pypgen Documentation

Release 0.2.2 beta

Nicholas Crawford

September 28, 2016

Contents

Pypgen provides various utilities for estimating standard genetic diversity measures including Gst, G'st, G''st, and Jost's D from large genomic datasets (Hedrick, 2005; Jost, 2008; Masatoshi Nei, 1973; Nei & Chesser, 1983). Pypgen operates both on individual SNPs as well as on user defined regions (e.g., five kilobase windows tiled across each chromosome). For the windowed analyses pypgen estimates the multi-locus versions of each estimator.

CHAPTER 1

Features:

- Handles multiallelic SNP calls
- Allows a single VCF file to contain multiple populations
- Operates on standard VCF (Variant Call Format) formatted SNP calls
- Uses bgziped input for fast random access
- Takes advantage of multiple processor cores
- Calculates additional metrics:
  - snp count per window
  - mean read depth (+/- STDEV) per window
  - populations with fixed alleles per SNP

CHAPTER 2

Prerequisites:

Pypgen is written in Python 2.7. It may run under Python 2.6, but I haven't tested it. It doesn't run under Python 3. In order to interact with bgziped files it requires samtools and pysam to be installed.

CHAPTER 3

Quick Installation:

If you already have a working install of pysam, pypgen can be installed from PyPi using pip or setuptools:

```
pip install pypgen
```

or,

```
easy_install -U pypgen
```

However, it's recommended, at least in these early days of pypgen, to install it directly from the github repository:

```
pip install -e git+https://github.com/ngcrawford/pypgen.git#egg=Package
```

CHAPTER 4

Reporting Problems:

If you have a general questions about pypgen you should post them on biostar and tag it `pypgen`. Detailed questions about If you think you've found a bug in pypgen you can open an issue in the pypgen github repo.

CHAPTER 5

Contents:

5.1 Detailed Installation:

Installing pypgen is very straightforward especially if you are familiar with installing python packages. Just follow the instructions below entering the appropriate commands in terminal.

5.1.1 Samtools and tabix

In OS X:

1. Install Xcode or Xcode Command Line Tools. The CLI tools take up less space, but are an optional install under Xcode. Details on how to do this may be found in the homebrew documenation.
2. Once xcode is installed, install the homebrew package installer:

```
$ ruby -e "$(curl -fsSkL raw.github.com/mxcl/homebrew/go)"
```

3. Then install samtools using homebrew:

```
$ brew tap homebrew/science
$ brew install samtools
$ brew install tabix
```

While you're at it you might want to use brew to install `wget`.

```
$ brew install wget
```

From source code (e.g., on linux):

1. Download the latest version of samtools and tabix.

```
# replace the ###version### with the appropriate version number (e.g., 0.2.6)

$ wget http://sourceforge.net/projects/samtools/files/tabix/tabix-###version###.tar.bz2
$ wget http://sourceforge.net/projects/samtools/files/samtools/0.1.18/samtools-0.1.18.ta
```

2. Extract the tar.bz2 files

```
tar jxf *.tar.bz2
```

3. Then run `make` in each directory

4. You'll need to add these directories to your system profile files (e.g., `.bashrc` or `.bash_profile`)

You can check that everything is working by opening a fresh shell. The commands `samtools` and `tabix` should now be available from anywhere in the file system.

5.1.2 Pip and Setuptools

1. Unfortunately, `pip` doesn't yet 'play well' with with `setup.py`, the script the automates the installation of python packages, as `setup.py` still relies on `easy_install/setuptools` to install dependancies. This means you'll need to install setup tools. You'll need to download the appropriate python .egg file. Then you can run it as an installation script.

```
$ [sudo] sh setuptools-0.6c9-py2.7.egg
```

2. Then you can use `easy_install` to install pip:

```
$ [sudo] easy_install install -U pip
```

5.1.3 Pypgen

Pypgen can be installed from PyPi using pip or setuptools:

```
$ [sudo] pip install pypgen
```

or,

```
$ [sudo] easy_install -U pypgen
```

However, it's recommended, at least in these early days of pypgen when I'm actively fixing bugs, to install it directly from the github repository:

```
$ [sudo] pip install -e git+https://github.com/ngcrawford/pypgen.git#egg=Package
```

This should complete your install.

5.1.4 UnitTests

UnitTests are in `pygen/tests/tests.py`

5.1.5 Pysam

NOTE: Pysam should automatically install when you install pypgen. These instructions are only necessary if you have problems with it.

Pysam is a bit of a finicky installation. The newest versions, in particular seem to have a lot of problems linking to their compiled cython libraries. With that in mind I recommend installing 0.6.

```
$ [sudo] pip install pysam==0.6
```

or,

```
$ [sudo] easy_install -U pysam==0.6
```

If that doesn't work you'll want to try installing it from source:

```
# replace the ###version### with the appropriate version number (e.g., 0.7.4)

$ wget http://pysam.googlecode.com/files/pysam-###version###.tar.gz
$ tar xzf pysam-###version###.tar.gz
```

then `cd` into the directory and run:

```
$ [sudo] python setup.py install
```

There is also a pysam google group that is a good source of information.

5.2 Tutorial:

Once pygen is installed two scripts, `vcfWindowedFstats` and `vcfSNVfstats`, should be available at the command line.

Running [script name].py will print out a short list of commands and adding the `--help` or `-h` prints out a more detailed list.

5.2.1 Basic analysis

1. Run your samples through GATK or samtools (or similar SNV caller) that emits calls in the standard VCF format. By default pypgen's VCF parser only looks at SNVs where the `FILTER` column is set to `PASS` so you should filter or recalibrate your VCF appropriately before running pypgen.
2. Once you have a VCF file you'll need to bgzip it. Tabix include bgzip so you make sure you have tabix installed. Tabix and samtools installation is detailed in the *Samtools and tabix* section of this guide. The basic command to run bgzip is:

```
bgzip -c  path/to/vcf_file.vcf > path/to/vcf_file.vcf.bgz
```

   This can exceed 30 minutes if your uncompressed VCF file is very large.

3. Next you need to index your bgzipped VCF file. The command to do this is:

```
tabix -p vcf path/to/vcf_file.vcf.bgz
```

   This command will produce a `path/to/vcf_file.vcf.tbi` index file.

4. Now you can run pypgen. In a text editor I recommend composing a test command that looks something like this.

```
vcfSNVfstats \
     -i path/to/vcf_file.vcf.bgz \
     -p pop1:sample1,sample2 \
        pop2:sample3,sample4,sample5 \
        pop3:sample6,sample7,sample8 \
     -c 2 \
     -r Chr:1-10001 | head
```

   You'll need to replace `path/to/vcf_file.vcf.bgz` as you did in the last command.

You'll also need to associate the sample names with their populations. The sample names should to exactly match the sample IDs in the VCF file. If you've forgotten what they are you can run the following command to print them out.

```
gunzip -c pypgen/data/example.vcf.gz | grep "#CHROM"
```

You will also want to change the regions flag such that it selects a valid region in your VCF file.

Piping the output into `head` prevents flooding your terminal with output.

If you have an enormous number of samples and get an error like `Argument list too long` you can just save the text file as a shell script and run it like:

```
sh path/to/shell_script.sh
```

If everything worked you should see a header line followed by ~ 9 lines of data. The amount of output varies depending on the region so it's a good idea to pick a region that you know contains SNVs.

Replacing `vcfSNVfstats` with `vcfWindowedFstats` and setting the `--window` flag is all that is necessary to run a sliding window analysis

5.2.2 Followup analysis

5.3 Scripts:

5.3.1 vcfSNVfstats

This script calculates *F*-statistics for each pair of populations at each SNV in the supplied region.

Working Example:

Note that `path/to/pypgen/data/example.vcf.gz` needs to be updated to the directory in which the source code for `pypgen` is found.

```
vcfSNVfstats \
  -i pypgen/data/example.vcf.gz \
  -p outgroups:h665,i02-210 \
     pop1:c511,c512,c513,c514,c515,c563,c614,c630,c639,c640 \
     pop2:m523,m524,m525,m589,m675,m676,m682,m683,m687,m689 \
  -c 2 \
  -r Chr01:1-10001 | head
```

Command Line Flags

Input: [`-i`, `--input`]

Defines the path to the input VCF file.

Output: [`-o`, `--output`]

Defines the path to the output csv/txt file. If it's not set it defaults to standard out (stout).

Cores: [`-c`, `--cores`]

The number of cores to use.

Regions: [`-r`, `-R`, `--regions`]

This allows for selecting a subset of the VCF file for analysis. The command format should familiar to if you use GATK or samtools. A region can be presented, for example, in the following format: 'chr2' (the whole chr2), 'chr2:1000000' (region starting from 1,000,000bp) or 'chr2:1,000,000-2,000,000' (region between 1,000,000 and 2,000,000bp including the end points). The coordinate system is 1-based. Multiple regions can be submitted separated by spaces. [Note: this is the same format as samtools/GATK and this example text is largely borrowed from samtools]

Populations: [`-p`, `--populations`]

Names of populations and samples. The format is: "PopName:sample1,sample2,.. PopName2:sample3,sample4,..." with colons after each population name and samples delimited by commas. Whitespace is used to delimit populations.

Minimum Number of Samples: [`-m`, `--min-samples`]

This allows one to set the minimum number of samples per population that a SNV needs to have in order to be included in the analysis.

Column Separator: [`-s`, `--column-separator`]

This allows one to set the separator to be uses in the output. The default value is `,` which makes the output comma separated (csv). If you're planning on using tabix to index the output you'll need to set the sep to `\t`.

Zero Based: [`--zero-based`]

Setting this flag makes the output positions zero based (e.g., BED like).

Output

- The chrom and pos columns are fixed in positions 1 and 2, but the rest of the columns vary depending on the number of populations being compared and their names.

Label:	Definition:
chrom	ID of chromosome/scaffold/contig/etc.
pos	Position of SNP
pop1.sample_count	Number of samples represented
cont.	Additional population sample counts
Pop1.Pop2.D_est	D corrected for sample size (Jost 2008)
Pop1.Pop2.G_double_prime_st_est	Corrected Hedrick's G'st (Meirmans & Hedrick 2011)
Pop1.Pop2.G_prime_st_est	Standardized Gst (Hedrick 2005)
Pop1.Pop2.Gst_est	Fst corrected for sample size and allowing for multiallelic loci (Nei & Chesser 1983)
Pop1.Pop2.Hs_est	Within-population gene/locus diversity (e.g., expected heterozygosity)
Pop1.Pop2.Ht_est	Total gene/locus diversity
cont...	Pairwise comparisons of F-statistics cont...
Pop1_fixed	If a sample is fixed at a particular allele this flag is set to 1 (= "True" in binary)
cont...	Additional fixed SNPs cont...

5.3.2 vcfWindowedFstats

This script calculates *F*-statistics for each pair of populations at each window in the supplied region. This script requires that the input VCF file be bgzipped because it uses `tabix` to extract the windows.

Working Example:

Note that `path/to/pypgen/data/example.vcf.gz` needs to be updated to the directory in which the source code for `pypgen` is found.

```
vcfWindowedFstats \
  -i path/to/pypgen/data/example.vcf.gz \
  -p outgroups:h665,i02-210 \
     pop1:c511,c512,c513,c514,c515,c563,c614,c630,c639,c640 \
     pop2:m523,m524,m525,m589,m675,m676,m682,m683,m687,m689 \
  -c 2 \
  -w 5000 \
  -r Chr01:1-10001 | head
```

Command Line Flags

`vcfWindowedFstats` shares the same commands as `vcfSNVfstats` with the single addition of a window size flag.

Window Size: [`-w`, `--window-size`]

Windows are non overlapping and start at the first bp in the particular chromosome.

Output

vcf_sliding_window.py:

- The format is loosely based on the BED specification. Although the first three column IDs will remain static for the foreseeable future, I expect to add more fields as I add additional functionality to pypgen. Also, the default output is one based, but it is possible to make the positions zero based by including the `--zero-based` flag when you run the script.
- The population IDs and the total number of populations come from those defined by the user. This means the number of pairwise population comparisons and hence the total number of columns is conditional on the number of defined populations.

Label:	Definition:
chrom	ID of chromosome/scaffold/contig/etc.
chromStart	Starting position of window
chromEnd	Ending position of window
snp_count	Total Number of SNPs in window
total_depth_mean	Mean read depth across window
total_depth_stdev	Standard deviation of read depth across window
Pop1.sample_count.mean	Mean number of samples per snp for 'Pop1'
Pop1.sample_count.stdev	Standard deviation of samples per snp for 'Pop1'
Pop2.sample_count.mean	Mean number of samples per snp for 'Pop2'
Pop2.sample_count.stdev	Standard deviation of samples per snp for 'Pop2'
Pop2.Pop1.D_est	Multilocus D_est (Jost 2008)
Pop2.Pop1.D_est.stdev	Standard Deviation of SNVwise D_est across the window
Pop2.Pop1.G_double_prime_st_est	Corrected Hedrick's G'st (Meirmans & Hedrick 2011)
Pop2.Pop1.G_double_prime_st_est.stdev	Standard Deviation of Corrected Hedrick's G'st across the window
Pop2.Pop1.G_prime_st_est	Standardized Gst (Hedrick 2005)
Pop2.Pop1.G_prime_st_est.stdev	Standard Deviation of Standardized Gst across the window
Pop2.Pop1.Gst_est	Fst corrected for sample size and allowing for multiallelic loci (Nei & Chesser 1983)
Pop2.Pop1.Gst_est.stdev	Standard Deviation of Fst corrected for sample size and allowing for multiallelic loci (Nei & Chesser 1983)
cont...	The rest of the pairwise comparisons follow...

CHAPTER 6

Indices and tables

- genindex
- modindex
- search