTimes(xdata, ydata, edata, groupCol, timeCol,
                            FILLMEANS = T, groupVars = groupVars,
                            typeNames = 'DA', missingEffort = .1)
xdata <- tmp$xdata
ydata <- tmp$ydata
edata <- tmp$edata
tlist <- tmp$timeList

effort <- list(columns = 1:S, values = edata)
```

The prior parameter distribution includes some different, but still wide bounds on parameters for `beta`. I have not changed `alphaSign`.

```
betaPrior <- list(lo = list(intercept = -Inf, x2 = -Inf, x3 = -Inf),
                  hi = list(intercept = Inf, x2 = Inf, x3 = Inf) )

rhoPrior <- list(lo = list(intercept = 0), hi = list(intercept = .3) )
formulaRho <- as.formula(~ 1)

if(!termR) rhoPrior <- NULL

priorList <- list( formulaBeta = formula, formulaRho = formulaRho,
                   betaPrior = betaPrior, rhoPrior = rhoPrior, alphaSign = alphaSign)

tmp <- gjamTimePrior( xdata, ydata, edata, priorList)

timeList <- mergeList(tlist, tmp)
```

Here is model fitting:

```
modelList <- list(typeNames = 'DA', ng = 10000, burnin = 5000,
                  timeList = timeList, effort = effort)
outputXRA   <- gjam(formula, xdata=xdata, ydata=ydata, modelList=modelList)
```

Here are plots:

```
plotPars <- list(PLOTALLY=T, trueValues = trueB, SAVEPLOTS = T,
                 outFolder = 'gjamOutputXRA')
gjamPlot(outputXRA, plotPars)
save(outputXRA, file='gjamOutputXRA/output.rdata')
```

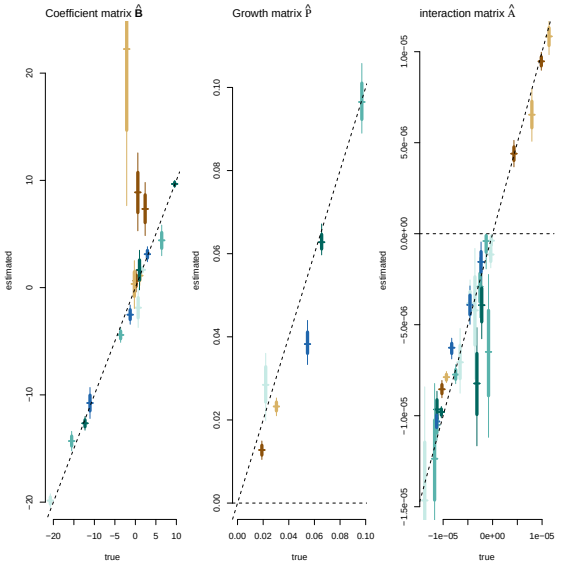


Figure 9: Parameter recovery for the model is not terrible, because the effect of movement in this example is small.

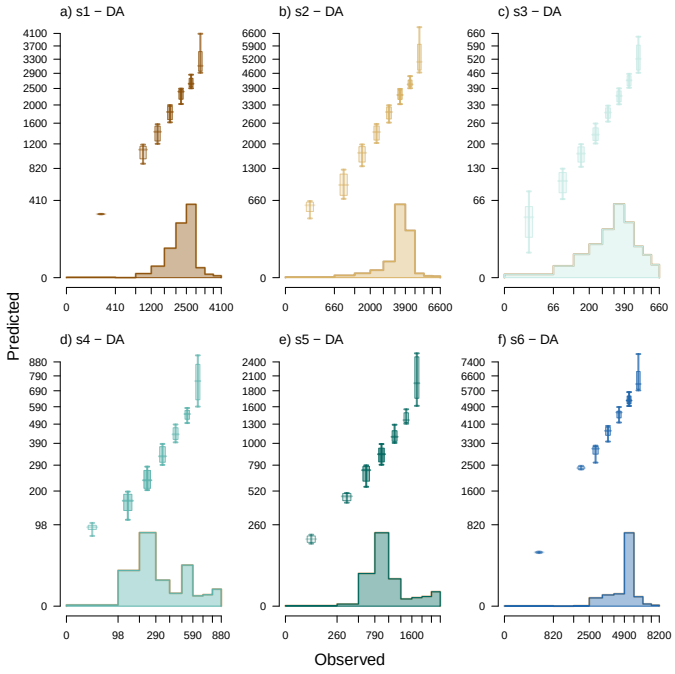


Figure 10: Data prediction is still good due to model size.

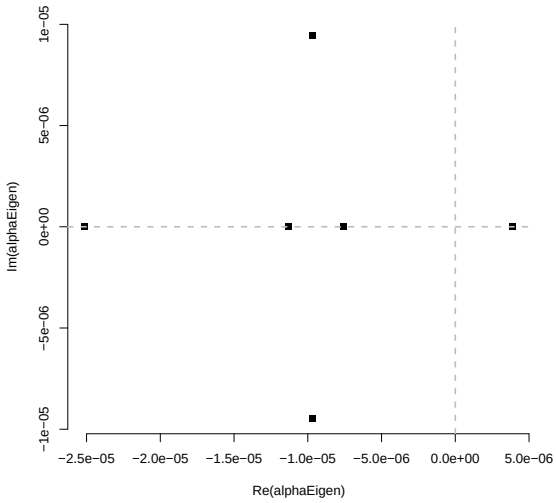


Figure 11: Eigenvalues of alpha all have negative real parts.

##Unknown environmental effects on movement

What are the effects of fitting a model where knowledge of the environment and food web are incomplete? Here I fit the model to the previous data, but without knowledge of movement. First I specify a prior distribution that omits `betaPrior`, because I don't know that it is operating here. I then fit the model:

```
priorList <- list( formulaRho = formulaRho, rhoPrior = rhoPrior, alphaSign = alphaSign)

tmp <- gjamTimePrior( xdata, ydata, edata, priorList )

timeB1 <- append( tlist, tmp )

modelList <- list(typeNames = 'DA', ng = 10000, burnin = 5000,
                  timeList = timeB1, effort = effort)
outputB1 <- gjam(formula = as.formula(~ 1), xdata=xdata, ydata=ydata,
                  modelList=modelList)
```

In the plots that follow there is no `beta` in the true values, because I have not fitted movement:

```
trueB1 <- trueB[ !names(trueB) == 'beta' ]
plotPars <- list(PLOTALLY=T, trueValues = trueB1, SAVEPLOTS = T, outFolder = 'gjamOutputB1')
gjamPlot(outputB1, plotPars)
save(outputB1, file='gjamOutputB1/output.rdata')
```

##A subset of the community

Four species are fitted to the six-species community. In other words, two species are unobserved. Here I am using the simulated community fitted previously, but fitting the model to only four species. This means that the effects of two competitors (S5, S6) of (S3, S4) are omitted from the model. I have included movement effects through beta in this example.

```
y4 <- ydata[,1:4]
e4 <- list(columns = 1:4, values = edata[,1:4])
betaPrior <- list(lo = list(intercept = -2, x2 = -100, x3 = -100),
                  hi = list(intercept = 2, x2 = 100, x3 = 100) )
```

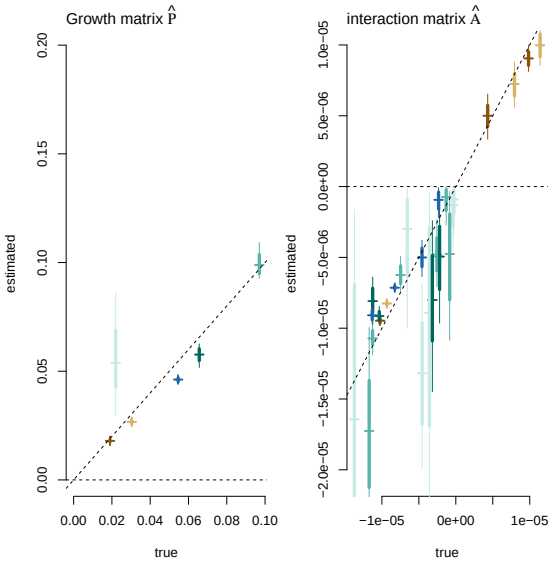


Figure 12: Parameter recovery for the model.

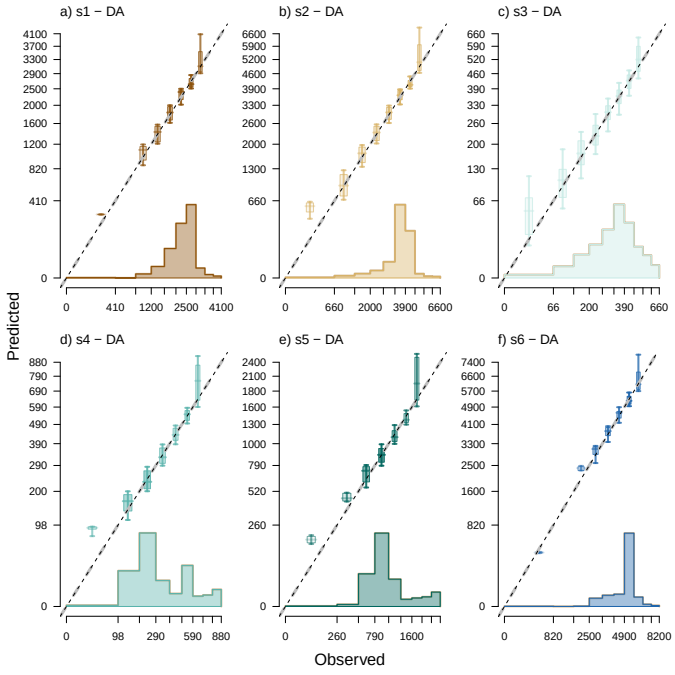


Figure 13: Data prediction.

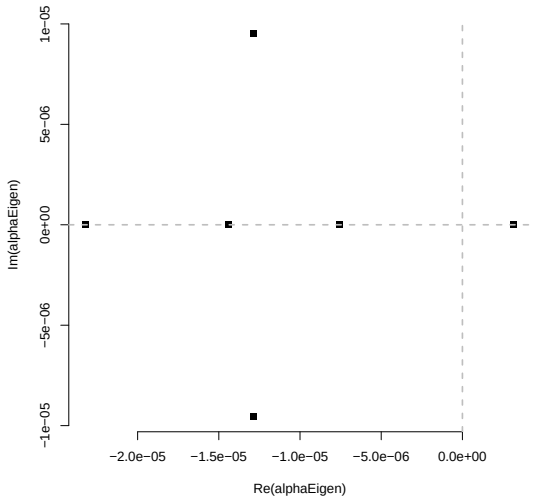


Figure 14: Eigenvalues of alpha all have negative real parts.

```
priorList <- list( formulaBeta = formula, formulaRho = formulaRho,
                   betaPrior = betaPrior, rhoPrior = rhoPrior, alphaSign = alphaSign[1:4,1:4])

tmp <- gjamTimePrior( xdata, ydata = y4, edata = edata[,1:4], priorList)

timeC <- append( tlist, tmp )

modelList <- list(typeNames = 'DA', ng = 10000, burnin = 5000,
           timeList = timeC, effort = e4)

outputC <- gjam(formula, xdata = xdata, ydata = y4, modelList = modelList)

trueC <- list(beta = trueB$beta[,1:4], rho = trueB$rho[1:4,1:4],
           alpha = trueB$alpha[1:4,1:4], sigma = trueB$sigma[1:4,1:4],
           w = trueB$w[,1:4])

plotPars <- list(PLOTALLY=T, trueValues = trueC, SAVEPLOTS = T, outFolder = 'gjamOutputC' )
gjamPlot(outputC, plotPars)
save(outputC, trueC, e4, y4, file = 'gjamOutputC/output.rdata')
```

Effects here are modest, so the deterioration in the fit is not large:

The model still has enough parameters to predict the data:

##Small sample size

Thus far, the sample has been in the thousands of counts. In many data sets only a few individuals are observed. This situation is simulated by using low `obsEffort`, i.e., only a small part of the population is observed.

```
obsEffort <- .01 # a small fraction observed

termB <- T    # include immigration/emigration term XB
termR <- T    # include DI population growth term VL
```

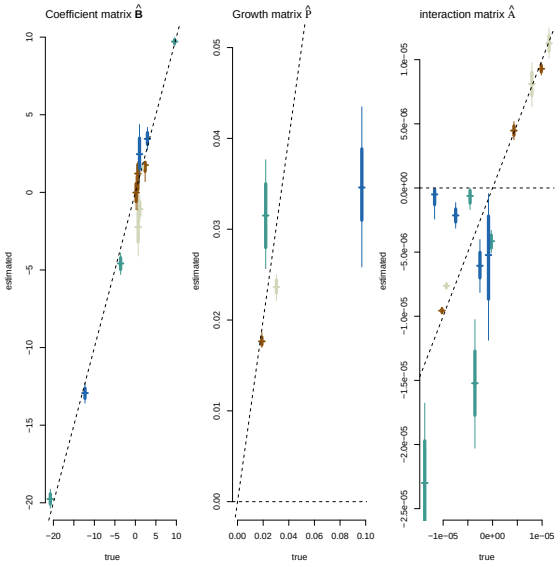


Figure 15: Parameter recovery for the model where two species are omitted.

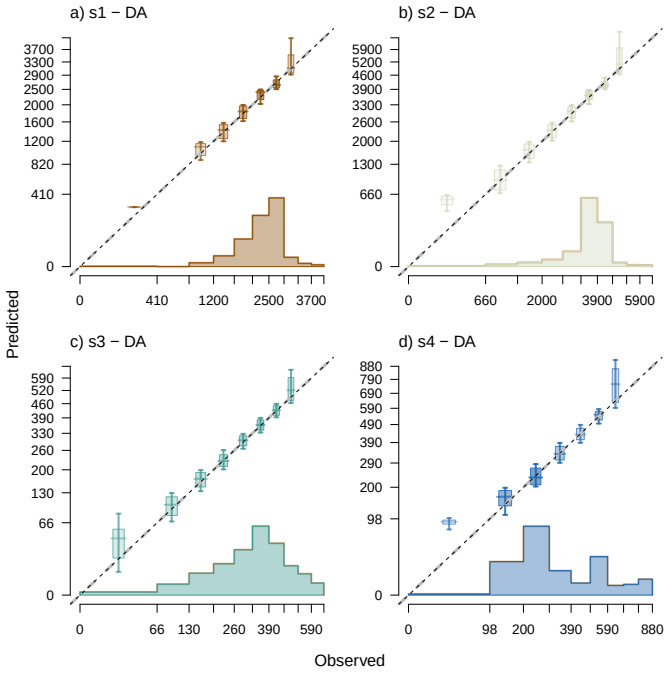


Figure 16: Data prediction.

```
termA <- T    # include DD spp interaction term UA

set.seed( seed )
tmp <- gjamSimTime(S, Q = 3, nsite, ntime, termB, termR, termA, obsEffort = obsEffort,
                   predPrey, zeroAlpha, PLOT = T)
xdata  <- tmp$xdata
ydata  <- tmp$ydata
edata  <- tmp$edata
groups <- tmp$groups
times  <- tmp$times
#wstar  <- tmp$wstar
trueE <- tmp$trueValues
formula <- tmp$formula

tmp <- gjamFillMissingTimes(xdata, ydata, edata, groupCol, timeCol,
                            FILLMEANS = T, groupVars = groupVars,
                            typeNames = 'DA', missingEffort = .1)
xdata <- tmp$xdata
ydata <- tmp$ydata
edata <- tmp$edata
tlist <- tmp$timeList

effort <- list(columns = 1:S, values = edata)
```

Here is the prior parameter distribution specified previously:

```
priorList <- list( formulaBeta = formula, formulaRho = formulaRho,
                   betaPrior = betaPrior, rhoPrior = rhoPrior, alphaSign = alphaSign)

tmp <- gjamTimePrior( xdata, ydata, edata, priorList)
timeList <- mergeList(tlist, tmp)
```

Here are model fitting and plots:

```
modelList <- list(typeNames = 'DA', ng = 10000, burnin = 5000,
           timeList = timeList, effort = effort)

outputE <- gjam(formula, xdata=xdata, ydata=ydata, modelList=modelList)
```

Here are plots:

```
plotPars <- list(PLOTALLY=T, trueValues = trueE, SAVEPLOTS = T, outFolder = 'gjamOutputE' )
gjamPlot(outputE, plotPars)
save(outputE, trueE, file='gjamOutputE/output.rdata')
```

Effects on parameter recovery are large. Note that the size of the sample is the same as before, all I have done is reduce observation effort:

Data prediction is ok for abundant species, but not for rare species. There are too many zeros:

##Combined effects

What if we fit the model without allowing for the environmental effects on movement and we have knowledge of only some of the interacting species. In this example I omit the movement term, species S5 and S6, and I have a limited sample. In other words, I combine effects examined individually in previous examples.

```
priorList <- list( formulaRho = formulaRho, rhoPrior = rhoPrior, alphaSign = alphaSign[1:4,1:4])
```

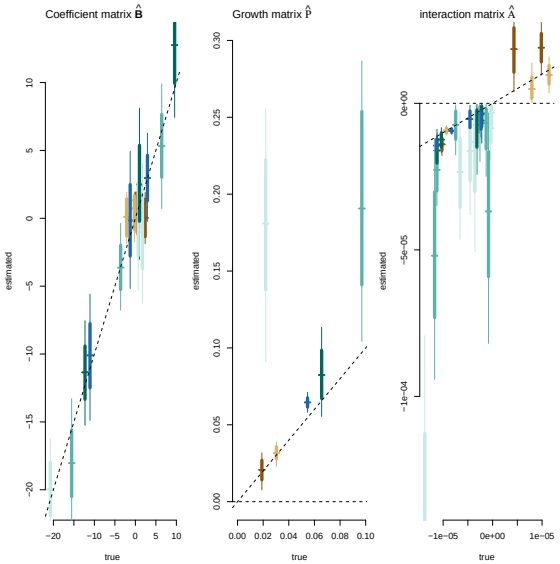


Figure 17: Parameter recovery deteriorated by small sample size.

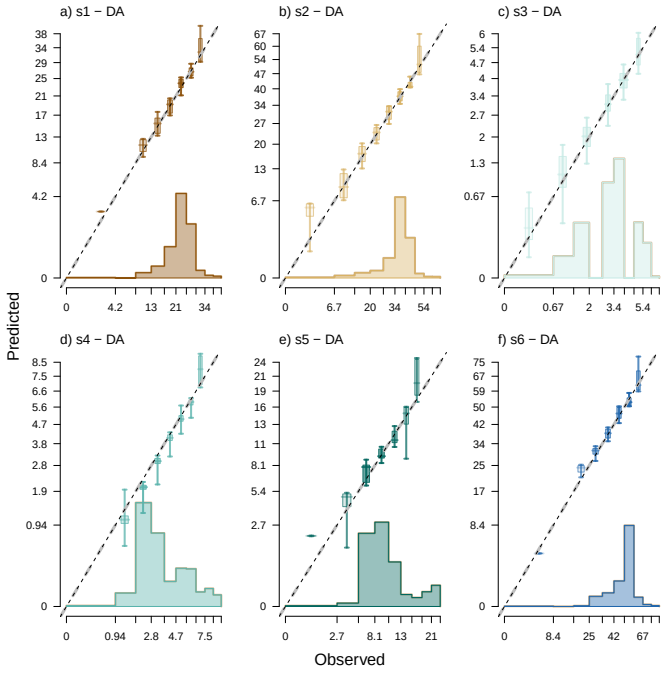


Figure 18: Data prediction is especially poor for rare species.

```
tmp <- gjamTimePrior( xdata, ydata[,1:4], edata[,1:4], priorList)

timeList <- mergeList(timeList, tmp)
```

Here are model fitting and plots

```
modelList <- list(typeNames = 'DA', ng = 10000, burnin = 5000,
           timeList = timeList, effort = e4)

outputF <- gjam(formulaRho, xdata=xdata, ydata=ydata[,1:4], modelList=modelList)
```

```
trueF <- trueC[!names(trueC) == 'beta']
plotPars <- list(PLOTALLY=T, trueValues = trueF, SAVEPLOTS = T, outFolder = 'gjamOutputF' )
gjamPlot(outputF, plotPars)
save(outputF, trueF, file='gjamOutputF/output.rdata')
```

Data prediction is still ok for abundant species, but not for rare species:

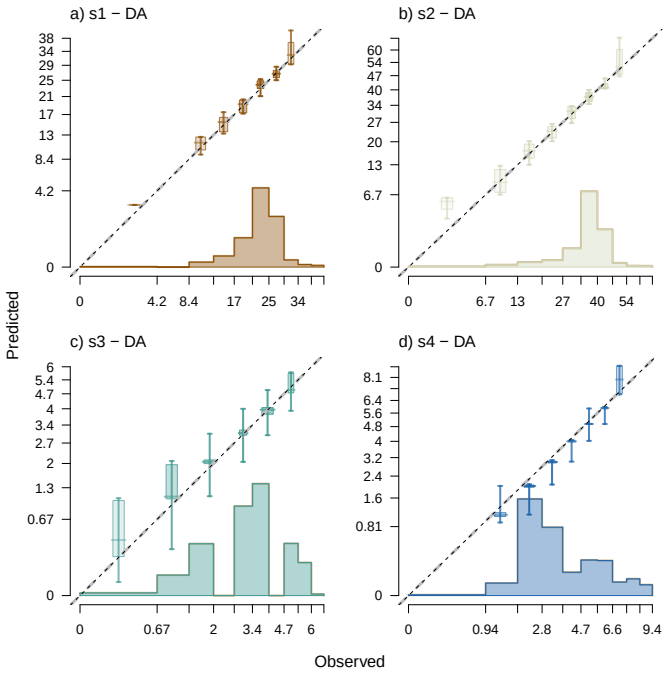


Figure 19: Data prediction is especially poor for rare species.

However, the parameter estimates give an inaccurate representation of the contributions to population growth.

##BBS data

Here is a sample of BBS data from North Carolina, which is smaller than the example in the main text to speed computation:

```
library(repmis)
library(maps)
d <- "https://github.com/jimclarkatduke/gjam/blob/master/BBSexampleTime.rdata?raw=True"
source_data(d)

colT <- colorRampPalette( c('#8c510a','#d8b365','#c7eae5','#5ab4ac','#01665e','#2166ac') )
cols <- rev( colT(20) )
```

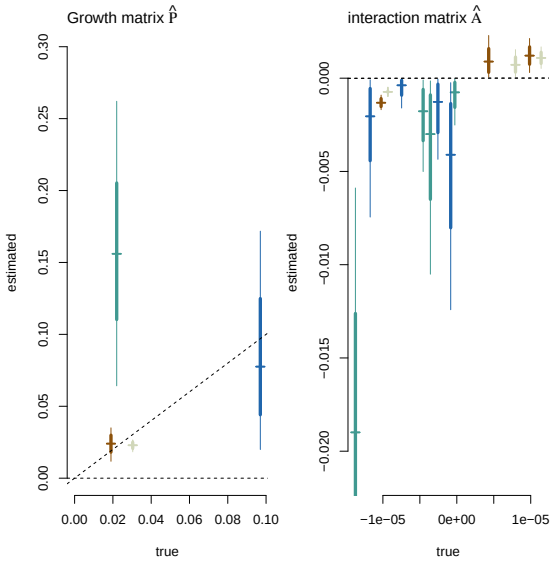


Figure 20: Parameter recovery deteriorated by the combined effects of missing movement, missing species, and small sample size.

```
spec <- 'WoodThrush'
y    <- ydata[,spec]

# route mean
cy   <- tapply(y, xdata$Route, mean, na.rm=T)
wi   <- match( as.numeric(names(cy)), xdata$Route )
ll   <- xdata[ wi, c('lon','lat') ]

# the moisture deficit varies by route
def <- xdata$defSite[wi]

# discretize color scheme
df   <- seq(min(def), max(def), length=20)

cols <- rev( colT(20) )

# assign a color to each anomaly
di <- findInterval(def, df, all.inside = T)

map('county', xlim = c(-85, -75), ylim = c(33.6, 36.8), col='grey')
map('state', xlim = c(-85, -75), ylim = c(33.6, 36.8), add=T)
cex <- 5*cy/max(cy, na.rm=T)
points(ll[,1], ll[,2], pch = 16, cex = cex, col = cols[di] )
title(spec)
```

Here is the specification of interactions, based on guilds of potential competitors:

```
xdata$StartWind <- as.factor(xdata$StartWind)
xdata$StartSky  <- as.factor(xdata$StartSky)
S <- ncol(ydata)
snames <- colnames(ydata)
```

```
guild1 <- c("AmericanRobin", "EuropeanStarling", "AmericanCrow", "CommonGrackle")
guild2 <- c("NorthernCardinal", "ChippingSparrow", "CommonGrackle", "AmericanCrow")
guild3 <- c("NorthernCardinal", "ChippingSparrow", "BlueJay", "BrownThrasher",
            "EasternTowhee", "WoodThrush", "MourningDove", "CarolinaWren")
guild4 <- c("EasternWood-Pewee", "Red-eyedVireo", "IndigoBunting",
            "TuftedTitmouse","Red-eyedVireo","CarolinaChickadee")

guildList <- list(guild1 = match(guild1, snames),
                  guild2 = match(guild2, snames),
                  guild3 = match(guild3, snames),
                  guild4 = match(guild4, snames)) # competition groups

foodWebDiagram(S, guildList, label = snames, intraComp = 1:S)

fromTo <- foodWebDiagram(S, guildList, PLOT = F)

formulaBeta <- as.formula(~ StartWind)
formulaRho <- as.formula(~ defSite + juneTemp + nlcd)

hi <- list(intercept = Inf, StartWind2 = 0, StartWind3 = 0, StartWind4 = 0)
lo <- list(intercept = -Inf, StartWind0 = 0, StartWind2 = -1, StartWind3 = -2, StartWind4 = -3)
betaPrior <- list(lo = lo, hi = hi)

lo <- list(intercept = -.05)                                  # winter temp
hi <- list(intercept = .1)
rhoPrior <- list(lo = lo, hi = hi)

alphaSign  <- matrix(0, S, S)
colnames(alphaSign) <- rownames(alphaSign) <- snames
alphaSign[ fromTo ] <- -1

priorList <- list( formulaBeta = formulaBeta, formulaRho = formulaRho,
                   betaPrior = betaPrior,
                   rhoPrior = rhoPrior, alphaSign = alphaSign)

tmp <- gjamTimePrior( xdata, ydata, edata, priorList)

timeList <- mergeList(timeList, tmp)
```

Here are model fitting and plots:

```
effort <- list(columns = 1:S, values = edata)
modelList <- list( typeNames = 'DA', ng = 2000, burnin=500,
                   timeList=timeList, effort = effort)

outputBBS <- gjam(formulaBeta, xdata=xdata, ydata=ydata, modelList=modelList)

specColor <- rep( '#000000' ,S) # black
names(specColor) <- snames
sc <- colF( length(guildList) )
for(j in 1:length(guildList)){
  specColor[ snames[ guildList[[j]] ] ] <- sc[j]
}
```

```
plotPars <- list(specColor = specColor, GRIDPLOTS=T, PLOTALLY=T, SAVEPLOTS = T,
                 outFolder = 'gjamOutputBBS')
gjamPlot(outputBBS, plotPars)
```

Equilibrium abundances can be evaluated here:

```
wstar <- .wrapperEquilAbund(outputBBS, covars = 'juneTemp',
                            nsim = 100, ngrid = 9, BYFACTOR = T,
                            verbose = T)

save(outputBBS, specColor, wstar, file='gjamOutputBBS/outputBBS.rdata')

notOther <- outputBBS$inputs$notOther
ccMu <- wstar$ccMu[,notOther]
ccSd <- wstar$ccSd[,notOther]
ccx  <- wstar$x

cols <- colorRampPalette( c('#a6611a','#dfc27d','#80cdc1','#018571') )

nlcd <- colnames(ccx)[ which(startsWith(colnames(ccx), 'nlcd')) ]
f1   <- rownames(outputBBS$inputs$factorRho$contrast$nlcd)[1]

coverType <- cbind(.5, ccx[,nlcd])
colnames(coverType)[1] <- f1
nlcd <- c(f1, nlcd)

coverType <- nlcd[ apply(coverType, 1, which.max) ]
coverType <- .replaceString(coverType, 'nlcd','')


SAVEPLOTS <- F
outFolder <- 'gjamOutputBBS'

types <- c( "dev",  "crop", "grassland","shrub",
           "forest", "wetland")

np <- ncol(ccMu)

npage <- 1
o   <- 1:np
if(np > 16){
  npage <- ceiling(np/16)
  np    <- 16
}

SO <- length(notOther)

mfrow <- .getPlotLayout(np)

k   <- 0
add <- F
o   <- 1:np
o   <- o[o <= 16]
```

```
xm <- 'juneTemp'

for(p in 1:npage){

  file <- paste('equilAbund_', xm, '_', p,'.pdf',sep='')

  if(SAVEPLOTS)pdf( file=.outFile(outFolder,file) )

  npp <- ncol(ccMu) - k
  if(npp > np)npp <- np
  mfrow <- .getPlotLayout(np)
  par(mfrow=mfrow$mfrow, bty='n', omi=c(.5,.5,0,0), mar=c(1,2,2,1))
  pj <- 1

  for(j in o){

    yy  <- ccMu[,j]

    ct  <- tapply(yy, list(defSite = round(ccx[,'juneTemp'], 1), nlcd = coverType), mean)
    cl  <- tapply(yy, list(defSite = round(ccx[,'juneTemp'], 1), nlcd = coverType),
                  quantile, pnorm(-1))
    ch  <- tapply(yy, list(defSite = round(ccx[,'juneTemp'], 1), nlcd = coverType),
                  quantile, pnorm(1))

    def <- as.numeric(rownames(ct))
    nk <- length(def)

    cdd  <- rev( cols(nk) )

    ylimit <- c(0, 1.5*max(ct))

    ct <- ct[,types]
    cl <- cl[,types]
    ch <- ch[,types]

    tbar <- barplot( ct, beside = T, plot = F)

    plot(NA, xlim = range(tbar), ylim = range( cbind(cl, ch) ),
         xaxt = 'n', xlab = '', ylab = '')
    axis(1, at = c(tbar[1,], tbar[nrow(tbar),]), labels = F)
    for(k in 1:ncol(ct)){
      .shadeInterval(tbar[,k], loHi = cbind(cl[,k],ch[,k]) )

      cc <- 1:(nk-1)
      segments(tbar[cc,k], ct[cc,k], tbar[cc+1,k], ct[cc+1,k], col = cdd[-1], lwd=2)
      segments(tbar[1:nk,k], cl[1:nk,k], tbar[1:nk,k], ch[1:nk,k], col = cdd, lwd=1)
      segments(tbar[cc,k], cl[cc,k], tbar[cc+1,k], cl[cc+1,k], col = cdd[-1], lwd=.5)
      segments(tbar[cc,k], ch[cc,k], tbar[cc+1,k], ch[cc+1,k], col = cdd, lwd=.5)
    }

    xbar <- colMeans(tbar)
    ybar <- apply(ct, 2, max)
    if(pj == 1)text(xbar, ybar, colnames(ct), srt = 80, pos = 4)
```

```
    k <- k + 1
    if(k > 26)k <- 1
    pj <- pj + 1

    lab <- colnames(ccMu)[j]

    .plotLabel( lab,above=T )
  }
  mtext('Moisture deficit by land cover', 1, outer=T)
  mtext('Equilibrium abundance', 2, outer=T)

  o <- o + 16
  o <- o[o <= SO]

  if(!SAVEPLOTS){
    readline('equilibrium abundance -- return to continue ')
  } else {
    dev.off()
  }
}
```

#Wisconsin lakes

"Potential planktivory was calculated as the biomass of fishes that could potentially consume zooplankton, based on species and body size. Only two fish samples were taken per year, at the beginning and end of summer stratification. We log-linearly interpolated between these points to obtain estimates of potential planktivory at the times of plankton samples."

```
d <- "https://github.com/jimclarkatduke/gjam/blob/master/lakeExampleTime.rdata?raw=True"
source_data(d)

xdata$lake <- as.factor(xdata$lake)
xdata$Pvory[ xdata$Pvory < 1 ] <- 0
xdata$Pvory <- as.factor(xdata$Pvory)
colnames(xdata)[colnames(xdata) == 'logBiomass'] <- 'bass'

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

colnames(ydata) <- colnames(edata) <- c('sPhy','lPhy','lZoo','sZoo')
ydata <- ydata[,c('sPhy','lPhy','sZoo','lZoo')]
edata <- edata[,c('sPhy','lPhy','sZoo','lZoo')]

snames <- colnames(ydata)
S <- length(snames)

specColor <- colF( S )
names(specColor) <- snames


predPrey  <- rbind( c(3, 1), c(4, 1), c(4, 2) )   # second is prey of first
zeroAlpha <- rbind( c(2, 3),
                    c(4, 3))
zeroAlpha <- rbind(zeroAlpha, zeroAlpha[,c(2,1)])
```

```
preyAll <-  rbind( predPrey,
                   c( 1, 5), c( 2, 5),
                   c( 7, 3), c(7, 4),
                   c( 6, 7))
zeroAll <- rbind( zeroAlpha,
                  c(5, 3), c(5, 4), c(5, 5), c(5, 6),
                  c(3, 5), c(4, 5), c(3, 6), c(3, 7), c(4, 6),
                  c(4, 7),
                  c(6, 1), c(6, 2), c(6, 5), c(6, 6),
                  c(1, 6), c(2, 6),
                  c(6, 3), c(6, 4),
                  c(1, 5), c(2, 5), c(5, 7),
                  c(7, 1), c(7, 2), c(7, 5), c(7, 6),
                  c(1, 7), c(2, 7), c(7, 7))

label <- c( colnames(ydata), 'Pvol','bass','perch')

foodWebDiagram(S = length(label), predPrey = preyAll, zeroAlpha = zeroAll, label = label,
               layout = 'rr')
```

Here is the prior distribution, setup, and model fitting:

```
formulaRho  <- as.formula( ~ Pvol + bass*temp )

rhoPrior  <- list(lo = list(intercept = 0, Pvol = 0),
                  hi = list(intercept = .5))

alphaSign  <- matrix(-1, S, S)
colnames(alphaSign) <- rownames(alphaSign) <- colnames(ydata)
alphaSign[ predPrey ] <- 1
alphaSign[ zeroAlpha ] <- 0
colnames(alphaSign) <- rownames(alphaSign) <- snames

priorList <- list( formulaRho = formulaRho,
                   rhoPrior = rhoPrior, alphaSign = alphaSign)

tmp <- gjamTimePrior( xdata, ydata, edata, priorList )

# planktivores only consume daphnia, non-daphnia, others get NA

brows <- grep('bass', rownames(tmp$rhoPrior$lo))
tmp$rhoPrior$lo['bass','sZoo'] <- 0  # neg effect of small fish, pos effect on daphnia
tmp$rhoPrior$lo['bass','lZoo'] <- 0
tmp$rhoPrior$lo[brows,'sPhy']  <- tmp$rhoPrior$hi[brows,'sPhy'] <- NA
tmp$rhoPrior$lo[brows,'lPhy']  <- tmp$rhoPrior$hi[brows,'lPhy'] <- NA

brows <- grep('Pvol', rownames(tmp$rhoPrior$lo))
tmp$rhoPrior$lo[brows,'sZoo']  <- tmp$rhoPrior$hi[brows,'sZoo'] <- NA
tmp$rhoPrior$lo[brows,'lZoo']  <- tmp$rhoPrior$hi[brows,'lZoo'] <- NA

timeList <- mergeList(timeList, tmp)
effort   <- list(columns = 1:S, values = edata)

modelList <- list( typeNames = 'DA', ng = 20000, burnin= 5000,
```

```
           timeList=timeList, effort = effort)

output1 <- gjam(formulaRho, xdata=xdata, ydata=ydata, modelList=modelList)

save(output1, file = 'gjamOutputLakes1/gjamOutputLakes1.rdata')

plotPars <- list(specColor = specColor, PLOTALLY=T, SAVEPLOTS = T, outFolder = 'gjamOutputLakes1')
gjamPlot(output1,plotPars)
```

Equilibrium abundances are here:

```
ngrid <- 11
wstar <- .wrapperEquilAbund(output1, nsim = 2000, covars = 'bass', 'temp',
                            ngrid = ngrid, BYFACTOR = T,
                            BYGROUP = F, verbose = T)

save(output1, wstar, file = 'gjamOutputLakes1/gjamOutputLakes1.rdata')

cols <- rev( colT(ngrid) )

notOther <- output1$inputs$notOther
ccMu <- wstar$ccMu[,notOther]
ccSd <- wstar$ccSd[,notOther]
ccx  <- wstar$x

ccx  <- ccx[, !colnames(ccx) %in% attributes(ccx)$factors, drop=F]
ccx  <- ccx[,-1, drop=F]

np <- ncol(ccMu)

npage <- 1
o   <- 1:np
if(np > 16){
  npage <- ceiling(np/16)
  np    <- 16
}

alpha <- output1$parameters$alphaMu
rho   <- output1$parameters$rhoStandXmu

intercept <- 1
bt <- ccx[,'bass']*ccx[,'temp']
rx <- cbind(intercept, ccx, bt)     # assumes formula for rho
rx <- rx%*%rho
ax <- -t( t(rx)/diag(alpha) )

mfrow <- .getPlotLayout(np)

for(m in 1:ncol(ccx)){   # loop over predictors

  xm  <- colnames(ccx)[m]
  xx  <- ccx[,m]
  k   <- 0
  add <- F
```

```
o   <- 1:np
o   <- o[o <= 16]

for(p in 1:npage){

  npp <- ncol(ccMu) - k
  if(npp > np)npp <- np
  mfrow <- .getPlotLayout(np)
  par(mfrow=mfrow$mfrow, bty='n', oma = c(2,2,1,1), mar=c(3,3,2,1))
  pj <- 1

  for(j in o){

    yy  <- ccMu[,j]
    cc  <- ax[,j]

    xmids <- sort( unique( ccx[,1] ) )

    c95 <- tapply( yy, list(bin = ccx[,xm]), quantile, pnorm( c(-1.96, -1, 1, 1.96) ))
    ci <- matrix( unlist( c95 ), ncol = 4, byrow = T )
    rownames(ci) <- names(c95)

    cmid <- tapply( yy, list(bin = ccx[,xm]), quantile, .5 )

    for(s in 1:4)ci[,s] <- smooth(ci[,s])

    cmu <- tapply( yy, list(bin = ccx[,xm]), mean)
    amu <- tapply( cc, list(bin = ccx[,xm]), mean)

    nk <- length(xmids)
    cdd  <- rev( colT(nk) )

    plot(xmids, cmid, xlim = range(xmids), ylim = range(ci), xlab = '',
         lwd = 2, col = 'grey', type = 'l')
    .shadeInterval(xmids, loHi = cbind(ci[,2],ci[,3]), add=T, trans = .2 )
    .shadeInterval(xmids, loHi = cbind(ci[,1],ci[,4]), add=T, trans = .2 )

     segments(xmids, ci[,2], xmids, ci[,3], col = cdd, lwd=6)
     arrows(xmids, ci[,1], xmids, ci[,4], col = cdd, lwd=1.5, angle=90, code=3,
           length=.03)
     points(xmids, cmid, col=cdd, cex=1.1, pch = 3, lwd = 5)

    points(xmids, cmu, pch = 15, col = cdd)

    k <- k + 1
    if(k > 26)k <- 1
    pj <- pj + 1

    lab <- colnames(ccMu)[j]

    .plotLabel( lab,above=T )
  }
  mtext( paste( xm, '(standard deviations)'), 1, outer=T)
```

```
    mtext('Equilibrium abundance', 2, outer=T)

    readline('equilibrium abundance -- return to continue ')

    o <- o + 16
    o <- o[o <= S]
  }
}
```