Unit Tests

Note: The GetSafeID() function was taken directly from the heterogeneity model and therefore has already been tested. Also, the rStudKern() function has been validated based on Allison’s code accomplishing the same thing.

1. SaveParams()
   1. Given a list of input parameters, check that a fully executable R file is produced that recreates all the parameters
2. ChangeClimate()
   1. Given a starting beta value and climate speed, calculate by hand the vector that should be produced and ensure the function produces it correctly
   2. If the length of climate shift is negative, make sure the function returns a helpful error message
   3. If the length of climate shift is 0, make sure the function returns an empty vector
3. PopMatColNames()
   1. Make sure the example functionality works
   2. Try different combinations of number of alleles and monoecious or not and make sure it returns the desired results
4. Initialize()
   1. Use the PopMatColNames() function to generate the ColumnNames, FitCols, and DispCols inputs for this function.
   2. Ensure that the returned matrix is of the correct dimensions according to the PopSize and ColumnNames argument
   3. Ensure that starting locations are all with x = BetaInit and y a random sample of 1 through the width of the landscape
   4. Ensure that altered sex ratios do indeed change the obtained sex ratios
   5. Ensure that initiated trait values produce approximately normal distributions of the correct mean and standard deviations
5. GenEnvQual() and CalcEnvMean()
   1. Give the CalcEnvMean() function a series of values on either side of the center value for the range. The function is by definition symmetric, so these should also be symmetric for equivalent ranges on either side of the center.
   2. Ensure that the GetEnvQual() function is capable of working with different patch scales. Larger scales should be the average of smaller scales.
6. CalcTraits()
   1. First, using a sample population made by the Initialize() function, make sure CalcTraits() returns a matrix of the correct proportions with the right column names.
   2. Use Initialize() to create a population matrix with 1,000 individuals and 100 loci for each trait. Plot histograms of the calculated trait values and make sure they match expected normal distributions. For the fitness trait, the distribution of trait values should match a normal distribution with mean of 200\*FitMean and a standard deviation of sqrt(200\*FitDiv^2) based on the properties of adding normally distributed random variables (the individual alleles). For dispersal, the histogram of logged values should an analogous normal distribution with the relevant dispersal parameters.
7. Disperse()
   1. Use the previously vetted Initialize() and CalcTraits() functions to generate the appropriate inputs
   2. First, set the landscape width to 1 to make it 1 dimensional and generate a large population matrix (1,000 or so) and try dispersal with all three dispersal kernels. The distribution of new x locations should be roughly in line with the three distribution shapes. However since the direction is still being chosen randomly from a uniform distribution, this actually might not be the case (e.g. a long distance could be paired with an angle perpendicular to the x axis, resulting in 0 displacement). There should still probably be differences between the three kernels…
   3. Next, set the width to 5 and pop size to 5,000 (to keep an average of the same number of individuals in each row) and repeat the same process. Visualize the displacements with a heat map and compare among kernels. Theoretically the normal and student’s t kernels should allow for more long distance dispersal.
   4. Repeat the above process for a width of 10 or 20 to see how landscape width affects the spread of individuals in the x direction. It should have no effect given the wrapping boundaries and random selection of direction.
8. RelFit()
   1. Use the Initialize() function to create a large population all at x = beta and trait values centered around 0.5 (which should be the optimum there regardless of any other parameters). Plug that into the RelFit function and plot a histogram of the fitness values. The distribution should have 1 as the largest value and have decreasing frequencies of lower values and no values above 1.
   2. Do the same thing, but with a larger variance in initial values. The histogram pattern should be exaggerated from before with more lower values, possibly a higher peak above the 1 values, but still no greater than 1 values
   3. For a given range, calculate by hand the optimum phenotype values from the equation, construct a simple population matrix and a traits matrix for a small number of individuals spread across the range at equal intervals. Use the RelFit() function to calculate fitness and plot the results. All individuals should have a value of 1.
9. Inheritence()
   1. Using manually created input matrices, test 0 and 0.5 for U and 0.1 and 1 (factorially) for U and Vm to ensure that mutation is working as I expect it to.
   2. Set mutation components to 0 and manually make a matrix with each allele value being completely unique and use the output to ensure that segregation is happening as expected and independently.
10. MatFill()
    1. Make sure output only includes occupied patches
    2. Make sure each patch has the designated number of offspring
    3. Try patch combinations with the same x or y coordinates to make sure that is all working appropriately
    4. Do a monoecious example to test that selfing is only occurring as a last resort
11. Reproduce()
    1. Make sure the error checks work.
    2. Check for zero growth in cases where there shouldn’t be any growth
    3. Check for reduced growth rates in the face of deteriorating local adaptation.

Tests of the full model

1. Populations with no dispersal don’t move but reach carrying capacity and are relatively stable there
2. Populations with good fitness and dispersal fill a stationary range but don’t move beyond it
3. Populations with no dispersal but good fitness, react to a moving environment, eventually going extinct

Format of the model testing documents

Each individual function being tested will have an associated R script to run the tests described here. Similarly, the tests of the full model will each have a single, associated R script. Each R script will result in a single text file consisting of a single word (“pass” or “fail”) as a record of the outcome of the test. Finally, there will be a single Rmarkdown document consisting of a table with a short description of the test and the associated outcome.