

Appendix 2: Simulating 20-species landscapes

Inferring species interactions from co-occurrence data with Markov networks

David J. Harris

Simulating landscapes from known coefficients

The following function draws random coefficients for a Markov network of a pre-specified size (20 species in these examples).

The α coefficients controlling species' intercepts were drawn from Gaussian distributions with unit variance and a mean given by `mean_alpha`. In the R code at the end of this document, `mean_alpha` is defined as -1 for presence-absence simulations and 5 for the abundance simulations.

The β coefficients' magnitudes were drawn from an exponential distribution with rate parameter equal to 1. The sign of each coefficient was randomly sampled, based on the value of `p_neg`, defined below as 0.75 (75% negative) for the presence-absence simulations and 1.0 (100% negative) for the simulations that included abundance.

```
make_coefficients = function(n_spp, p_neg, mean_alpha){  
  # Exponential distribution has lots of mass near 0 but has  
  # a long tail.  
  true_beta_magnitudes = rexp(choose(n_spp, 2), rate = 1)  
  
  # Multiply some proportion of the interactions  
  # by -1  
  b = true_beta_magnitudes * sample(  
    c(-1, 1),  
    size = length(true_beta_magnitudes),  
    prob = c(p_neg, 1 - p_neg),  
    replace = TRUE  
  )  
  
  # Species' intercepts are normally distributed  
  a = rnorm(n_spp, mean_alpha)  
  
  # Return the simulated values.  
  # The rosalia function stores pairwise parameters in the upper  
  # triangle of an n-by-n matrix and stores the species intercepts  
  # along the diagonal, so these values are named accordingly.  
  c(alpha = a, beta = b)  
}
```

The next function calls the function above to generate “true” parameters, then uses them to simulate a landscape with Markov chain Monte Carlo, and finally saves the results to a “fakedata” folder. I

used Gibbs sampling as my Markov chain Monte Carlo transition operator. In each round of Gibbs sampling, I cycled through all the species, updating each one's binary presence/absence vector with random samples from a Bernoulli distribution whose conditional probability was defined by

$$p(y_i) = \text{logistic}(\alpha_i + \sum_j \beta_{ij} y_j),$$

where the logistic function is $\frac{1}{1+e^{-x}}$. For efficiency, $\sum_j \beta_{ij} y_j$ is calculated with matrix multiplication in the code below.

For simulations that included environmental heterogeneity, α_i varied from site to site as follows:

- **n_env** environmental variables were simulated for each site; these were drawn from zero-mean Gaussians with standard deviation given by **sd**. In the R code at the end of this document, both **n_env** and **sd** were set to 2, corresponding to an environment with two important axes of variation and had a standard deviation of two.
- The local value of α_i for species i is given by the species-level α defined by the **make_coefficients** function, plus a linear combination of the environmental variables. For efficiency, this linear combination is calculated with matrix multiplication. The weights associated with this linear combination (**alpha_env**) are drawn from a Gaussian distribution with zero mean and unit variance.

For the abundance-based simulations, each species' conditional abundance was simulated with a Poisson distribution instead of a Bernoulli distribution. The rate parameter of this distribution (which corresponds to the expected abundance) was given by

$$y_i \sim \log(1 + \exp(\alpha_i + \sum_j \beta_{ij} y_j))$$

(The more conventional exponential inverse link was not used because preliminary experiments showed that it led to very large y values, which prevented the Gibbs sampler from mixing effectively.)

Each simulated landscape was saved after the Gibbs sampler had updated the value of each species **n_gibbs** times; see the end of this document for the value used.

```
simulate_data = function(n_spp, n_sites, rep_name, n_gibbs, n_env, sd, f,
                        rdist, p_neg, mean_alpha){
  # n_spp:      number of species to include in the landscape
  # n_sites:    number of sites to include in the landscape
  # rep_name:   an identifier to use for the landscape replicate
  # n_gibbs:    number of Gibbs samples to perform
  # n_env:      number of environmental variables to simulate
  # sd:         standard deviation of environmental variables (can be zero)
  # f:          inverse link function (see above for two examples)
  # rdist:      a function for sampling a random value from a distribution
  # p_neg:      proportion of negative interactions (e.g. competition)
  # mean_alpha: the intercept value for the average species
```

```

# Determine the "true" parameters for the simulated assemblage
par = make_coefficients(n_spp, p_neg, mean_alpha)

# "True" interaction strengths, to save for later
truth = par[-(1:n_spp)]

# "True" intercepts, possibly adjusted below by environment
alpha = par[1:n_spp]

# Turn the interaction values into an n-by-n matrix
# Start with empty matrix; fill in upper triangle;
# then fill in lower triangle with its transpose
beta = matrix(0, n_spp, n_spp)
beta[upper.tri(beta)] = truth
beta = beta + t(beta)

# Environmental states are normally distributed with mean=0 and sd=1
env = matrix(rnorm(n_sites * n_env), ncol = n_env)

alpha_env = matrix(rnorm(n_spp * n_env, sd = sd), nrow = n_env)

# Simulate the landscape from known process with Gibbs sampling

# Landscape starts as if betas were all zero. Each species' occurrence
# probability or abundance depends on its alpha value and on the
# environment (assuming alpha_env is not set to zero).
x = matrix(
  f(rep(1, n_sites) %*% t(alpha) + env %*% alpha_env),
  nrow = n_sites,
  ncol = n_spp
)

# Gibbs sampling
for(i in 1:n_gibbs){

  # Each round of Gibbs sampling updates one species (column) across all sites
  # according to its conditional probability (i.e. conditional on environment
  # and the other species that are present).
  for(j in 1:n_spp){
    x[,j] = rdist(

```

```

    nrow(x),
    f(x %*% beta[ , j] + alpha[j] + env %*% alpha_env[,j])
  )
}
}

# Collapse abundance data to presence/absence and store
# it as integer values rather than true/false
x = x > 0
mode(x) = "integer"

colnames(x) = paste0("V", 1:n_spp)

# Save the results in a "fake data" folder

file_stem = paste(n_sites, rep_name, sep = "-")

# Gotelli and Ulrich's Pairs software rejects empty sites, so I remove them here
x_subset = x[rowSums(x) != 0, colSums(x) != 0]

# Save the matrix of presence/absence observations
write.csv(
  x,
  file = paste0("fakedata/matrices/", file_stem, ".csv")
)

# Gotelli and Ulrich's Pairs method expects the data matrices to be transposed,
# So I save them separately
write.table(
  t(x_subset),
  file = paste0("fakedata/matrices/", file_stem, "-transposed.txt"),
  quote = FALSE
)

# Save the "true" species interactions
write(
  truth,
  file = paste0("fakedata/truths/", file_stem, ".txt"),
  ncolumns = length(truth)
)
}

# Define a convenience function for Bernoulli random samples
rbern = function(n, prob){
  rbinom(n = n, size = 1, prob = prob)
}

```

```
}
```

A loop to simulate and save 50 landscapes for each combination of simulation type and landscape size, using the functions defined above:

```
set.seed(1)
n_spp = 20

for(n_sites in c(25, 200, 1600)){
  for(type in c("no_env", "env", "abund")){
    for(i in 1:50){
      simulate_data(
        n_spp = n_spp,
        n_sites = n_sites,
        n_gibbs = 1000,
        n_env = 2,
        rep_name = paste0(type, i),
        sd = ifelse(type == "env", 2, 0),
        f = if(type == "abund"){function(x){log(1 + exp(x))}}else{plogis},
        rdist = if(type == "abund"){rpois}else{rbern},
        p_neg = if(type == "abund"){1}else{0.75},
        mean_alpha = if(type == "abund"){5}else{-1}
      )
    }
  }
}
```