sfs_coder Documentation

Release 0

Lawrence Uricchio, Raul Torres, Ryan Hernandez

June 16, 2014

CONTENTS

CHAPTER

ONE

INTRODUCTION

Forward simulation of DNA sequences is a powerful tool for analyzing the impact of evolutionary forces on genetic variaiton. Forward simulation can generate sequence data under arbitrarily complex models that include nautral selection as well as complex demography. These models are difficult to handle analytically, and other simulation frameworks such as the coalescent cannot provide the same level of generality.

However, there remain some barriers to the adoption of population genetic simulators for many members of the genetics community. Simulation software tools can be daunting to master because of the vast number of input options and the density of the output data.

sfs_coder is a python based front end to the popular forward simulation software SFS_CODE. It allows python coders to easily execute and analyze simulations performed with SFS_CODE. For beginning users and coders, it provides a number of useful cannonical models that are prepackaged and can be accessed and analyzed with just a few lines of code.

For advanced users, sfs_coder provides a tool set to access the power of SFS_CODE through a python based interface. We encourage any interested users to extend the code base that we provide and add any models that could be useful for the community.

sfs_coder is currently under developement and will be released in a few months. We don't expect these pages to get much traffic at the moment but we have put them online for our development team. If you have stumbled across them and have any suggestions or would like to inquire about the development feel free to email Lawrence (uricchio@berkeley.edu).

sfs_coder will be free to use and distribute for personal or academic use. We hope that you will find it useful.

CHAPTER

TWO

INSTALLATION

sfs_coder does not require any installation of its own in order to get its basic functionality. Just download the source and alter the PYTHONPATH variable on your machine so that python knows where to find the modules (see the section on "Importing" below).

However, some methods within sfs_coder require external software in order to run.

2.1 Required dependencies

- SFS_CODE

SFS_CODE can be installed anywhere on the user's machine. The path to the binary is supplied to sfs_coder (see the section on running SFS_CODE through sfs_coder).

- python (*2.7 or greater*)

2.2 Optional dependencies

- mpmath (*required for rescaled recurrent hitchhiking simulations*)
- scipy (*required for rescaled recurrent hitchhiking simulations and some methods in sfsplot*)
- matplotlib (*required for plotting the output*)

2.3 Importing sfs_coder's modules

Python uses the PYTHONPATH system variable to search for modules that are imported. Suppose we download sfs_coder and store it in the directory '~/sfs_coder', and then we try to execute the following script called 'basic.py':

```
import command

com = SFSCommand()
```

If the directory that contains command.py ('~/sfs_coder/src' by default) is not included in the PYTHONPATH variable, this will result in an error similar to the following:

```
Traceback (most recent call last):
  File "basic.py", line 13, in <module>
    import command
ImportError: No module named command
```

Python does not know where the command module is! To fix this, we can add the '~/sfs_coder/src' directory to the PYTHONPATH variable in a couple different ways.

2.3.1 Adding the sfs_coder source directory to PYTHONPATH in .bashrc

If you execute your scripts at the command line with a bash shell, you can add a line to your .bashrc file that will fix this problem and allow you to run the above script.

```
export PYTHONPATH=$PYTHONPATH:~/sfs_coder/src
```

The .bashrc file exists in your home directory and is read by bash every time you open a new shell. If a file called .bashrc doesn't exist in your home directory you can create it.

Of course, if the path to your sfs_coder 'src' directory is different than above you will need to provide the path to your copy of this directory.

2.3.2 Adding the path to sfs_coder's source directory within a python script

You can also add the path to sfs_coder's 'src' directory to any python script if for any reason you don't want to modify your .bashrc as above. Assuming the same directory layout as the above example, we can use:

```
import sys
sys.path.append('~/sfs_coder/src')
import command

com = SFSCommand()
```

This adds '~/sfs_coder/src' to the PYTHONPATH variable within the script. Note that this solution requires us to add this line of code to every python script that imports something from sfs_coder, whereas the first solution allows us to import the modules just like any other python modules.

CHAPTER

THREE

RUNNING SFS_CODE SIMULATIONS

3.1 The basic structure of an sfs_coder script

Running SFS_CODE simulations with sfs_coder requires only a few lines of code.

First, we import the command module, initialize an SFSCommand object, and tell the software where the sfs_code binary is located.

```
import command

com = command.SFSCommand()

com.sfs_code_loc = '/path/to/sfs_code'
```

Next, we need to build a command line. Although this process is very flexible (in fact we can build any command line that is accepted by SFS_CODE), we have prepackaged several models that may be of general interest. For example, to simulate the model of Gutenkunst (2009, *PLoS Genetics*), we call the following:

```
com.gutenkunst()

com.execute()
```

And that's it! Of course, there are many more options that can be altered to modify the parameters of the simulation, such as the ancestral population size. Please see the "scripts" directory in the top level of sfs_coder for more complicated examples. Below, we include a slight modification of the above script that demonstrates some basic functionality that may be useful to users, as well as a few other examples.

```
import command
import os
from random import randint

# initialize an SFS_CODE command, set the prefix of the output subdirectory
com = command.SFSCommand(prefix='guten.N500')

# build the command line for the Gutenkunst model, specifying some parameters
com.gutenkunst(N=500,nsam=50,nsim=10)

# set the location of the sfs_code binary
com.sfs_code_loc = os.path.join(os.path.expanduser('~'),
                        'path/to/sfs_code')

# execute the command, supplying a random number
com.execute(rand=randint(1,100000))
```

3.2 Examples

3.2.1 Adding selection

```
import command
import os
from random import randint

# initialize a new SFS_CODE command
com = command.SFSCommand(prefix='tennessen.N1000')

# build the command line for the tennessen model
# a selection model is added with sel = sel=['-W','1','5','0','1']
# this adds a type 1 selection model, with gamma =5,
# and the probability of negative seleciton set to 1.
# for more on selection models in SFS_CODE, see the SFS_CODE handbook

com.three_pop(N=1000,nsam=[50,50,0],nsim=10,model='tennessen',
              L=['-L','1','100'],sel=['-W','1','5','0','1'])

# set the location of sfs_code
com.sfs_code_loc = os.path.join(os.path.expanduser('~'),
                    'rotations/hernandez/software/sfs_code/bin/sfs_code')

# execute the command
com.execute(rand=randint(1,100000))
```

3.2.2 Simulations with realistic genomic structure and demography

```
import command
import os
from random import randint

# initialize a new SFS_CODE command
com = command.SFSCommand(prefix='guten.lactase')

# build the command line for the gutenkunst model in the lactase region
com.genomic(N=100,model='gutenkunst',sel=False)

# set the location of sfs_code
com.sfs_code_loc = os.path.join(os.path.expanduser('~'),'rotations/hernandez/software/sfs_code/bin/s

# execute the command
com.execute(rand=randint(1,100000))
```

3.2.3 Using SGE

sfs_coder uses the sge_task_id system variable to number output files. If you submit an sfs_coder script to a cluster as an array job, it will take care of all the work of numbering the output files for you.

For example, any of the above scripts can be sent to a cluster with the following header:

```
#!/usr/bin/python
#$ -e sim.div.log
```

```
#$ -o sim.div.log
#$ -S /usr/bin/python
#$ -cwd
#$ -r yes
#$ -l h_rt=240:00:00
#$ -t 1-100
#$ -l arch=linux-x64
#$ -l mem_free=1G
#$ -l netapp=1G
```

3.2.4 Simulations of phenotypes

CHAPTER

FOUR

ANALYZING THE OUTPUT

sfs_code stores a great deal of information about each mutation or substitution in a simulation, which can make it challenging to parse the output. With sfs_coder, all the data is stored internally in the Mutation class, allowing for flexible manipulation of the output.

4.1 Opening and reading an output file

Opening and reading files is simple. Below is an example that reads in all the data in an SFS_CODE output file and caluclates π (the average pairwise diversity) in the 0th locus in all sampled populations.

```
import sys
import readsfs

# an sfs_code output file that we will analyze
f =sys.argv[1]

# initializing a data object and setting the file path
data = readsfs.SFSData(file=f)

# getting all the data from the simulations in the file
data.get_sims()

# The simulations in the file are stored in the data.sims attribute
# for each Simulation object in data.sims, we can calculate pi for
# a set of loci

for sim in data.sims:

    pis = sim.calc_pi(loci=[0]) # array of pi values, indexed by population
    print pis[0]                # pis[0] is pi in the 0th population
```

CHAPTER

FIVE

PLOTTING

CHAPTER

SIX

SFS_CODER CLASSES AND METHODS

6.1 contents of command.py

class command.**Command**

This class stores information from parsed command lines and provides the mechanics to call SFS_CODE commands.

execute(*rand=1*)

execute a simulation command

•Parameters:

–rand=1 a random integer. If the value is not reset by the user then a new random number is rolled within self.execute. This value is used as the random seed for SFS_CODE.

class command.**SFSCommand**(*outdir='/Users/luricchio/projects/cluster_backup/sfs_coder/doc/sims'*, *prefix='out'*, *err='err'*)

This class is used to store, parse, and convert SFS_CODE command lines.

Upon initialization, an object of the SFSCommand class sets the values of many of its attributes to the SFS_CODE defaults.

•Parameters:

–outdir=os.path.join(os.getcwd(), 'sims') A directory containing subdirectories with sfs_code simulations.

–prefix='out' The prefix of the out directory and the data files within the out directory.

–err='err' The name of the directory that contains all the stderr ouput from calling sfs_code.

•Attributes:

–self.com_string='' the entire command stored as a single string.

–self.outdir= outdir the parent directory of output directories for sets of SFS_CODE simulations

–self.sfs_code_loc = '' the location of the SFS_CODE binary.

–self.N = 500 the number of individuals in the ancestral population.

–self.P = [2] the ploidy of the individuals in each population.

–self.t = 0.001 $\theta = 4Nu = 0.001$. This is the value of θ in the ancestral population

–self.L = [5000] an array containing the length of each simulate locus.

–self.B = 5 self.p[0] self.N the length of the burn in (generations).

–self.prefix= prefix the prefix for the output file directory and each simulation file.

–***self.r=0.0.*** $\rho = 4Nr = 0.0$. The value of ρ in the ancestral population.

–***self.n_pops=1*** number of populations.

–***self.n_iter=1*** number of simulations.

–***self.line=[]*** an array of strings, each of which is an argument to SFS_CODE. This is the attribute that is used to execute SFS_CODE commands.

add_event(*event*)
add an event to a command line.

•Parameters:

–***event*** an array of strings corresponding to the event.

e.g., adding a mutation at locus 0 at time 0 with

```
event = ['--mutation', '0','L','0']
```

build_RHH(*alpha=1000.0*, *N0=5000.0*, *rho0=0.001*, *lam0=1e-10*, *delta=0.01*, *L0=-1*, *L1=100000.0*, *loop_max=10*, *L_neut=1000.0*, *theta_neut=0.001*, *minpop=100*, *recomb_dir='recombfiles'*, *outdir='sims'*, *TE=2*, *r_within=True*, *neg_sel_rate=0.0*, *alpha_neg=5*, *additive=1*, *Lextend=1*, *mutation=[ ]*, *bottle=[ ]*, *expansion=[ ]*, *nreps=10*, *Boyko=False*, *Lmult=False*, *lmultnum=50*, *Torg=False*, *non_coding=True*)
A method to build a recurrent hitchhiking command line using the method of Uricchio & Hernandez (2014, *Genetics*).

•Dependencies:

–scipy

–mpmath

•Parameters

–***alpha = 1000*** $\alpha = 2Ns$, the ancestral population scaled strength of selection. Note that demographic events can change N, and hence they also changle alpha.

–***N0 = 5000*** the ancestral population size

–***rho0 = 0.001*** the population scaled recombination coefficient in the ancestral population.

–***lam0 = 10*-10*** the rate of positive substitutions per generation per site in the population.

–***delta = 0.01*** a single parameter that encapsulates both delta parameters from Uricchio & Hernandez (Genetics, 2014). Smaller values of delta result in dynamics that are a better match for the original population of size N0, but are more computationally expensive. We do not recommend using values of delta greater than 0.1. For more information please see the paper referenced above.

–***L0 = -1*** the length of the flanking sequence on each side of the neutral locus. If L0 is not reset from it's default value, it is automatically set to L0 = s0/r0, where s0 and r0 are alpha/2N0 and rho/4N0, respectively.

–***theta_neut = 0.001*** the neutral value of theta.

–***TE=2*** the ending time of the simulation in units of 2*N0*self.P[0] generations.

–***r_within=False*** Currently only works with this option set to False, but in the future will allow for recombination within the neutral locus.

genomic(*basedir='../src/../src/req', indir='/Users/luricchio/projects/cluster_backup/sfs_coder/doc/input_files', datafile='hg19_gencode.v14.gtf.gz', phast_file='hg19_phastCons_mammal.wig.gz', dense_dist=5000, begpos=134545415, endpos=138594750, db=-1, chr=2, de=-1, withseq=0, fafile='', N=2000, mutation=[], sel=True, model='', t=0.001, rho=0.001, nsim=10, nsam=[20]*)

A method for running simulations of genomic elements using realistic genome structure and demographic models. The demographic models of Gutenkunst (2009, *PLoS Genetics*), Gravle (2011, *PNAS*), Tennessenn (2012,*Science*), and the standard neutral model are included.

The default data sources and options are all human-centric, but in principle these methods could be used to simulate sequences from any population for which the relevant data sources are available (recombination map, conserved elements, exon positions).

This function calls a number of perl scripts, originally implemented by Ryan Hernandez, to build the input to SFS_CODE. These perl scripts are bundled with sfs_coder in the directory src/req

•Parameters:

–basedir=os.path.join(os.path.dirname(__file__),'../src/req') the directory where all the data-sources and perl for this method are located. You shouldn't have to change this unless you're moving around the source files relative to each other.

–outdir=os.path.join(os.getcwd(),'input_files') the directory where the sfs_code annotation files will be written

–chr=2 chromosome number to query

–begpos=134545415 genomic coordinate of the beginning of the simulated sequence

–endpos=138594750 genomic coordinate of the end of the simulated sequence

–db=begpos genomic coordinate of the region in which to include dense neutral sequences. Neutral sites within this region will be simulated if they are within a specified distance of one of the simulated genomic elements (given by *dense_dist*)

–de=begpos genomic coordinate of the end of the dense neutral region

–dense_dist=5000 The amount of neutral sequence to pad onto the end of each conserved element or exon. For example, if two adjaacent conserved elemets are 20,000 base pairs apart, and the dense_dist=5,000, then 5,000 base pairs are padded onto the end of each of the conserved elements and the middle 10,000 base pairs are not simulated.

To include no neutral sequence, use dense_dist = 0.

To include every neutral base pair in the region, use dense_dist=-1 (potentially very computationally expensive for large regions!)

–sel=True If true, draw selection coefficients from a gamma distribution of selection coefficients for conserved elements and coding regions. These distributions are taken from Boyko et al (coding, 2008, *PLoS Genetics*) and Torgerson et al (conserved non-coding, 2009, *PLoS Genetics*).

gutenkunst(*add_on=False, nsim=1, N=10000, non_coding=False, recombfile='', nsam=[100], mutation=[], t=0.001, rho=0.001, loci=[], sel=[], L=[]*)

A method that adds the Gutenkunst (2009, *PLoS Genetics*) model to an SFS_CODE command line.

parse_string()

A method to parse SFS_CODE command lines. By default, every switch is stored as an array with the exception of certain special cases that are stored as dictionaries.

Note, **only the short form of SFS_CODE options are currently fully supported!** For example, *-t 0.002* is supported but *–theta 0.002* is not. Hence, if you run SFS_CODE with the long forms or wish to analyze code that used the long form to run, you may have issues.

three_pop(*add_on=False, nsim=1, N=10000, non_coding=False, recombfile='', nsam=[100, 100, 100], mutation=[], t=0.001, rho=0.001, loci=[], sel=[], L=[], t_end=0.60274, t_expand_p0=0, expand_p0=1.68493, t_split_p0_p1=0.219178, t_split_p1_p2=0.544658, bottle_p1_0=0.170732, bottle_p1_1=0.47619, bottle_p2=0.242857, growth_p1=58.4, growth_p2=80.3, mig_p0_p1=6.15, mig_p1_p0=0.5, mig_all=[0.738, 0.4674, 0.06, 0.192, 0.01938, 0.09792], t_super=0.391465, model='guten'*)

A general three population model with growth events. Inspired by Gutenkunst et al (2009, *PLoS Genetics*) and Gravel et al (2011, *PNAS*). For a pictorial representation see Figure 3A of the Gutenkunst paper.

The default parameters are set to Gutenkunst et al. The maximum likelihood estimates of Gravel et al and Tennessen et al (2012, *Science*) are also included.

Use model='gravel', model='tennessen' or model='guten' to explicitly choose one of the models.

The user can specify the model parameters as desired. To use a user specified model, simply use model = ''. Any unspecified parameters are set by default to the parameters of the Gutenkunst model.

6.2 contents of sfs.py

class sfs.**Mutation**

a class to store the data associated with a variant in an SFS_CODE output file. Note, both mutations and substitutions are stored as instances of this class.

•Attributes:

–self.locus=-1 The locus number of the variant

–self.AXY='?' 'A' for autosomal, 'X' or 'Y' for the corresponding sex chromosomes

–self.pos=-1 The position within the locus. Note that the positions within each locus start from 0.

–self.t_init={} A dictionary, keyed by population number, and storing the time that the variant arose

–self.t_fix={} A dictionary, keyed by population number, and storing the time that the variant fixed within the population. If the variant is segregating, the time stored is the time of sampling.

–self.tri_nuc='NNN' The ancestral trinucleotide (the middle base is the mutated base, so this is not necessarily a codon!)

–self.deriv_n='N' The derived nucleotide

–self.non_or_syn='?' Is the mutation synonymous (0) or nonsynonymous (1). 0 also is used to indicate non-coding.

–self.ancest='?' Ancestral amino acid

–self.deriv_aa='?' Derived amino acid

–self.fit='?' fitness effect of the mutation (0 for neutral)

–self.chrs = defaultdict(dict) A dictionary of dictionaries that is keyed by population and chromosome number.

E.g., if the derived allele is present on chromosome 11 in population 2, then

self.chrs[2][11] = True

–self.pops_numchr = {}

A dictionary that stores the number of chromosomes that carry the derived allele in each population.

class sfs.**SFSData**(*file=''*)

A class that handles the basic parsing of sfs_code output file data.

•Parameters:

–file= '' the path to the file that is to be read.

•Attributes:

–self.file = file the path to the file that is to be read.

–sims = [] an array of sfs.Simulation objects.

get_sims()

A method that reads sfs_code output files and stores all the data in sfs.Simulation objects.

class sfs.**Simulation**

A class to store the data from SFS_CODE simulations.

•Attributes:

–self.command = command.SFSCommand()

–self.data = ''

–self.loci = defaultdict(lambda: defaultdict(list))

A dictionary of dictionaries indexed by locus and position. Each dictionary of dictionaries is a list of Mutation objects that occur at the corresponding locus and position.

e.g., self.loci[0][0] is a list of Mutation objects that occur at the first position in the first locus.

Note that these keys will only exist in self.loci if there were mutations at this particular point in the sequence in the sample from the simulation.

–self.muts = []

A list of all the Mutation objects in the simulation.

calc_S(*multi_skip=True*, *loci=*[ ], *pop=0*)

calculate the number of segretating sites within all populations.

•Parameters:

–multi_skip = True skip sites that are more than biallelic if True

–loci = [] A list of loci over which to calculate the number of segregating sites. Uses all loci if this is left blank.

calc_fit(*pop=0*)

calculate the fitness of the sampled chromosomes within a population.

•Parameters:

–pop=0 population number

calc_pi(*multi_skip=True*, *loci=*[ ])

calculate the mean pairwise diversity per site bewteen pairs of sequences across a set of loci. If the loci parameter is left undefined by the user, then this method calculates π over all loci in the simulation.

•Parameters:

–multi_skip=True

If True, skip loci that are multiallelic. Otherwise lump all the derived alleles together.

–loci=[]

An array of loci over which to calculate π. If left blank, all loci are used in the calculation.

- Return value: An array of values of π values indexed by population number.

calc_pi_by_locus()
calculate the value of π independently for each locus.

- Return value: An array of arrays of pi values, indexed by population and then locus number.

e.g., if the return value is stored in the variable pi, pi[0][1] is the value of π in population 0 at locus 1.

get_sfs(*pop=0*, *NS=True*, *SYN=True*)
compute the site frequency specutrum for a population

- Parameters:

–pop=0 population number

6.3 contents of sfsplot.py

6.4 contents of ms.py

CHAPTER

SEVEN

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

c

m

s