Estimating pairwise interactions by stochastic maximum likelihood

```
library(rosalia)
library(mistnet)
`%plus%` = mistnet:::`%plus%` # convenience function for adding intercepts to each column
logistic = binomial()$linkinv # logistic inverse link
rbern = function(p){rbinom(length(p), size = 1, prob = p)} # Random bernoulli trial
```

```
set.seed(1)
n_spp = 250     # number of species
n_loc = 2500    # number of locations
n_env = 5       # number of environmental predictors
n_gibbs = 5000  # number of Gibbs sampling iterations


# What portion of the coefficients should come from each mixture component?
type_frequencies = rmultinom(1, size = choose(n_spp, 2), prob = c(.2, .5, .3))


# Vector of "true" interaction strengths.
# Three mixture components, shuffled by `sample`.
true_beta_vec = sample(
  c(
    rnorm(type_frequencies[[1]], 0, 2),
    rnorm(type_frequencies[[2]], 0, .2),
    rnorm(type_frequencies[[3]], -.5, .5)
  )
)

# Visualize the distribution of interaction terms
plot(density(true_beta_vec))

# Make a symmetric matrix of "true" beta coefficients
true_beta = matrix(0, nrow = n_spp, ncol = n_spp)
true_beta[upper.tri(true_beta)] = true_beta_vec
true_beta = true_beta + t(true_beta)

# Create a set of environmental predictors
true_env = matrix(rnorm(n_loc * n_env, 0, 2.5), nrow = n_loc)

# "true" responses of each species to each environmental variable
true_alpha_env = matrix(rnorm(n_spp * n_env, 0, 1), nrow = n_env)

# "true" intercepts for each species
true_alpha_species = rnorm(n_spp, 2)

# site-level intercept depends on species baselines environmental responses
true_alpha = true_env %*% true_alpha_env %plus% true_alpha_species
```

```r
# Initialize the `y` matrix.
# This will hold the "observed" presence-absence data
y = matrix(0.5, nrow = n_loc, ncol = n_spp)

# For each round of Gibbs sampling...
for(i in 1:n_gibbs){
  # For each species (in random order)...
  for(j in sample.int(n_spp)){
    # update its occurrence with samples from the conditional distribution
    y[,j] = rbern(logistic(true_alpha[ , j] + y %*% true_beta[ , j]))
  }
}
```

```r
# Calculate sufficient statistics of the data
y_stats = crossprod(y)
y_env_stats = t(true_env) %*% y
```

```r
# Initialize the simulated landscape for stochastic approximation
y_sim = matrix(0.5, nrow = nrow(y), ncol = ncol(y))

# In this example, the true state of the environment is known without error
env = true_env

# Initialize species' responses to environment at 0.
# Also initialize the delta (change in parameter values from the
# previous optimization iteration) to zero, since no optimization
# has occurred yet
alpha_env = delta_alpha_env = matrix(0, nrow = n_env, ncol = n_spp)

# Initialize species' intercepts to match observed occurrence rates
# plus a small amount of regularization
alpha_species = qlogis((colSums(y) + 1) / (nrow(y) + 2))

# Initialize the deltas for the intercepts to zero
delta_alpha_species = rep(0, n_spp)

# Initialize pairwise interactions and deltas to zero
beta = delta_beta = matrix(0, nrow = n_spp, ncol = n_spp)

# overall alpha depends on alpha_species and alpha_env.
# Will be filled in later, so can initialize it with zeros
alpha = matrix(0, nrow = n_spp, ncol = n_spp) # no delta alpha to initialize
                                              # b/c alpha not optimized directly
```

```r
# Very weak priors on alpha terms, somewhat stronger on beta terms
alpha_env_prior = rosalia::make_logistic_prior(scale = 2)$log_grad
alpha_species_prior = rosalia::make_logistic_prior(scale = 2)$log_grad
beta_prior = rosalia::make_logistic_prior(scale = 0.5)$log_grad
```

```r
initial_learning_rate = 1 # step size at start of optimization
maxit = 50000             # Number of rounds of optimization
start_time = as.integer(Sys.time())
```

```
# Record the R-squared values in this vector
r2s = numeric(maxit)

# Record the timing history in this vector
times = integer(maxit)

for(i in 1:maxit){
  ##############################
  # Gibbs sampling for predicted species composition
  ##############################

  # Update alpha
  alpha = env %*% alpha_env %plus% alpha_species

  # Sample entries in y_sim from their conditional distribution (Gibbs sampling)
  for(j in sample.int(n_spp)){
    y_sim[,j] = rbern(logistic(alpha[ , j] + y_sim %*% beta[ , j]))
  }

  ##############################
  # Stochastic approximation for updating alpha and beta
  ##############################

  # Update learning rate and momentum
  learning_rate = initial_learning_rate * 1000 / (998 + 1 + i)
  momentum = .9 * (1 - 1/(.1 * i + 2))

  # Calculate sufficient statistics
  y_sim_stats = crossprod(y_sim)
  y_sim_env_stats = t(env) %*% y_sim

  # Calculate the gradient with respect to alpha and beta.
  # Gradients are differences in sufficient statistics plus prior
  # gradients, all divided by the number of locations
  stats_difference = y_stats - y_sim_stats
  beta_grad = (stats_difference + beta_prior(beta)) / n_loc

  alpha_species_grad = (diag(stats_difference) + alpha_species_prior(alpha_species)) / n_loc
  diag(beta_grad) = 0 # beta_ii is 0 by convention
  y_env_difference = y_env_stats - y_sim_env_stats
  alpha_env_grad = (y_env_difference + alpha_env_prior(alpha_env))  / n_loc


  # Calculate parameter updates: gradient times learning rate plus momentum times delta
  delta_beta = beta_grad * learning_rate + momentum * delta_beta
  delta_alpha_species = alpha_species_grad * learning_rate + momentum  * delta_alpha_species
  delta_alpha_env = alpha_env_grad * learning_rate +  momentum  * delta_alpha_env

  # Add the deltas to the previous parameter values
  beta = beta + delta_beta
  alpha_species = alpha_species + delta_alpha_species
  alpha_env = alpha_env + delta_alpha_env
```

```
  # Record R-squared and timing
  r2s[i] = cor(true_beta[upper.tri(true_beta)], beta[upper.tri(beta)])^2
  times[i] = as.integer(Sys.time()) - start_time
}
```

```
library(cowplot)
library(ggplot2)
out = plot_grid(
  ggplot(NULL, aes(x = beta[upper.tri(beta)], y = true_beta[upper.tri(beta)])) +
    stat_binhex() +
    xlab("Estimated coefficient value") +
    ylab("\"True\" coefficient value") +
    stat_hline(yintercept = 0, size = 1/8) +
    stat_vline(xintercept = 0, size = 1/8) +
    scale_fill_gradient(low = "#F0F0F0", high = "darkblue", trans = "log10") +
    stat_abline(intercept = 0, slope = 1, size = 1/2) +
    coord_equal(),
  ggplot(NULL, aes(x = c(alpha_env), y = c(true_alpha_env))) +
    stat_binhex() +
    xlab("Estimated coefficient value") +
    ylab("\"True\" coefficient value") +
    stat_hline(yintercept = 0, size = 1/8) +
    stat_vline(xintercept = 0, size = 1/8) +
    scale_fill_gradient(low = "#F0F0F0", high = "darkblue", trans = "log10") +
    stat_abline(intercept = 0, slope = 1, size = 1/2) +
    coord_equal(),
  nrow = 1,
  labels = c("A. Pairwise biotic coefficients (n = 31125)", "B. Abiotic coefficients (n = 1250)")
)
save_plot("stochastic/estimates.pdf", out, base_aspect_ratio = 2, base_height = 6)
```

Small landscape

```
small_n_spp = 20
small_n_loc = 500
small_true_beta = true_beta[1:small_n_spp, 1:small_n_spp]
small_true_alpha = true_alpha_species - 2
small_y = matrix(0.5, nrow = small_n_loc, ncol = small_n_spp)

for(i in 1:n_gibbs){
  for(j in sample.int(small_n_spp)){
    small_y[,j] = rbern(logistic(small_true_alpha[j] + small_y %*% small_true_beta[ , j]))
  }
}
```

```
exact_time = system.time({
  exact = rosalia(small_y, prior = make_logistic_prior(scale = 1), maxit = 500)
})

mle_beta = exact$beta
```

```
maxit_small = 50000
start_time_small = as.numeric(Sys.time())

beta_prior_small = rosalia::make_logistic_prior(scale = 1)$log_grad
alpha_species_prior_small = rosalia::make_logistic_prior(scale = 1)$log_grad

alpha_small = qlogis((colSums(small_y) + 1) / (nrow(small_y) + 2))
delta_alpha_small = rep(0, small_n_spp)
beta_small = delta_beta_small = matrix(0, nrow = small_n_spp, ncol = small_n_spp)

y_sim_small = matrix(0.5, nrow = small_n_loc, ncol = small_n_spp)

y_stats = crossprod(small_y)

mses = numeric(maxit_small)
small_times = numeric(maxit_small)

for(i in 1:maxit_small){
  ##############################
  # Gibbs sampling for predicted species composition
  ##############################

  # Sample entries in y_sim from their conditional distribution (Gibbs sampling)
  for(j in sample.int(small_n_spp)){
    y_sim_small[,j] = rbern(logistic(alpha_small[j] + y_sim_small %*% beta_small[ , j]))
  }

  ##############################
  # Stochastic approximation for updating alpha and beta
  ##############################

  # Update learning rate and momentum
  learning_rate = initial_learning_rate * 1000 / (998 + 1 + i)
  momentum = .9 * (1 - 1/(.1 * i + 2))

  # Calculate sufficient statistics
  y_sim_stats = crossprod(y_sim_small)

  # Calculate the gradient with respect to alpha and beta
  stats_difference = y_stats - y_sim_stats
  beta_grad = (stats_difference + beta_prior_small(beta_small)) / small_n_loc

  alpha_species_grad = (diag(stats_difference) +
                          alpha_species_prior_small(alpha_small)) / small_n_loc
  diag(beta_grad) = 0

  # Calculate parameter updates
  delta_beta = beta_grad * learning_rate +
    momentum * delta_beta_small
  delta_alpha_species = alpha_species_grad * learning_rate +
    momentum  * delta_alpha_small

  beta_small = beta_small + delta_beta
```

```
  alpha_small = alpha_small + delta_alpha_species

  mses[i] = mean((beta_small[upper.tri(beta_small)] - exact$beta[upper.tri(exact$beta)])^2)
  small_times[i] = as.numeric(Sys.time()) - start_time_small
}
```

Plotting results

```
pdf("stochastic/convergence.pdf", width = 10, height = 7)
par(mfrow = c(1, 2), las = 1)

plot(small_times, mses, type = "l", ylab = "mean square deviation from MLE\n(log scale)", xlab = "time

plot(c(0, times / 60 / 60), c(0, r2s), type = "l", xlab = "time (hours)", ylab = "R^2",ylim = c(0, max(
abline(h = max(r2s), col = "#00000080", lty = 2, lwd = 2)
dev.off()
```