sfs_coder Documentation

Release 0

Lawrence Uricchio, Raul Torres, Ryan Hernandez

April 08, 2014

CONTENTS

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 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 put it wherever you want on your machine. However, some functionality within sfs_coder requires 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. The .bashrc file exists in your home directory and is read by bash every time you open a new shell. If the file doesn't exist in your home directory you can create it.

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

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 your sfs_coder 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 whereas the first solution allows us to import the modules just like any other python modules.

CHAPTER

THREE

RUNNING SFS_CODE SIMULATIONS

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

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

```
import command

com = 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 added to modify the parameters of the simulation. 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.

```
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 with N=500, etc
com.gutenkunst(N=500,nsam=50,nsim=10)

# set the location of sfs_code, set the prefix of the out files
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))
```

CHAPTER

FOUR

ANALYZING THE OUTPUT

CHAPTER

FIVE

PLOTTING

class `command.`**`Command`**

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

`add_out``()`

Adds the path to an output directory to self.line. This method takes no arguments but uses self.prefix (a label for the set of simulation experiments you are doing) and self.outdir (a label for the directory in which sets of simulations are stored).

Thus, the full path is set to:

- *os.path.join(self.outdir,self.prefix)*

The values of self.outdir and self.prefix are set when an object is instantiated. See the documentation of command.SFSCommand() for the defaults of these values.

`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.

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_rand`(*rand*)
Just a little helper method to add random numbers to command lines. Not really recommended for use outside of this module.

`build_BGS`(*n_sim=10, theta=0.0001, recomb=False, rho=0.001, N=250, n_sam=10, alpha=5, L=100000, Lmid=100000*)
A method for running simulations of background selection. This method is a bit underdeveloped at the moment, so stay tuned for more on how it works.

`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=False, neg_sel_rate=0.0, alpha_neg=5, additive=1, Lextend=1, mutation=[ ], bottle=[ ], expansion=[ ]*)
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.

build_genomic(*basedir='/Users/luricchio/projects/cluster_backup/sfs_coder/doc/req'*, *outdir='/Users/luricchio/projects/cluster_backup/sfs_coder/doc/input_files'*, *datafile='hg19_gencode.v14.gtf.gz'*, *chr=2*, *begpos=134545415*, *endpos=138594750*, *db=136545415*, *de=136594750*, *withseq=0*, *fafile='hg19.fa.gz'*, *phast_file='hg19_phastCons_mammal.wig'*, *dense_dist=5000*, *regname=''*, *annotfile=''*, *recombout=''*, *sel=True*)

A method to build simulations of genomic regions with realistic genome structure. 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.

convert_sfs_ms()

A method to convert an sfs_code command line to ms. Most but not all switches are converted (the method should let you know if you try to use an unsupported switch). The method will ignore selection based switches in SFS_CODE (since ms only includes neutral demographic models).

This method has been tested only sparingly and it is recommended that it is used with great caution.

genomic(*basedir='/Users/luricchio/projects/cluster_backup/sfs_coder/doc/req'*, *outdir='/Users/luricchio/projects/cluster_backup/sfs_coder/doc/input_files'*, *datafile='hg19_gencode.v14.gtf.gz'*, *chr=2*, *begpos=134545415*, *endpos=138594750*, *db=136545415*, *de=136594750*, *withseq=0*, *fafile='hg19.fa.gz'*, *phast_file='hg19_phastCons_mammal.wig'*, *dense_dist=5000*, *N=2000*, *mutation=[]*, *sel=True*)

A method for running simulations of genomic elements using realistic genome structure. Most of the heavy lifting is done by the build_genomic() method. This method exists mostly as a slimmed down interface to that method, with the major difference being that this version allows for the inclusion of a demographic model. Currenrly, only the demographic model of Gutenkunst (2009, *PLoS Genetics*) is included.

gutenkunst(*add_on=False*, *nsim=1*, *N=10000*, *non_coding=False*, *recombfile=''*, *nsam=100*, *mutation=[]*, *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.

class sfs.**Simulation**

A class to store the data from SFS_CODE simulations.

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

calculate the number of segretating sites within a specific population.

- Parameters:
  - *pop = 0* population in which to calculate the number of segregating sites.
  - *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.

make_muts()

This method takes the string of data that is generated by readsfs and parses it into individual mutations. It handles some of the oddities of how mutations are stored when they segregate in multiple populations.

This is mostly intended for internal use within this module, so use or modify at your own risk.

class readsfs.**FileData**(*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 `readsfs.`**`msData`**`(`*file*=’‘`)`

A class to store ms output file data.

`get_sims``()`

A method that parses ms output files and stores each simulation as an ms.Simulation object. This method is *not* fully developed at this time and currently does not store several essential pieces of information, such as the positions of the segregating sites.

CHAPTER

SIX

INDICES AND TABLES

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

PYTHON MODULE INDEX

INDEX