**Scripts unique to this study.**

**split-bidirectional-migration δaδi** **model**

# splitbidirmig script in Demographics2Dmod.py

def split\_bidirmig(params, ns, pts):

"""

params = (nu1,nu2,T,m12,m21)

ns = (n1,n2)

Split into two populations of specified size, with bidirectional migration.

nu1: Size of population 1 after split.

nu2: Size of population 2 after split.

T: Time in the past of split (in units of 2\*Na generations)

m12: Migration rate 2>>1

m21: Migration rate 1>>2

n1,n2: Sample sizes of resulting Spectrum

pts: Number of grid points to use in integration.

"""

# modify params to include bidirectional gene flow

nu1,nu2,T,m12,m21 = params

# use grid

xx = Numerics.default\_grid(pts)

# use the grid and split pops

phi = PhiManip.phi\_1D(xx)

phi = PhiManip.phi\_1D\_to\_2D(xx, phi)

# allow for gene flow between the pops bidirectional m12,m21

phi = Integration.two\_pops(phi, xx, T, nu1, nu2, m12=m12, m21=m21)

#calculate and return the spectra

fs = Spectrum.from\_phi(phi, ns, (xx,xx))

return fs

**The Python script used for the best-fit split-migration δaδi model, with associated parameters:**

# The basis for this script is from Kevin Hawkins, 5 Feb 2016. I added in a few comments.

# Numpy is the numerical library dadi is built upon from numpy import array

import dadi

# import the demographic model.

import Demographics2D

# Load the data

data = dadi.Spectrum.from\_file('buntings-UCEs-no-Zs-folded-spectrum.fs')

ns = data.sample\_sizes

data.mask\_corners()

# print number of samples to verify correct load

print ns

# Grid point settings will be used for extrapolation.

# Grid points need to be formated [n,n+10,n+20].

# Needs to be bigger than the number of samples you have (n>ns) and this will be a strong

# determination as to how long your program will run.

pts\_l = [50,60,70]

# Call particular model to run, the model choosen here is split.w.migration

func = dadi.Demographics2D.split\_mig

# params = (nu1,nu2,T,m)

# ns = (n1,n2)

#

# Split into two populations of specifed size, with migration.

# nu1: Size of population 1 after split.

# nu2: Size of population 2 after split.

# T: Time in the past of split (in units of 2\*Na generations)

# m: Migration rate between populations (2\*Na\*m)

# n1,n2: Sample sizes of resulting Spectrum

# pts: Number of grid points to use in integration.

# Now let's optimize parameters for this model.

# The upper\_bound and lower\_bound lists are for use in optimization.

# Occasionally the optimizer will try wacky parameter values. We in particular

# want to exclude values with very long times, very small population sizes, or

# very high migration rates, as they will take a long time to evaluate.

# Parameters are: (nu1,nu2,T,m)

#Set the upper and lower bounds to make sure that the boundaries are

#there. Suggested time parameters: lower 0, upper 5, migration

#parameters: lower 0, upper 10, size parameters: lower 1e-2, upper 100

upper\_bound = [6, 7, 2, 2]

lower\_bound = [1e-1, 1e-1, 0, 0]

# This is our initial guess for the parameters, which is somewhat arbitrary.

p0 = [1,1,1,1]

# Make the extrapolating version of our demographic model function.

func\_ex = dadi.Numerics.make\_extrap\_log\_func(func)

# Perturb our parameters before optimization. This does so by taking each

# parameter up to a factor of two up or down.

p0 = dadi.Misc.perturb\_params(p0, fold=2, upper\_bound=upper\_bound,

lower\_bound=lower\_bound)

print('Beginning optimization \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

popt = dadi.Inference.optimize\_log(p0, data, func\_ex, pts\_l,

lower\_bound=lower\_bound,

upper\_bound=upper\_bound,

verbose=2)

# The verbose argument controls how often progress of the optimizer should be

# printed. It's useful to keep track of optimization process.

print('Finished optimization \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

print('Best-fit parameters: {0}'.format(popt))

# Calculate the best-fit model AFS.

model = func\_ex(popt, ns, pts\_l)

# Likelihood of the data given the model AFS.

ll\_model = dadi.Inference.ll\_multinom(model, data)

print('Maximum log composite likelihood: {0}'.format(ll\_model))

# The optimal value of theta given the model.

theta = dadi.Inference.optimal\_sfs\_scaling(model, data)

print('Optimal value of theta: {0}'.format(theta))

print pts\_l, upper\_bound, lower\_bound

# Plot a comparison of the resulting fs with the data.

import pylab

pylab.figure(1)

dadi.Plotting.plot\_2d\_comp\_multinom(model, data, vmin=0.1, resid\_range=1,

pop\_ids =('hyperboreus','nivalis'))

# This ensures that the figure pops up. It may be unnecessary if you are using

# ipython.

pylab.show()