Fitting the ZSM

Fitting the ZSM is a complex and computationally intensive process. I developed a series of computer programs written in C and MATLAB to implement the algorithms described in UNTB (i.e. the book *The Unified Neutral Theory of Biogeography* by Stephen Hubbell published by Princeton University Press 2001) on pp. 289-293. The source code for these algorithms are available from the author (see below). A number of alternative implementations will be found commented out in the code. In each case, I chose the implementation which maximized the fit of the ZSM to the data. A C program was written to implement the algorithm pictured in Figure 9.1 of UNTB as well as the local population processes that occur once the metacommunity distribution is calculated (i.e. the equations on p. 86). This process was executed millions of times and well over 95% of the computational time was spent in this routine. Hence, it was coded in C and highly optimized for performance. I will hereafter call this code the "generator" routine. The code for the generator is found in the file "hubbell.c". Two issues of note were found in the implementation of the generator. First, I implemented a method that is identical in results to Figure 9.1 but which trades performance for memory (i.e. runs much faster but uses more memory). Readers who are interested should examine the source code. Even with this modification, estimating the ZSM parameters is extremely computationally intensive: on a Pentium 500 MHz machine it takes 20-30 minutes to estimate a single ZSM fit. This contrasts with the ability to fit 6 other distributions (including two that use Newton-Raphson iterated methods) for 1000 BBS routes in roughly the same amount of time. Secondly, nowhere could I find a mention of how long the equations on p. 86 must be iterated before the metacommunity abundances converge to the stable local community abundance distribution. The answer to this question raises several interesting scientific questions, which I address in the main text (See figure 1 and discussion thereof in the main text).

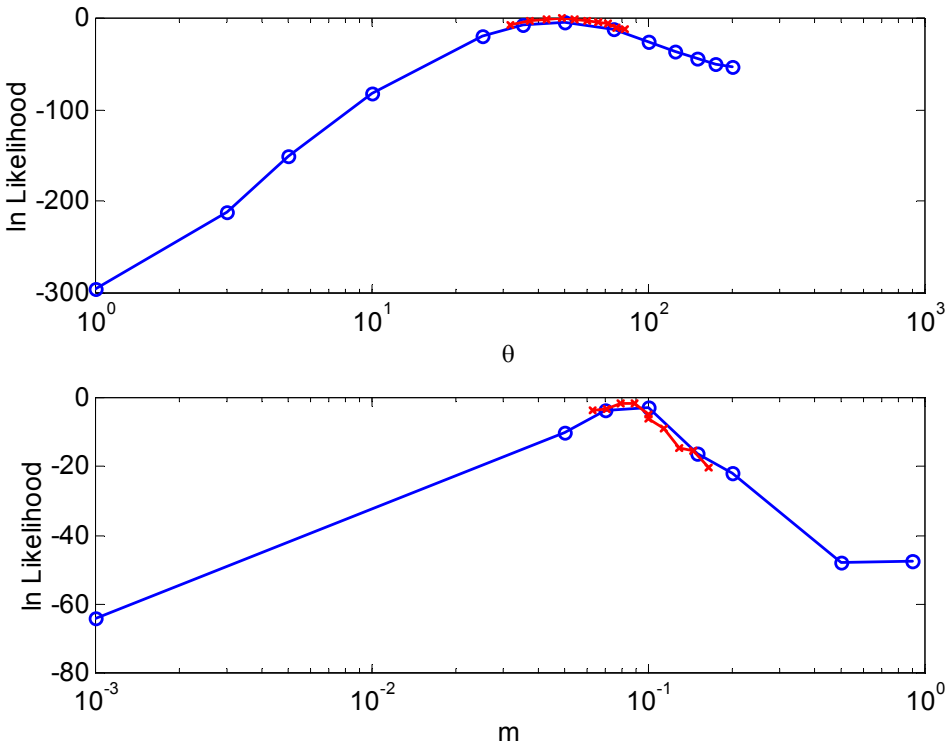
The basic fitting process is:

1. Loop over "all" possible values of θ and m
   a. For each pair of values loop NREPS times
      i. Call the generator
   b. Average the rank abundances produced by the generator across the NREPS repetitions
   c. Evaluate the likelihood (equation on p.292) of the observed data vs. the expected (averaged) data
2. Choose θ and m which correspond to the maximum (largest) likelihood value.

In practice, the loop in step 1 as described is impossible (there are an infinite number of values). UNTB suggests that θ can be fit first and then m can be fit, which I followed. I implemented a hierarchical search as follows. First, the computer would search over 13 values of θ that covered the range of possible values (1-200). The computer would pick the value that had the best likelihood and then search a second time over a range of values near this best fit (covering a small subset of the range of θ from 0.1 to 300). The value of θ which produced the highest likelihood was selected and the computer repeated this two level hierarchical search process to find a very good value of

m (over the range 0-1). See below for an example plot of the resulting likelihood surfaces calculated. Reassuringly, they are fairly smooth and fairly flat near the optimum, although it is discomforting that there remains some amount of noise in the second iteration for m. While this discrete approach cannot find the absolute best fit, it can get close enough not to matter (the likelihood surface is reasonably flat close to the optimal value). No method could find the best fit unless derivatives of the likelihood surface could be taken, something that I deemed hopelessly computationally intensive. The code that implements this algorithm is written in MATLAB and found in the file "esthub.m". It should also be noted that the formula for the likelihood found on p. 292 is undefined if either n_r or $E_r(\Psi)$ is zero. In theory $E_r(\Psi)$ should never be zero, but, in practice, it is only approximated by averaging across NREPS samples, and hence can be zero. The approach to zeros in the data has a large impact on the quality of the fit. After trying several approaches, the one I adopted was to ignore the first term of the likelihood for those abundance bins where n_r was zero since the limit as n_r approaches zero of $n_r \log(E_r(\Psi)/n_r)$ is zero. For those bins where $E_r(\Psi)$ was zero and n_r was not zero I substituted the value one (1.0), which penalized these cases, but avoided drastic deformations of the likelihood surface.

It should be noted that this algorithm is in fact a probabilistic algorithm. It converges to the best solution only as NREPS approaches infinity. This is, of course, computationally impossible. UNTB suggests using NREPS=100 (p. 290), so I followed this. For the most part it worked very well (see the figure in the main article for an example of the fit to BCI data). However, the results were probabilistic. Rerunning the fitting algorithm with only a different starting random seed would result in a different fit. This explains the noise observed in the 2nd iteration of the likelihood surface shown in the figure below. However, very often the fit was worse (a majority of the cases in a random sample that I reran) or if it was an improvement, it was very small. Most of the worst fits of the ZSM were due to variance in the data, not fitting algorithms. Nonetheless, as a further precaution against this problem, I also examined just the 50 cases where the ZSM performed best. It did not noticeably change the results, affecting on average the various measures of fit only in the 2nd or 3rd significant digit and not substantially changing the most important statistic in this paper (percentage of cases where the ZSM performs better than the Lognormal). Thus, even, this bias in favor of the ZSM does not qualitatively change the results, and I do not present these results in detail elsewhere in the paper.



Likelihood surface for fitting the ZSM multinomial The top subfigure represents ln Likelihood for fitting 'θ', the fundamental biodiversity number. The bottom subfigure represents ln Likelihood for fitting 'm', the percent of births that are emigrations from the metacommunity. In each figure, the blue line represents a first pass calculation of the surface over a wide range of values. The red line with 'x' markers represents a second pass calculation at a finer scale in the area of greatest likelihood identified by the first pass. In this case, the data is for the BCI dataset.

Source code

Three computer programs are available from the author at http://www.brianmcgill.org/zsmcode.html. These are:

File name	Prog. Language	Description
hubbell.c	C	The "generator" routine as described in the methods above. Creates one random sample of a community (# of species and their abundances) based on the input parameters, θ, J_m, m, and t. m is an optional parameter if the community is local. If m is 1 or not provided, a metacommunity is simulated. t indicates how many generations to run in the local community.
esthub.m	Matlab	Given a set of species abundances as inputs, estimates the maximum likelihood estimates of θ and m.
myhubeng.m	Matlab	Calls the generator routine (hubbell.c) NREPS (usually 100) times and averages to produce the mean and standard error for a ZSM SAD for a given set of parameters.