Supplemental methods for Beckman, Martin *et al.*

**Table of Contents:**

Methods for neutral simulations ... 2

Figure of models 1 and 2 ... 4

Literature cited ... 5

Code ... 6

 SLiM code for Model 1 (constant Ne): 6

 SLiM code for Model 2 (bottlenecks): 8

 Add neutral mutations to genealogies: 10

 Run SLiM simulations in serial: ... 13

 Calculate PBS*n1* from FST: ... 15

**Methods for Neutral Simulations**

We simulated a neutral, bifurcating divergence history with no migration between three populations to establish a false discovery rate (FDR) for the PBS*n1* analysis. Elucidating the demography of house mouse colonization in South America would require much broader geographic sampling and is beyond the scope of this paper; however, we considered two null models (presented below) based on simple demographic scenarios. We intended these to provide a context in which to interpret the PBS*n1* selection scan results. For simplicity, we focused on the northern transect PBS*n1* analysis; we used nucleotide diversity (π) and FST from the following populations to inform our models: <90 m in the southern transect, <330 m in the northern transect and > 2800 m in the northern transect. We used SLiM (v 3.6, Haller and Messer 2019), pyslim (v 0.600, Haller *et al*. 2019), and msprime (v 1.0.2, Kelleher, Etheridge and McVean 2016) for the simulations. We defined the mutation rate as 6 x 10-9 mutations/base pair/generation (Milholland *et al.* 2017), the recombination rate as 5.984 x 10-9 per base pair per generation (Cox *et al.* 2009) and the generation time as 1 generation per year (Geraldes *et al.* 2008). We simulated a one megabase region per iteration. In the models, the ancestral South American house mouse population was split into north and south populations ~480 generations ago, based on the founding of the cities of La Paz and Quito by Europeans; high and low populations in the northern transect diverged after 100 generations. In Model 1, we based effective population size (Ne) for each population on the observed π and it remained constant once a population was established. In Model 2, we imposed bottlenecks of varying severity and duration at each colonization event to approximate FST between the highest and lowest northern populations (<330 m and > 2800 m) and π for each population. For each model, we generated 1000 simulations, calculated PBS*n1* as described in the methods and computed the proportion of neutral SNPs per simulation that surpassed the actual threshold set in the PBS*n1* northern transect analysis. To get the PBS*n1* false discovery rate, we averaged this proportion across all simulations.

**A.**

**B.**

**Neutral simulation Model 1 (A) and Model 2 (B) for PBS*n1* analysis.** In model 1, effective population size (Ne) was calculated from intronic π. In model 2, we used bottlenecks (*eg.* Ne = 100) to more closely approximate intronic π and FST between the northern transect populations.

**Literature cited:**

Cox, A., Ackert-Bicknell, C., Dumont, B., Ding, Y., Bell, J. *et al*. (2009). A new standard genetic map for the laboratory mouse. *Genetics*. **182**: 1335-1344.

<https://doi.org/10.1534/genetics.109.105486>

Geraldes, A., Basset, P., Gibson, B., Smith, K. L., Harr, B. *et al*. (2008). Inferring the history of speciation in house mice from autosomal, X‐linked, Y‐linked and mitochondrial genes. *Molecular ecology*. **17**: 5349-5363. <https://doi.org/10.1111/j.1365-294X.2008.04005.x>

Haller, B. C., Galloway, J., Kelleher, J., Messer, P. W., & Ralph, P. L. (2019). Tree‐sequence recording in SLiM opens new horizons for forward‐time simulation of whole genomes. *Molecular ecology resources*. **19**: 552-566. <https://doi.org/10.1111/1755-0998.12968>

Haller, B. and Messer, P. (2019). SLiM 3: Forward genetic simulations beyond the Wright–Fisher model. *Molecular biology and evolution*. **36**: 632-637. <https://doi.org/10.1093/molbev/msy228>

Kelleher, J., Etheridge, A. M. and McVean, G. (2016). Efficient Coalescent Simulation and Genealogical Analysis for Large Sample Sizes*. PLOS Comput Biol*. **12**: e1004842. <https://doi.org/10.1371/journal.pcbi.1004842>

Milholland, B., Dong, X., Zhang, L., Hao, X., Suh, Y. *et al.* (2017). Differences between germline and somatic mutation rates in humans and mice. *Nat Commun.* **8:** 15183. <https://doi.org/10.1038/ncomms15183>

**Code**

**SLiM code for Model 1 (constant Ne):**

// Neutral simulation of basic demography of M. m. domesticus in South America

// Simulations will be used to establish PBSn1 false discovery rate based on Northern transect

// Simple bifurcating history with no migration, u and r extracted from literature, Ne based on empirical estimates of intronic pi

// Elizabeth J. Beckman

// libby.beckman@berkeley.edu

// 03 September 2021

// Model 1, 1 Mb (aka 19)

initialize() {

 // initalize tree sequence recording

 initializeTreeSeq();

 // define mutation rate

 initializeMutationRate(0);

 // m1 mutation type: neutral

 initializeMutationType("m1", 0.5, "f", 0.0);

 // g1 genomic element type: uses m1 for all mutations

 initializeGenomicElementType("g1", m1, 1.0);

 // uniform chromosome of length 1 Mb with uniform recombination

 initializeGenomicElement(g1, 0, 999999);

 initializeRecombinationRate(5.98e-9);

}

// create a population of ~50000 individuals, average of two lowest SA populations

// Ancestral pop of South America, given empirical pi

// total simulation time of 580 generations

1 {

 sim.addSubpop("p1", 56542);

}

// Split between North population (p2) and South (p1) lowland population

// Ancestor of North is average of low North populations

// South p1 is resized to empirical South lowland Ne

// 480 generations from present, founding of Quito

100 { sim.addSubpopSplit("p2",asInteger(round(55437.50)),p1);

 p1.setSubpopulationSize(asInteger(round(58583.33)));

}

// Split between North high and low populations

// Ne set by empirical pi in both populations

// Longer separation time of high and low gets closer to empirical Ne for high elevation

200 { sim.addSubpopSplit("p3",asInteger(round(33583.33)),p2);

 p2.setSubpopulationSize(asInteger(round(54500.00)));

}

// Downstream steps expect an outfile in the format of: Model#\_size of genomic element.trees

// eg. Model1\_1Mb.trees

580 late() {

 // Output .trees can be read into python using tskit package

 sim.treeSeqOutput("<outfile name>.trees");

}

**SLiM code for Model 2 (bottlenecks):**

// Neutral simulation of basic demography of M. m. domesticus in South America

// Simulations will be used to establish PBSn1 false discovery rate based on northern transect

// Simple bifurcating history with no migration, u and r extracted from literature, Ne based on empirical estimates of intronic pi

// Bottlenecks of varying severity and duration imposed when a new subpopulation is started to approximate genetic diversity and differentiation observed in the real data

// Elizabeth J. Beckman

// libby.beckman@berkeley.edu

// 03 September 2021

// Model 2, 1 Mb

// set up a simple neutral simulation

// Model 2 (aka 14)

initialize() {

 // initalize tree sequence recording

 initializeTreeSeq();

 // define mutation rate

 initializeMutationRate(0);

 // m1 mutation type: neutral

 initializeMutationType("m1", 0.5, "f", 0.0);

 // g1 genomic element type: uses m1 for all mutations

 initializeGenomicElementType("g1", m1, 1.0);

 // uniform chromosome of length 100 kb with uniform recombination

 // simulation 1 Mb

 initializeGenomicElement(g1, 0, 999999);

 initializeRecombinationRate(5.98e-9);

}

// create a population of 56542 individuals

// Ancestral pop of lowest populations in South America, given empirical pi

// total simulation time of 580 generations

// no bottleneck

1 {

 sim.addSubpop("p1", 56542);

}

// Split between North population (p2) and South (p1) lowland population

// 480 generations from present

// 50 generations bottleneck for each budding population (N and S)

100 { sim.addSubpopSplit("p2",1000,p1);

 p1.setSubpopulationSize(1000);

}

// Population growth

150:168 { newSizep1 = asInteger(round(1.25^(sim.generation - 149) \* 1000));

 newSizep2 = asInteger(round(1.25^(sim.generation - 149) \* 1000));

 p1.setSubpopulationSize(newSizep1);

 p2.setSubpopulationSize(newSizep2);

}

// p1 to pi of South <90 m

// p2 to average pi of two low populations in northern transect

169 { p1.setSubpopulationSize(asInteger(round(58583.33)));

 p2.setSubpopulationSize(asInteger(round(55437.50)));

}

// Split between northern high and low populations

// 380 generations from present

// 140 generations of bottleneck for high elevation population

200 { sim.addSubpopSplit("p3",100,p2);

 p2.setSubpopulationSize(asInteger(round(54500.00)));

}

// Population growth

340:366 { newSizep3 = asInteger(round(1.25^(sim.generation - 339) \* 100.00));

 p3.setSubpopulationSize(newSizep3);

}

367 { p3.setSubpopulationSize(asInteger(round(33583.33)));

}

// Downstream steps expect an outfile in the format of: Model#\_size of genomic element.trees

// eg. Model2\_1Mb.trees

580 late() {

 // Output .trees can be read into python using tskit package

 sim.treeSeqOutput("<outfile name>.trees");

}

**Add neutral mutations to genealogies:**

#! /usr/bin/env python3

# The purpose of this script is to recapitate the genealogies generated through SLiM and

# apply neutral mutations to the trees using mutation rate and recombination rate

# Input: Model2\_1Mb\_run${i}.trees generated in SLiM

# Output: Model2\_1Mb\_run${i}.vcf with neutral mutations

# Model2\_1Mb\_run${i}.p1.txt the list of individuals sampled for p1

# Model2\_1Mb\_run${i}.p2.txt as above for p2

# Model2\_1Mb\_run${i}.p3.txt as above for p3

# Elizabeth J. Beckman

# libby.beckman@berkeley.edu

# 24 August 2021

import msprime

import pyslim, tskit

import numpy as np

import sys

# Specify model

MOD = str(sys.argv[1])

# Specify length of simulation sequence

LEN = str(sys.argv[2])

# Specify run

RUN = str(sys.argv[3])

InFile1 = "<path to folder>/Model%s/Model%s\_%s\_run%s.trees" %(MOD,MOD,LEN,RUN)

# Load SLiM run

ts = pyslim.load(InFile1)

# Recapitate .trees and overlay mutations

rts = ts.recapitate(Ne=50000, recombination\_rate=5.984213e-09)

mrts = pyslim.SlimTreeSequence(msprime.mutate(rts, rate=6.00e-09))

# Calculate nucleotide diversity for each population

# Modified for four populations (empty, p1, p2, p3)

# pop\_indivs appears to be a list of four items, where each item includes the individual ID if in population 0, 1, 2 etc

pop\_indivs = [[], [], [], []]

pop\_nodes = [[], [], [], []]

for i in mrts.individuals\_alive\_at(0):

 ind = mrts.individual(i)

 pop\_indivs[ind.population].append(i)

 pop\_nodes[ind.population].extend(ind.nodes)

diversity = mrts.diversity(pop\_nodes[1:])

#print(f"There are {mrts.num\_mutations} mutations across {mrts.num\_trees} distinct "

# f"genealogical trees describing relationships among {mrts.num\_samples} "

# f"sampled genomes, with a mean genetic diversity of {diversity[0]}, "

# f"{diversity[1]} and {diversity[2]} within the three populations.")

# Subsample 10 individuals per population, randomly sample 10 individuals that are alive (see above)

sample\_inds = [np.random.choice(pop\_indivs[i], 10, replace=False) for i in [1,2,3]]

# Desired output will be (1) a separate list with the individuals for each population

# A VCF file with the vcf\_label

indivlist = []

indivnames = []

Ind\_in\_p1 = []

for i in sample\_inds[0]:

 indivlist.append(i) # add the ind\_ID to indivlist

 ind = mrts.individual(i) # specify the individual metadata for ind\_ID

 vcf\_label = f"tsk\_{ind.id}" # create the vcf\_label with the ind.id from the metadata

 indivnames.append(vcf\_label) # add the vcf\_label with ind.id to the indivnames list

 Ind\_in\_p1.append(vcf\_label) # make mini list of individuals within population p1

Ind\_in\_p2 = []

for i in sample\_inds[1]:

 indivlist.append(i) # add the ind\_ID to indivlist

 ind = mrts.individual(i) # specify the individual metadata for ind\_ID

 vcf\_label = f"tsk\_{ind.id}" # create the vcf\_label with the ind.id from the metadata

 indivnames.append(vcf\_label) # add the vcf\_label with ind.id to the indivnames list

 Ind\_in\_p2.append(vcf\_label) # make mini list of individuals within population p2

Ind\_in\_p3 = []

for i in sample\_inds[2]:

 indivlist.append(i) # add the ind\_ID to indivlist

 ind = mrts.individual(i) # specify the individual metadata for ind\_ID

 vcf\_label = f"tsk\_{ind.id}" # create the vcf\_label with the ind.id from the metadata

 indivnames.append(vcf\_label) # add the vcf\_label with ind.id to the indivnames list

 Ind\_in\_p3.append(vcf\_label) # make mini list of individuals within population p3

Outfile1="<path to folder>/Model%s/Model%s\_%s\_run%s.vcf" %(MOD,MOD,LEN,RUN)

with open(Outfile1, "w") as vcffile:

 mrts.write\_vcf(vcffile, individuals=indivlist, individual\_names=indivnames) # indivnames is the vcf\_label

# print individuals for p1

Outfile2="<path to folder>/Model%s/Model%s\_%s\_run%s.p1.txt" %(MOD,MOD,LEN,RUN)

with open(Outfile2,"w") as p1\_ind:

 for each in Ind\_in\_p1:

 p1\_ind.write(each + "\n")

# print individuals for p2

Outfile3="<path to folder>/Model%s/Model%s\_%s\_run%s.p2.txt" %(MOD,MOD,LEN,RUN)

with open(Outfile3,"w") as p2\_ind:

 for each in Ind\_in\_p2:

 p2\_ind.write(each + "\n")

# print individuals for p3

Outfile4="<path to folder>/Model%s/Model%s\_%s\_run%s.p3.txt" %(MOD,MOD,LEN,RUN)

with open(Outfile4,"w") as p3\_ind:

 for each in Ind\_in\_p3:

 p3\_ind.write(each + "\n")

Outfile5="<path to folder>/Model%s/Model%s\_%s\_pi.txt" %(MOD,MOD,LEN)

with open(Outfile5,"a") as Div:

 Div.write(str(diversity[0]) + "\t" + str(diversity[1]) + "\t" + str(diversity[2]) + "\n")

**Run SLiM simuations in serial:**

#! /bin/bash

# To run a simulation in SLiM, recapitate and mutate in python3, and calculate PBSn1 using vcftools and the custom PBSn1 script

# Elizabeth J. Beckman

# libby.beckman@berkeley.edu

# 03 September 2021

# Model 2, 1 Mb

# Specify model number

MOD="19"

# Specify length of simulation sequence

LEN="1Mb"

DATE="2021.09.03"

cd <path to folder>

mkdir Model${MOD}

cd <path to folder>/Model${MOD}/

for i in {1..1000}

do

 # Step 1, run slim simulation

 echo ${i}

 slim <path to scripts>/Model${MOD}\_${LEN}.${DATE}.slim

 mv Model${MOD}\_${LEN}.trees Model${MOD}\_${LEN}\_run${i}.trees

 # Step 2 and 3, use pyslim and msprime to apply mutations and make VCF file

 python3 <path to scripts>/Model.recap.mutate.py $MOD $LEN $i

 # Step 4, filter for MAF>0.05, calculate pairwise per site Fst using VCFtools

 vcftools --vcf Model${MOD}\_${LEN}\_run${i}.vcf --maf 0.05 --recode --out Model${MOD}\_${LEN}\_run${i}.maf.05

 # Calculate pairwise Fst between three populations

 vcftools --vcf Model${MOD}\_${LEN}\_run${i}.maf.05.recode.vcf --weir-fst-pop Model${MOD}\_${LEN}\_run${i}.p1.txt --weir-fst-pop Model${MOD}\_${LEN}\_run${i}.p2.txt --out Model${MOD}\_${LEN}\_run${i}.p1p2

 vcftools --vcf Model${MOD}\_${LEN}\_run${i}.maf.05.recode.vcf --weir-fst-pop Model${MOD}\_${LEN}\_run${i}.p1.txt --weir-fst-pop Model${MOD}\_${LEN}\_run${i}.p3.txt --out Model${MOD}\_${LEN}\_run${i}.p1p3

 vcftools --vcf Model${MOD}\_${LEN}\_run${i}.maf.05.recode.vcf --weir-fst-pop Model${MOD}\_${LEN}\_run${i}.p2.txt --weir-fst-pop Model${MOD}\_${LEN}\_run${i}.p3.txt --out Model${MOD}\_${LEN}\_run${i}.p2p3

 # Step 5, calculate PBSn1 from Fst

 python3 <path to scripts>/PBSn1\_CalculatefromVCFtools\_SLiM.py $MOD $LEN $i

 # Step 6, calculate how many sites in the run are above the empirical threshold for the northern transect

 # Output has format:

 # PBSn1\_T is in column 9

 # Empiric threshold from northern transect top 1 percent of PBSn1 = 0.7160947

 # Count sites above empiric threshold

 var=$(awk '$9>0.7160947' Model${MOD}\_${LEN}\_run${i}.PBSn1.txt | wc -l)

 var=$(echo $var)

 nb\_lines=${var%% \*}

 # Count total number of lines

 var2=$(wc -l < Model${MOD}\_${LEN}\_run${i}.PBSn1.txt)

 var2=$(echo $var2)

 tl\_lines=${var2%% \*}

 proportion=$(echo "scale=4; $nb\_lines / $tl\_lines" | bc)

 printf "%s\t%s\t%s\t0%s\n" "$i" "$nb\_lines" "$tl\_lines" "$proportion" >> Model${MOD}\_${LEN}.PBSn1.Simulation.0.716.threshold.txt

 rm \*.weir.fst

 rm Model${MOD}\_${LEN}\_run${i}.maf.05.recode.vcf

 rm Model${MOD}\_${LEN}\_run${i}.trees

done

**Calculate PBS*n1* from FST:**

#! /usr/bin/env python3

# Script is to calculate PBSn1 for SLiM simulations

# per site Fst for sites with MAF>0.05 was calculated in VCFtools

# Elizabeth J. Beckman

# libby.beckman@berkeley.edu

# 24 August 2021

import numpy as np

import sys

# Specify model

MOD = str(sys.argv[1])

# Specify length of simulation sequence

LEN = str(sys.argv[2])

# Specify run

RUN = str(sys.argv[3])

# Script to calculate PBSn1 for set of populations:

# The infile from VCFtools includes CHROM POS WEIR\_AND\_COCKERHAM\_FST

# with nan and negative numbers possible

# Take nan to 0, negative numbers to 0

# In SLiM simulations, p3 is the Target population

Folder = "<path to folder>/Model%s/" %(MOD)

Target\_POP1name = "%sModel%s\_%s\_run%s.p1p3.weir.fst" %(Folder,MOD,LEN,RUN)

Target\_POP2name = "%sModel%s\_%s\_run%s.p2p3.weir.fst" %(Folder,MOD,LEN,RUN)

POP1\_POP2name = "%sModel%s\_%s\_run%s.p1p2.weir.fst" %(Folder,MOD,LEN,RUN)

# chr, pos, PBS\_ECU.H, PBSn1\_ECU.H

Outfile1name = "%sModel%s\_%s\_run%s.PBSn1.txt" %(Folder,MOD,LEN,RUN)

Outfile1 = open(Outfile1name,"w")

Data\_TP1 = []

with open(Target\_POP1name) as f:

 next(f)

 for line in f:

 line = line.strip("\r\n")

 line = line.split("\t")

 # Calculate Fst for individual SNP

 if line[2] == "nan":

 Fst = 0

 else:

 if float(line[2])<0:

 Fst = 0

 else:

 Fst = float(line[2])

 Info = [line[0],line[1],Fst]

 Data\_TP1.append(Info)

Data\_TP2 = []

with open(Target\_POP2name) as f:

 next(f)

 for line in f:

 line = line.strip("\r\n")

 line = line.split("\t")

 # Calculate Fst for individual SNP

 # Try taking nan to 0, nan in vcftools probably means there is no variation between the two populations in this case

 # nan are not present in the Angsd output files...

 if line[2] == "nan":

 Fst = 0

 else:

 if float(line[2])<0:

 Fst = 0

 else:

 Fst = float(line[2])

 Info = [line[0],line[1],Fst]

 Data\_TP2.append(Info)

Data\_P1P2 = []

with open(POP1\_POP2name) as f:

 next(f)

 for line in f:

 line = line.strip("\r\n")

 line = line.split("\t")

 # Calculate Fst for individual SNP

 if line[2] == "nan":

 Fst = 0

 else:

 if float(line[2])<0:

 Fst = 0

 else:

 Fst = float(line[2])

 Info = [line[0],line[1],Fst]

 Data\_P1P2.append(Info)

### Since it is possible that there will be extreme values of Fst in the Target population due to the 0.05 MAF

### being applied across all 3 populations together (meaning it could be very rare in ECU)

### Additional option to filter by Ecuador filtered sites:

# /media/nachmanlab/lbeckman/Ecuador/June19/Ecuador.JuneQMAF.txt

### Comment below out if you want to skip

#Transect\_JuneQMAF\_Filter = True

# Transect filter skipped for X chromosome since site frequencies from Angsd are not calculated as correctly anyways

Transect\_Filter = False

#header = ["chr","pos","Fst\_TP1","Fst\_TP2","Fst\_P1P2","PBS\_T","PBS\_P1","PBS\_P2","PBSn1\_T"]

#Outfile1.write("\t".join(header)+"\n")

if Transect\_Filter == True:

 SitesQMAF = []

 SitesFileName = Folder + "Bolivia.NovQMAF.chr" + CHR + ".txt"

 SitesFile = open(SitesFileName,newline="")

 for line in SitesFile:

 line=line.strip("\n\r")

 line=line.split("\t")

 SitesQMAF.append([line[0],line[1]])

Fst\_summary = []

Fst\_num = []

# Find corresponding trios of Fst:

for i in range(0,len(Data\_TP1)):

 SNP\_data = [Data\_TP1[i][0],Data\_TP1[i][1],str(Data\_TP1[i][2])]

 SNP\_num = [Data\_TP1[i][2]]

 if Transect\_Filter == True:

 InTransect = False

 for Site in SitesQMAF:

 if Site[0] == SNP\_data[0] and Site[1] == SNP\_data[1]:

 InTransect = True

 #print("works")

 else:

 pass

 else:

 InTransect = True

 # If InTransect=True, continue to verify SNP data for the 2 additional population comparisons

 if InTransect == True:

 for j in range(0,len(Data\_TP2)):

 if SNP\_data[0] == Data\_TP2[j][0] and SNP\_data[1] == Data\_TP2[j][1]:

 SNP\_data.append(str(Data\_TP2[j][2]))

 SNP\_num.append(Data\_TP2[j][2])

 else:

 pass

 if len(SNP\_data) < 4:

 pass

 else:

 for k in range(0,len(Data\_P1P2)):

 if SNP\_data[0] == Data\_P1P2[k][0] and SNP\_data[1] == Data\_P1P2[k][1]:

 SNP\_data.append(str(Data\_P1P2[k][2]))

 SNP\_num.append(Data\_P1P2[k][2])

 else:

 pass

 # SNP\_data equals chr, pos, Fst\_TP1, Fst\_TP2, Fst\_P1P2

 if len(SNP\_data) == 5:

 #Fst\_summary.append(SNP\_data)

 #Fst\_num.append(SNP\_num)

 # Turn Fsts into an array

 Fst\_Array = np.array(SNP\_num)

 # Transform Fst into T

 T\_Array = -np.log(1-Fst\_Array)

 # Calculate PBS for each population

 PBS\_Tar = (T\_Array[0]+T\_Array[1]-T\_Array[2])/2

 PBS\_P1 = (T\_Array[0]+T\_Array[2]-T\_Array[1])/2

 PBS\_P2 = (T\_Array[1]+T\_Array[2]-T\_Array[0])/2

 # Calculate normalized PBS for Target population

 PBSn1\_Tar = PBS\_Tar/(1+PBS\_Tar+PBS\_P1+PBS\_P2)

 Outfile1.write("\t".join(SNP\_data) + "\t" + str(PBS\_Tar) + "\t" + str(PBS\_P1) + "\t" + str(PBS\_P2) + "\t" + str(PBSn1\_Tar) + "\n")

 else:

 pass

Outfile1.close()