

Ecography

ECOG-02306

Yen, J. D. L., Thomson, J. R., Keith, J. M., Paganin, D. M. and Mac Nally, R. 2016. How do different aspects of biodiversity change through time? A case study on an Australian bird community. – Ecography doi: 10.1111/ecog.02306

Supplementary material

Appendix 1: Statistical model details for

Yen, J. D. L., Thomson, J. R., Keith, J. M., Paganin, D. M. and Mac Nally, R. How do different aspects of biodiversity change through time? A case study on an Australian bird community

Model overview

We analysed data on species richness, total abundance, total biomass, inferred total energy use, SADs, ISDs and RASs. All fitted models had the same overall form:

$$\text{response} = \text{intercept} + \text{month} + \text{season} + \text{year} + \text{residual},$$

where *response* is the community diversity measure (e.g., species richness), *intercept* is the average value of the response variable (e.g., mean species richness over all surveys), *month*, *season* and *year* are the deviations from the intercept for a given time (e.g., the deviation in Winter species richness from the overall mean species richness) and *residual* is the unexplained (within month) variation in the response variable. The exact model form depended on whether the response variable was scalar (i.e., a single value) or a function (i.e., a discrete or continuous curve).

We used a Bayesian approach to model fitting, and fitted scalar and function regression models in WinBUGS version 1.4 (Lunn et al. 2000). WinBUGS is general-purpose software for fitting Bayesian Markov chain Monte Carlo models. Function regression models were fitted using the reversible-jump add-in for WinBUGS (Lunn et al. 2006, 2008), which accommodates the transdimensional nature of the spline parameters. We based parameter estimates for scalar regression models on 3 chains of 100 000 iterations with 40 000 iteration burn-in periods. We based parameter estimates for function regression models on 3 chains of 25 000 iterations with 10 000 iteration burn-in periods. We assessed convergence and mixing of chains through

visual inspection of chains and with Gelman-Rubin statistics (Gelman and Rubin 1992). We managed all data and model outputs in R 3.1.2 (R Core Team 2014). R and WinBUGS code for model fitting is provided here.

References

- Gelman, A. and Rubin, D. B. 1992. Inference from iterative simulation using multiple sequences. - *Stat. Sci.* 7: 457-511.
- Lunn, D. J., Thomas, A., Best, N. & Spiegelhalter, D. 2000. WinBUGS -- a Bayesian modelling framework: concepts, structure, and extensibility. - *Stat. Comp.* 10: 325-337.
- Lunn, D. J., Whittaker, J. C. & Best, N. 2006. A Bayesian toolkit for genetic association studies. - *Gen. Epidem.* 30: 231-247.
- Lunn, D. J., Best, N. & Whittaker, J. 2008. Generic reversible jump MCMC using graphical models. - *Stat. Comp.* 19: 395-408.
- R Core Team. 2014. R: A language and environment for statistical computing. - R Foundation for Statistical Computing, Vienna, Austria. URL: <http://www.R-project.org/> (accessed 14 April 2015).

Model code

We prepared model inputs in R and used the R2WinBUGS package to interface with WinBUGS 1.4. Data files are not provided here, but code for simulating similar data types is included.

R code – scalar regression model

```
# R code for scalar variance partitioning model in WinBUGS

# set working directory
#setwd("/path/to/dir")

# load libraries
library(R2WinBUGS)

# prepare/load response data - simulated data are used here
richness <- rpois(100, lambda=20)
# prepare/load predictor data - simulated data are used here
groups <- cbind(sample(c(1:12), size=length(richness), replace=TRUE),
                 sample(c(1:4), size=length(richness), replace=TRUE),
                 sample(c(1:3), size=length(richness), replace=TRUE),
                 sample(c(1:3), size=length(richness), replace=TRUE))

# prepare scalar data
y <- richness
n.obs <- length(y)
MONTH <- groups[, 1]
SEASON <- groups[, 2]
YEAR <- groups[, 3]
SITE <- groups[, 4]
n.month <- length(unique(MONTH))
n.season <- length(unique(SEASON))
n.year <- length(unique(YEAR))
n.site <- length(unique(SITE))

# create subset variable to apply sum-to-zero constraints on months within seasons
month.ind <- rbind(c(1, 3), c(1, 3), c(1, 3),
                     c(4, 6), c(4, 6), c(4, 6),
                     c(7, 9), c(7, 9), c(7, 9),
                     c(10, 12), c(10, 12), c(10, 12))

# prepare bugs inputs
maxsd <- rep(max(sd(y)) / 2, 6)
save.params <- c("mu", "alpha", "beta", "phi", "delta", "gamma", "sd",
                "month.fp.sd", "season.fp.sd", "year.fp.sd", "site.fp.sd",
                "month.fp.sd.scaled", "season.fp.sd.scaled", "year.fp.sd.scaled",
                "site.fp.sd.scaled", "sd.scaled")
bugdata <- c("y", "MONTH", "SEASON", "YEAR", "SITE", "n.obs",
            "n.month", "n.season", "n.year", "n.site",
            "maxsd", "month.ind")
initials <- function(){
  list(sd=1, sd.month=1, sd.season=1, sd.year=1, sd.site=1)
}
# tell bugs which model file to use
#bugs.file <- "VAR_PART_SCALAR_BUGS.txt"

# WinBUGS settings
n.chains <- 3
n.iters <- 100000
debug <- FALSE
n.burnin <- 40000
n.thin <- 1

# need to tell bugs where to find the WinBUGS executable
#bugs.dir <- "/path/to/dir"

# run bugs model
```

```

model <- bugs(data=bugdata, inits=initials, model.file=bugs.file,
               parameters.to.save=save.params, n.chains=n.chains,
               n.iter=n.iter, n.burnin=n.burnin, n.thin=n.thin, debug=debug,
               bugs.directory=bugs.dir)

# save fitted model if needed
#save(model, file="MOD_FITTED.R")

```

WinBUGS code – scalar regression model

```

model {

  for (i in 1:n.obs) {

    y[i] ~ dnorm(mu[i], tau)

    # linear model with different exchangeable splines
    mu[i] <- alpha + beta[MONTH[i]] + phi[SEASON[i]] + delta[YEAR[i]]
           + gamma[SITE[i]]

  }

  # prior for overall model variance
  tau <- 1 / (sd * sd)
  sd ~ dunif(0, maxsd[1])
  sd.scaled <- sd / mean(mu[])

  # prior for mean
  max_prec <- 1 / (maxsd[1] * maxsd[1])
  alpha ~ dnorm(0, max_prec)

  # prior for month term
  for (month in 1:n.month) {
    beta.temp[month] ~ dnorm(0, tau.month)
    beta[month] <- beta.temp[month] -
      mean(beta.temp[month.ind[month, 1]:month.ind[month, 2]])
  }
  tau.month <- 1 / (sd.month * sd.month)
  sd.month ~ dunif(0, maxsd[3])
  month.fp.sd <- sd(beta[])
  month.fp.sd.scaled <- month.fp.sd / mean(mu[])

  # prior for season term
  for (season in 1:n.season) {
    phi.temp[season] ~ dnorm(0, tau.season)
    phi[season] <- phi.temp[season] - mean(phi.temp[])
  }
  tau.season <- 1 / (sd.season * sd.season)
  sd.season ~ dunif(0, maxsd[4])
  season.fp.sd <- sd(phi[])
  season.fp.sd.scaled <- season.fp.sd / mean(mu[])

  # prior for year term
  for (year in 1:n.year) {
    delta.temp[year] ~ dnorm(0, tau.year)
    delta[year] <- delta.temp[year] - mean(delta.temp[])
  }
  tau.year <- 1 / (sd.year * sd.year)
  sd.year ~ dunif(0, maxsd[5])
  year.fp.sd <- sd(delta[])
  year.fp.sd.scaled <- year.fp.sd / mean(mu[])

  # prior for site term
}

```

```
for (site in 1:n.site) {  
  gamma.temp[site] ~ dnorm(0, tau.site)  
  gamma[site] <- gamma.temp[site] - mean(gamma.temp[ ])  
}  
tau.site <- 1 / (sd.site * sd.site)  
sd.site ~ dunif(0, maxsd[6])  
site.fp.sd <- sd(gamma[])  
site.fp.sd.scaled <- site.fp.sd / mean(mu[ ])  
}
```

R code – function regression model

```
# R code for function-valued variance partitioning model in WinBUGS

# set working directory
#setwd("/path/to/dir")

# load libraries
library(R2WinBUGS)

# prepare/load response data - simulated data are used here
isd <- matrix(rpois(500, lambda=20), nrow=50)
# prepare/load predictor data - simulated data are used here
groups <- cbind(sample(c(1:12), size=nrow(isd), replace=TRUE),
                 sample(c(1:4), size=nrow(isd), replace=TRUE),
                 sample(c(1:3), size=nrow(isd), replace=TRUE),
                 sample(c(1:3), size=nrow(isd), replace=TRUE))

# prepare data
y <- isd
n.obs <- nrow(y)
n.bins <- ncol(y)
m <- 1:n.bins
MONTH <- groups[, 1]
SEASON <- groups[, 2]
YEAR <- groups[, 3]
SITE <- groups[, 4]
n.month <- length(unique(MONTH))
n.season <- length(unique(SEASON))
n.year <- length(unique(YEAR))
n.site <- length(unique(SITE))

# prepare bugs inputs
maxsd <- rep(sd(y) / 2, 6)
save.params <- c("mu", "alpha", "beta", "phi", "delta", "gamma",
                 "alpha.fun", "beta.fun", "phi.fun", "delta.fun",
                 "gamma.fun", "mod.sd", "rho.m", "month.fp.sd",
                 "season.fp.sd", "year.fp.sd", "site.fp.sd",
                 "mean.fp.bin", "month.fp.main", "month.fp.int",
                 "season.fp.main", "season.fp.int",
                 "year.fp.main", "year.fp.int",
                 "site.fp.main", "site.fp.int")
## remove m and maxsd if using the RAS code
bugdata <- c("y", "m", "MONTH", "SEASON", "YEAR", "SITE", "n.obs",
            "n.bins", "n.month", "n.season", "n.year", "n.site",
            "maxsd")
## SD parameters are named differently for RAS model (e.g., sd.beta1)
initials <- function(){
  list(mod.sd=1, sd.mean=1, sd.month=1, sd.season=1, sd.year=1,
       sd.site=1, rho.m=0.01, rho=rep(0.01, n.obs),
       sd.main.month=1, sd.main.season=1, sd.main.year=1,
       sd.main.site=1, sd.mean.int=1, alpha=0,
       beta.temp=rep(0, n.month), phi.temp=rep(0, n.season),
       delta.temp=rep(0, n.year), gamma.temp=rep(0, n.site))
}
# tell bugs which model file to use
#bugs.file <- "VAR_PART_SCALAR_BUGS.txt"

# WinBUGS settings
n.chains <- 3
n.iters <- 25000
debug <- FALSE
n.burnin <- 10000
n.thin <- 1
# need to tell bugs where to find the WinBUGS executable
#bugs.dir <- "/path/to/dir"

# run bugs model
model <- bugs(data=bugdata, inits=initials, model.file=bugs.file,
               parameters.to.save=save.params, n.chains=n.chains,
               n.iter=n.iters, n.burnin=n.burnin, n.thin=n.thin, debug=debug,
               bugs.directory=bugs.dir)
```

```

# save fitted model if needed
#save(model, file="MOD_FITTED.R")

WinBUGS code – function regression model (SAD and ISD)
model {

  for (i in 1:n.obs) {

    for (j in 1:n.bins) {

      # model with AR1 error structure
      y[i, j] ~ dnorm(mu.ar[i, j], tau)
      mu.ar[i, j] <- mu[i, j] + cor.e[i, j]
      resid[i, j] <- y[i, j] - mu[i, j]

      # linear model with exchangeable intercepts and splines
      # at each time scale (MONTH, SEASON, YEAR)
      mu[i, j] <- alpha + alpha.fun[j]
                  + beta[MONTH[i]] + beta.fun[MONTH[i], j]
                  + phi[SEASON[i]] + phi.fun[SEASON[i], j]
                  + delta[YEAR[i]] + delta.fun[YEAR[i], j]
                  + gamma[SITE[i]] + gamma.fun[SITE[i], j]

    }

    # details for AR1 errors
    cor.e[i, 1] <- 0
    rho[i] ~ dnorm(rho.m, 10)
    for (j in 2:n.bins) {
      cor.e[i, j] <- rho[i] * resid[i, (j - 1)]
    }

  }

  # prior for AR1 error term
  rho.m ~ dunif(-1, 1)

  # prior for overall model variance
  mod.sd ~ dunif(0, maxsd[1])
  tau <- 1 / pow(mod.sd, 2)

  # prior for grand mean (separated from the spline function
  # to improve model mixing; same for beta, phi, delta and gamma)
  alpha ~ dnorm(0, tau.mean.int)
  tau.mean.int <- 1 / pow(sd.mean.int, 2)
  sd.mean.int ~ dunif(0, maxsd[2])

  # set the maximum number of breakpoints for all splines
  kmax <- n.bins / 2

  # prior for mean curve
  for (j in 1:n.bins) {
    # subtract mean to enforce sum-to-zero constraint (needed
    # because of separate intercept term alpha)
    alpha.fun[j] <- alpha.fun.temp[j] - mean(alpha.fun.temp[])
  }
  alpha.fun.temp[1:n.bins] <- jump.pw.poly.df.lin(m[1:n.bins],
                                                 k.mean, tau.mean)
  k.mean ~ dbin(0.5, kmax)
  tau.mean <- 1 / pow(sd.mean, 2)
  sd.mean ~ dunif(0, maxsd[2])
  mean.fp.bin <- sd(alpha.fun[])
}

```

```

# priors for month term
for (j in 1:n.bins) {
  for (month in 1:n.month) {
    # subtract seasonal mean to enforce sum-to-zero within
    # seasons for each bin
    beta.fun.temp2[month, j] <- beta.fun.temp[month, j]
    - mean(beta.fun.temp[month.ind[month, 1]:month.ind[month, 2], j])
    # subtract curve mean to enforce sum-to-zero for each
    # individual curve
    beta.fun[month, j] <- beta.fun.temp2[month, j]
      - mean(beta.fun.temp2[month, ])
  }
}
for (month in 1:n.month) {
  beta.fun.temp[month, 1:n.bins] <-
    jump.pw.poly.df.lin(m[1:n.bins], k.month, tau.month)
  beta.temp[month] ~ dnorm(0, tau.main.month)
  beta[month] <- beta.temp[month]
    - mean(beta.temp[month.ind[month, 1]:month.ind[month, 2]])
}
k.month ~ dbin(0.5, kmax)
tau.month <- 1 / pow(sd.month, 2)
sd.month ~ dunif(0, maxsd[3])
tau.main.month <- 1 / pow(sd.main.month, 2)
sd.main.month ~ dunif(0, maxsd[3])

# store finite-population SDs
for (j in 1:n.bins) {
  for (k in 1:n.month) {
    beta.term[k, j] <- beta.fun[k, j] + beta[k]
  }
  month.fp.sd[j] <- sd(beta.term[, j])
}
month.fp.main <- sd(beta[])
month.fp.int <- sd(beta.fun[,])

# priors for season term
for (j in 1:n.bins) {
  for (season in 1:n.season) {
    phi.fun.temp2[season, j] <- phi.fun.temp[season, j]
      - mean(phi.fun.temp[, j])
    phi.fun[season, j] <- phi.fun.temp2[season, j]
      - mean(phi.fun.temp2[season, ])
  }
}
for (season in 1:n.season) {
  phi.fun.temp[season, 1:n.bins] <-
    jump.pw.poly.df.lin(m[1:n.bins], k.season, tau.season)
  phi.temp[season] ~ dnorm(0, tau.main.season)
  phi[season] <- phi.temp[season] - mean(phi.temp[])
}
k.season ~ dbin(0.5, kmax)
tau.season <- 1 / pow(sd.season, 2)
sd.season ~ dunif(0, maxsd[4])
tau.main.season <- 1 / pow(sd.main.season, 2)
sd.main.season ~ dunif(0, maxsd[4])

# store finite-population SDs
for (j in 1:n.bins) {
  for (k in 1:n.season) {
    phi.term[k, j] <- phi.fun[k, j] + phi[k]
  }
}

```

```

    season.fp.sd[j] <- sd(phi.term[, j])
}
season.fp.main <- sd(phi[])
season.fp.int <- sd(phi.fun[,])

# priors for year term
for (j in 1:n.bins) {
  for (year in 1:n.year) {
    delta.fun.temp2[year, j] <- delta.fun.temp[year, j]
      - mean(delta.fun.temp[, j])
    delta.fun[year, j] <- delta.fun.temp2[year, j]
      - mean(delta.fun.temp2[year, ])
  }
}
for (year in 1:n.year) {
  delta.fun.temp[year, 1:n.bins] <-
    jump.pw.poly.df.lin(m[1:n.bins], k.year, tau.year)
  delta.temp[year] ~ dnorm(0, tau.main.year)
  delta[year] <- delta.temp[year] - mean(delta.temp[])
}
k.year ~ dbin(0.5, kmax)
tau.year <- 1 / pow(sd.year, 2)
sd.year ~ dunif(0, maxsd[5])
tau.main.year <- 1 / pow(sd.main.year, 2)
sd.main.year ~ dunif(0, maxsd[5])

# store finite-population SDs
for (j in 1:n.bins) {
  for (k in 1:n.year) {
    delta.term[k, j] <- delta.fun[k, j] + delta[k]
  }
  year.fp.sd[j] <- sd(delta.term[, j])
}
year.fp.main <- sd(delta[])
year.fp.int <- sd(delta.fun[,])

# priors for site term
for (j in 1:n.bins) {
  for (site in 1:n.site) {
    gamma.fun.temp2[site, j] <- gamma.fun.temp[site, j]
      - mean(gamma.fun.temp[, j])
    gamma.fun[site, j] <- gamma.fun.temp2[site, j]
      - mean(gamma.fun.temp2[site, ])
  }
}
for (site in 1:n.site) {
  gamma.fun.temp[site, 1:n.bins] <-
    jump.pw.poly.df.lin(m[1:n.bins], k.site, tau.site)
  gamma.temp[site] ~ dnorm(0, tau.main.site)
  gamma[site] <- gamma.temp[site] - mean(gamma.temp[])
}
k.site ~ dbin(0.5, kmax)
tau.site <- 1 / pow(sd.site, 2)
sd.site ~ dunif(0, maxsd[6])
tau.main.site <- 1 / pow(sd.main.site, 2)
sd.main.site ~ dunif(0, maxsd[6])
# store finite-population SDs
for (j in 1:n.bins) {
  for (k in 1:n.site) {
    gamma.term[k, j] <- gamma.fun[k, j] + gamma[k]
  }
  site.fp.sd[j] <- sd(gamma.term[, j])
}

```

```

site.fp.main <- sd(gamma[])
site.fp.int <- sd(gamma.fun[,])

}

```

WinBUGS code – function regression model (RAS)

```

model {

  for (i in 1:n.obs) {

    for (j in 1:n.bins) {

      # model with AR1 error structure
      y[i, j] ~ dnorm(mu.ar[i, j], tau)
      mu.ar[i, j] <- mu[i, j] + cor.e[i, j]
      resid[i, j] <- y[i, j] - mu[i, j]

      # linear model with exchangeable intercepts and splines
      #   at each time scale (MONTH, SEASON, YEAR)
      mu[i, j] <- alpha + alpha.fun[j]
                  + beta[MONTH[i]] + beta.fun[MONTH[i], j]
                  + phi[SEASON[i]] + phi.fun[SEASON[i], j]
                  + delta[YEAR[i]] + delta.fun[YEAR[i], j]
                  + gamma[SITE[i]] + gamma.fun[SITE[i], j]

    }

    # details for AR1 errors
    cor.e[i, 1] <- 0
    rho[i] ~ dnorm(rho.m, 10)
    for (j in 2:n.bins) {
      cor.e[i, j] <- rho[i] * resid[i, (j - 1)]
    }

  }

  # prior for AR1 term
  rho.m ~ dunif(-1, 1)

  # prior for overall model variance
  mod.sd ~ dunif(0, 10)
  tau <- 1 / pow(mod.sd, 2)

  # prior for grand mean (separated from the spline function
  #   to improve model mixing; same for beta, phi, delta and gamma)
  alpha ~ dnorm(0, 0.001)
  # prior for "spline" term for the model intercept
  for (j in 1:n.bins) {
    alpha.fun.temp[j] ~ dnorm(0, tau.mean)
    # subtract mean to enforce sum-to-zero constraint (needed
    #   because of separate intercept term alpha)
    alpha.fun[j] <- alpha.fun.temp[j] - mean(alpha.fun.temp[])
  }
  tau.alpha <- 1 / pow(sd.alpha, 2)
  sd.alpha ~ dunif(0, 10)

  # priors for month term
  for (month in 1:n.month) {
    beta[month] <- beta.temp[month] - mean(beta.temp[])
    beta.temp[month] ~ dnorm(0, tau.beta)
  }

}

```

```

        for (j in 1:n.bins) {
            beta.fun.temp[month, j] ~ dnorm(0, tau.beta2)
            beta.fun.temp2[month, j] <- beta.fun.temp[month, j] -
mean(beta.fun.temp[month, ])
            beta.fun[month, j] <- beta.fun.temp2[month, j]
            - mean(beta.fun.temp2[month.ind[month,
1]:month.ind[month, 2], j])
        }
    }
tau.beta <- 1 / pow(sd.beta, 2)
tau.beta2 <- 1/ pow(sd.beta2, 2)
sd.beta ~ dunif(0, 10)
sd.beta2 ~ dunif(0, 10)
month.fp.int <- sd(beta.fun[,])
month.fp.main <- sd(beta[])

# priors for season term
for (season in 1:n.season) {
    phi[season] <- phi.temp[season] - mean(phi.temp[])
    phi.temp[season] ~ dnorm(0, tau.phi)
    for (j in 1:n.bins) {
        phi.fun.temp[season, j] ~ dnorm(0, tau.phii2)
        phi.fun.temp2[season, j] <- phi.fun.temp[season, j] -
mean(phi.fun.temp[season, ])
        phi.fun[season, j] <- phi.fun.temp2[season, j] -
mean(phi.fun.temp2[, j])
    }
}
tau.phi <- 1 / pow(sd.phi, 2)
tau.phii2 <- 1/ pow(sd.phii2, 2)
sd.phi ~ dunif(0, 10)
sd.phii2 ~ dunif(0, 10)
season.fp.int <- sd(phi.fun[,])
season.fp.main <- sd(phi[])

# priors for year term
for (year in 1:n.year) {
    delta[year] <- delta.temp[year] - mean(delta.temp[])
    delta.temp[year] ~ dnorm(0, tau.delta)
    for (j in 1:n.bins) {
        delta.fun.temp[year, j] ~ dnorm(0, tau.delta2)
        delta.fun.temp2[year, j] <- delta.fun.temp[year, j] -
mean(delta.fun.temp[year, ])
        delta.fun[year, j] <- delta.fun.temp2[year, j] -
mean(delta.fun.temp2[, j])
    }
}
tau.delta <- 1 / pow(sd.delta, 2)
tau.delta2 <- 1/ pow(sd.delta2, 2)
sd.delta ~ dunif(0, 10)
sd.delta2 ~ dunif(0, 10)
year.fp.int <- sd(delta.fun[,])
year.fp.main <- sd(delta[])

# priors for site term
for (site in 1:n.site) {
    gamma[site] <- gamma.temp[site] - mean(gamma.temp[])
    gamma.temp[site] ~ dnorm(0, tau.gamma)
    for (j in 1:n.bins) {
        gamma.fun.temp[site, j] ~ dnorm(0, tau.gamma2)
        gamma.fun.temp2[site, j] <- gamma.fun.temp[site, j] -
mean(gamma.fun.temp[site, ])
    }
}

```

```
gamma.fun[site, j] <- gamma.fun.temp[site, j] -  
mean(gamma.fun.temp[, j])  
}  
}  
tau.gamma <- 1 / pow(sd.gamma, 2)  
tau.gamma2 <- 1/ pow(sd.gamma2, 2)  
sd.gamma ~ dunif(0, 10)  
sd.gamma2 ~ dunif(0, 10)  
site.fp.int <- sd(gamma.fun[,])  
site.fp.main <- sd(gamma[,])  
  
}
```