Appendix [[X]] simulating species interactions

"Estimating species interactions from observational data with Markov networks"

David J. Harris

This code simulates from a stochastic demographic model, then saves the results for subsequent analysis.

```r
# Initialize the random number generator for reproducibility
set.seed(1)

thin = 5E3  # Save every 5000th sample
n_spp = 20  # 20 species
n_reps = 50 # 50 replicates for each landscape size
```

In the large population limit, the number of individuals of species i, n_i, evolves over time based on the following differential equation, based on multi-species Lotka-Volterra dynamics. The basic multi-species model is

$$\frac{\mathrm{d}n_i}{\mathrm{d}t} = r_i n_i (1 - \frac{n_i + \sum \alpha_{ij} n_j}{K}), \tag{1}$$

where r_i is the per-capita growth rate of species i, n_i is its abundance, α_{ij} describes the strength of its competition with species j, and K is the carrying capacity. These simulations use a more general form, which includes propagules from outside the system (r_e) and a separate mechanism for mutualisms ($f_i(\vec{n})$).

$$\frac{\mathrm{d}n_i}{\mathrm{d}t} = r_e + r_i n_i (1 - \frac{n_i f_i(\vec{n}) + \sum \alpha_{ij} n_j}{K}). \tag{2}$$

Setting $r_e = 0$ and $f_i(\vec{n}) = 1$ recovers the original Lotka-Volterra dynamics. The function $f_i(\vec{n})$ acts to reduce the strength of intraspecific competition in the presence of mutualists. For these simulations, I used

$$f_i(\vec{n}) = \frac{1}{2}(1 + \exp(-\frac{\sum_j \gamma_{ij} n_j}{5K}), \tag{3}$$

where γ_{ij} is the strength of the mutualism between species i and species j. The nonlinearity ensures that mutualisms can never eliminate more than half of the effects of intraspecific competition and destabilize the system too strongly. In these simulations, each species pair acted either as competitors $\alpha_{ij} > 0$ or as mutualists $\gamma_{ij} > 0$, but not both. Note that α_{ii} and γ_{ii} were always zero (i.e. intraspecific competition is modeled outside of the α term and species cannot be their own mutualists).

Equation 2 can be decomposed into a birth rate, $r_e + r_i n_i$, and a death rate, $r_i n_i (\frac{n_i f_i(\vec{n}) + \sum \alpha_{ij} n_j}{K})$. If we treat births and deaths as discrete events (e.g. if we leave the large population regime), then the probability that the next event will be a birth (or death) of species i is directly proportional to the corresponding demographic rate.

Because these events' probabilities only depend on the current state of the system, they form a Markov chain that explores the space of possible species assemblages under the population dynamic model. For each set of competition and mutualism coefficients, I simulated births and deaths in this chain, collecting one sample after every 5000th demographic event. For the subsequent analysis, each species was considered "present" in a sample if one or more individuals was present, and "absent" if its abundance in that sample was zero.

The following function randomly generates a set of α and γ values, then simulates `maxit` demographic events with those parameters, retaining every `thin`=5000 samples.

```r
simulate_communities = function(n_spp = 20,
         K = n_spp / 3,
         maxit = 5E6,
         thin = 1E4,
         seed_rain = rep(.1, n_spp),
         growth = rep(1, n_spp),
         interaction_strength = 1/2,
         prob_negative = 0.75,
         mutualism_maximum = 0.5
){

  # Intraspecific competition can be mitigated by mutualists, up to some
  # maximal amount.  This function rescales the mutualism effect so that
  # it doesn't turn off intraspecific competition entirely.
  mutualism_scaler = function(x, mutualism_maximum){
    (1 + exp(- x / 5 / K)) * (1 - mutualism_maximum)
  }

  # Randomly generate a vector of interaction strengths (one for each
  # pair of species)
  interaction_vector = rexp(choose(n_spp, 2), interaction_strength)

  # Randomly mark interactions as positive or negative
  interaction_signs = ifelse(
    rbinom(choose(n_spp, 2), size = 1, prob = prob_negative),
    -1,
    1
  )

  # Negative interactions are competition; positive ones are mutualisms
  competition_vector = interaction_vector * (interaction_signs == -1)
  mutualism_vector = interaction_vector * (interaction_signs == +1)

  # Fill in the competition and mutualism matrices:
  # Start with a matrix of zeros
  competition = matrix(0, nrow = n_spp, ncol = n_spp)
```

```r
# Fill in the upper triangle with the appropriate interaction vector
competition[upper.tri(competition)] = competition_vector
# Symmetrize the matrix by adding it to its transpose
competition = competition + t(competition)

# Note that intraspecific competition (which would go along the diagonal)
# is handled separately, and the diagonal is left with zeros.

mutualism = matrix(0, nrow = n_spp, ncol = n_spp)
mutualism[upper.tri(mutualism)] = mutualism_vector
mutualism = mutualism + t(mutualism)

# Create an empty matrix for storing population sizes
pops = matrix(nrow = n_spp, ncol = maxit / thin)


# Initialize the population with a fixed number of individuals per species,
# depending on the number of species, carrying capacity, and strength of
# interspecific competition.
population = rep(ceiling(K / n_spp / mean(competition)), n_spp)


for(i in 1:maxit){

  # The local growth rate is proportional to population size
  local_growth = growth * population

  # Mutualists act by mitigating intraspecific competition
  mutualist_mitigation = mutualism_scaler(
    population %*% mutualism,
    mutualism_maximum
  )

  # Add up the two types of interactions affecting death rates
  interactions = population * mutualist_mitigation + population %*% competition

  # Germinating seeds can come from local growth or from outside the system
  birthrates = seed_rain + local_growth

  # Death rates depend on population size, interactions, and carrying capacity
  deathrates = local_growth * interactions / K

  # Generate a vector of vital rates
  rates = c(birthrates, deathrates)

  # Randomly select a birth/death event to occur. Probability
  # of occurring is proportional to the corresponding rate
```

```r
    event = sample.int(length(rates), 1, prob = rates)


    if(event <= length(birthrates)){
      # Event from first batch, i.e. birth.  Increase the population by one
      index = event
      population[index] = population[index] + 1
    }else{
      # Event from the second batch, i.e. death. If population is nonzero,
      # decrease it by one for the corresponding species
      index = event - n_spp
      if(population[index] > 0){
        population[index] = population[index] - 1
      }
    }


    if(i%%thin == 0){
      # Save the results in the `pops` matrix
      pops[ , i %/% thin] = population
    }
  }

  # Return the "observed" presence-absence vector
  # along with the"true" interaction signs and strengths.
  # The output is transposed because all fitting methods but Pairs
  # require it in that format.
  # Multiplying by one turns TRUEs into 1s and FALSEs into 0s
  list(
    observed = t(pops > 0) * 1,
    truth = interaction_vector * interaction_signs
  )
}
```

I called the above code 50 times for each landscape size, saving the results in a folder called `fakedata` in two formats (one for Gotelli and Ulrich's Pairs software, and one for the other methods).

```r
for(n_sites in c(25, 200, 1600)){
  message(n_sites)

  for(rep in 1:n_reps){
    id = paste(n_spp, n_sites, rep, sep = "-")

    results = simulate_communities(
      maxit = n_sites * thin,
      thin = thin,
      n_spp = n_spp
    )

    saveRDS(results, file = paste0("fakedata/", id, ".rds"))

    # Gotelli and Ulrich's Pairs software needs a different format.
    x = results$observed

    # It rejects empty sites, so I remove them here
    x_subset = x[rowSums(x) != 0, colSums(x) != 0]

    # It also expects the data matrices to be transposed
    # relative to the other methods
    write.table(
      t(x_subset),
      file = paste0("fakedata/", id, "-transposed.txt"),
      quote = FALSE
    )
  }
}
```