Anime description 11/03/2016

Current contributors: JD Yeakel, T Gross, MM Pires, MAM de Aguiar, PR Guimaraes, J O'Donnell

Note: all of the code is written in `julia` and can be found and accessed publicly via github at: `https://github.com/jdyeakel/Lego` Required packages include `Distributions` for statistical functions and `Gadfly` for basic plotting. Eventually, I will likely use the package `RCall` to plot results from the simulations in `R`, as plotting things in `julia` is a real pain. The `Lego` thing is a bit of an historical artifact, as the project is now referred to as the `ANIME` project. **Importantly** this code works for `julia 0.5` or above, and does not work for earlier versions, as they changed how arrays are created/stored in v0.5. That was fun to discover (not)

Basic layout

The directed interactions come in five flavors:

> `a`: assimilate `n`: need `i`: ignore `m`: make `e`: exclude (emergent property)

Combinations of these interactions form specific ecological pairwise interactions

Interaction	Biological meaning
{n,a}	Trophic mutualism
{n,n}	Non-trophic mutualism
{n,i}	Commensalism
{n,m}	Engineering
{i,a}	Asymmetric Predation
{a,a}	Symmetric predation
{i,i}	Stable competition, coexistence
{i,e}	Asymmetric unstable competitive exclusion
{e,e}	Symmetric unstable competitive exclusion

The `n: need` and `a: assimilate` interactions are functionally quite similar. Conceptually, the `a` interaction defines a trophic interaction where biomass flow between one species to another is assumed, whereas the `n` interaction defines any other type of need. For example, the `{n,a}` pairwise interaction represents a trophic mutualism, where species 1 eats species 2, and species 2 needs species 1 for reproduction, say. This could be a classic plant/pollinator type of interaction. The `{n,n}` interaction is another form of mutualism where biomass is not passed from one species to the other. The `{n,i}` interaction is a commensal interaction - for example, a bird might `n: need` a tree (to build a nest), but the tree effectively

`i: ignores` the bird.

It is also important to note that this interaction matrix is composed of elements that may be living species, but may also be objects that are created and used by living species. Thus, we can disentangle both direct interactions between species as well as the indirect interactions.

For community assembly, we assume that species that coexist do so at some stable steady state. Accordingly, asymmetric predation involves an `a` interaction in one direction (predator eats the prey) and an `i` interaction in the other. These seems strange at first glance, but essentially says that the prey views the existence of the predator simply as an additional mortality source, effectively ignoring it from the predator's perspective. This perspective on species interactions thus operates with respect to species presence/absence in the system rather than with respect to their population densities. As I'll mention in detail below, we can still account - in some way - for the numerical response of species by using the interaction combinations to generate extinction probabilities.

Building the master interaction matrix

First, a *master interaction matrix* is constructed, which defines how each species/object interacts with every other species/object. The *master interaction matrix* is built in `Lego_Program/src/build_template_degrees.jl`

System Size N: The size of the system, which is composed of **species** and the **objects** that are made by species is `N`. An species makes an object through interaction `m`. An object needs the species that makes it with interaction `n`.

I currently have it set up so that the input is the probability that each interaction type is drawn (except `e: exclude`, which is an emergent property and described in more detail below). E.g. pr_a = 0.01, pr_n = 0.005, pr_m = 0.005, pr_i = 1 - (pr_a) + pr_n + pr_m), and it is assumed that most directed interactions between species are `ignore`. Maybe these probability are scaled to `N`.

We can then calculate the probabilities of pairwise interactions:

```
pr_na = p_n*(p_a/(p_a+p_n+p_i+p_m)) + p_a*(p_n/(p_a+p_i+p_n))
pr_nn = p_n*(p_n/(p_a+p_n+p_i+p_m)) pr_ni = p_n*(p_i/(p_a+p_n+p_i+p_m))
+ p_i*(p_n/(p_a+p_n+p_i)) pr_nm = p_n*(p_m/(p_a+p_n+p_i+p_m))
+ p_m pr_ia = p_i*(p_a/(p_a+p_n+p_i)) + p_a*(p_i/(p_a+p_i+p_n))
pr_ii = p_i*(p_i/(p_a+p_n+p_i)) pr_aa = p_a*(p_a/(p_i+p_n+p_a))
```

Trophic interactions: The core of the `a: assimilate` (trophic) interactions is determined by the niche model (the number of interactions per species rather than the structure of interactions). First we make an estimate of the number of living species that will be in the system by discounting objects, which are made by species, where `S = N - (pr_nm*N)`. There are 3 different types of trophic interactions: `a-i`, `a-a`, and `a-n`. Thus, the estimated Connectance

of the trophic network that is embedded in the larger interaction network is `C = ((pr_ia+pr_na+pr_aa)*(S*(S-1)))/(S^2)`. For each element of the interaction network, establish the number of trophic interactions following the niche model. Because in our case the niche axis only serves to obtain a distribution of trophic links, the structure of interactions is not retained (though this could be changed). Each element of the matrix must have at least one trophic interaction.

For each `a` interaction determined from the above steps, the opposing interaction is drawn based on `pr_aa, pr_ai, pr_an`.

Non-trophic interactions: For non-trophic interactions, we draw from the remaining possible interactions with probability `pr_nn, pr_ni, pr_nm, pr_ii`.

Fine-tuning *Basal resource:* We assume that row/column 1 of the interaction matrix is always the basal resource of the system. So anything that `a: assimilates` it is a primary producer. We assume that it ignores everything, such that row 1 = `i`, and does not make objects (it essentially serves as a passive resource).

Fixed primary producer: Row/Column 2 is established as a primary producer always (so there is at least one) with no `n: need` requirement. This allows the community assembly process to always have at least one species that is independent of the others and a primary producer that can serve as the founder.

Made objects: Through the `m: make` interaction, we now have both species and objects in the network, and we will assume that made objects have different interaction constraints than living species. We go through the master interaction matrix and find which elements are now objects made by other species. We impliment some rules: 1) Made objects must `n: need` the species that make them (note: multiple species can make the same object with some probability); 2) Made objects ignore everything else (a property similar to the basal resoure), such that species can `a: assimilate`, `n: need`, or `m: make` objects, however objects only `n: need` the species that make them.

Diagonal identities: The diagonal will be used to quickly identify which element of the network is a species or object. Species will have a `n: need` interaction on the diagonal (they need themselves to exist), and objects will have a `i: ignore` interaction on the diagonal.

Calculate similarity For efficiency, we can calculate the similarity of each species with every other species in the network by comparing their non-ignore interactions. This will be used later on to assess competitive overlap, which influences the probability of extinction of a species in a community. I'm simply calculating similarity as the proportion of non-ignore interactions that are the same between each species pair. Doing this straight-away speeds up simulations later on. The similarity function is at `Lego_Program/src/sim_func.jl`

Community assembly

The idea here is to construct a community from the master interaction matrix. First, a primary producer that does not have additional species/objects that it needs is selected as a founding species (plus any objects that it makes). At the minimum Row/Column 2 always fulfills these requirements. The founding species is selected using the function `Lego_Program/src/initiate_comm_func.jl`

Assimilate and Need thresholds: We establish two threshold conditions that must be met for colonizations that occur after the founding species is chosen. The assimilate threshold `a_thresh` is the less selective of the two. The need threshold `n_thresh` is a higher threshold that needs to be passed for a colonization to be successful.

The colonization function, which is found in `Lego_Program/src/colonize_func.jl` works as follows:

1) Randomly choose a species that is not present in the community, but has at least one trophic interaction with something that is present (or is a primary producer)
2) Does the number of realized trophic interactions pass `a_thresh`? So if `a_thresh=0.1`, then at least 10% of the colonizer's prey (from the master interaction matrix) must be present in the assembling community for colonization to be successful.
3) Does the number of needed species/objects pass `n_thresh`? So if `n_thresh=0.5`, then at least 50% of the species/objects that the colonizer needs (from the master interaction matrix) must be present in the assembling community for colonization to be successful.
4) If the randomly chosen colonizer passes these two tests, it is allowed to enter the community.

In the assembly simulation, the colonization function is run at each timestep with a set probability.

Extinctions

Primary extinctions: Extinctions are evaluated at every timestep, and are a function of the assembling community (rather than by a set probability). The extinction function is found at `Lego_Program/src/extinct_func.jl` Currently, I am calculating the probability of extinction based on two characteristics for each species in the assembling community:

1) Predation load: extinction probability increases with the number of predators consuming a given species
2) Competetive load: extinction probability increases with the maximum similarity between a given species and all of the rest

in the assembling community. Similarity is calculated as the proportion of non-ignore interactions that are the same between each species pair.

The probability of extinction `prext` is determined for each species in the system at each time step. Then extinction is determined by drawing from a binomial distribution independently for each species. These are the primary extinctions. Given that the probability of extinction increases as predative and competitive loads increase, it naturally serves to regulate the size of the community as it assembles.

Secondary extinctions: After the primary extinctions are determined, those species and the unique objects that they make are removed from the community. We then determine whether the remaining species pass `a_thresh` and `n_thresh` as we did in the colonization step. If there are species that do not pass, they secondarily go extinct and are removed - after which we must test the threshold tests again and do so until all remaining species pass the test. This dynamic allows for extinction cascades that are internally initiated by predation and competition load, which changes as the community grows/shrinks over time.

Community assembly simulations

The community assembly process, which combines colonizations and extinctions over time are at `Lego_Program/Anime_comsim.jl` There are 3 versions of the assembly process: $>$ 1) Only colonizations are considered and the community grows until no other species 'fit' $>$ 2) Where both colonizations and extinctions occur over time $>$ 3) A parallelized version of the colonization/extinction dynamic with repetitions. The repetitions are parallelized, and to use `n` cores on a given machine, `julia` must be opened from the terminal as `julia -p n`

I have some very basic analysis code to look at results of simulations, but I've only just started playing around with this. I have been spending most of my time just making sure the functions work as they are supposed to.

Where to go next

Lots of ideas... one primary one is to build multiple communities on a spatial network, where species can migrate to/from nearby communities. We can then see if biomes form by looking at community similarity over space (pattern formation), look at the effects of large perturbations (untargeted/targeted extinction events), etc.

Another idea is to incorporate the idea of macro-evolution, where an evolutionary event would alter the interactions between species in the *master interaction matrix* over time.

Other stuff.