Population Genetics in BioPerl HOWTO

Jason Stajich, Dept Molecular Genetics and Microbiology, Duke University
<jason-at-bioperl-dot-org>

2005-03-1

Revision History

Revision 0.1	2004-06-28	JES
	First draft	
Revision 0.2	2004-02-22	JES
	Updated method docs	
Revision 0.3	2005-03-05	JES
	Expanded to cover coalescent and others	

Table of Contents

1. Introduction

We have aimed to build a set of modules that can be used as part of an automated process for testing population genetics and molecular evolutionary hypotheses. These typically center around sequence based data and we have built a set of routines which will enable processing of large datasets in a pipeline fashion.

To see results of using these tools see Stajich and Hahn (2005) using `tajima_D`, Hahn MW et al (2004) using `composite_LD`, and Rockman MV et al (2003) using F_{st}.

This document will be split up into sections which describe the data objects for representing populations, tests you can perform using these objects, a coalescent implementation, and objects for performing sequence distance based calculations. A full treatment of the Bioperl interface to the PAML suite (Z.Yang, 1997) is covered in the *PAML HOWTO* [http://bioperl.org/HOWTOs/html/PAML.html] and objects and data pertinent to phylogenetic data manipulation are covered in the *Trees HOWTO* [http://bioperl.org/HOWTOs/html/Trees.html].

2. The Bio::PopGen Objects

In Bioperl we have created a few objects to describe population genetic data. These are all located in the Bio::PopGen namespace, so they can be browsed by looking at the Bio/PopGen directory.

Bio::PopGen::Population is a container for a set of Bio::PopGen::Individual in order to represent individuals from a population. Each Individual has a set of Bio::PopGen::Genotype genotype objects which are an allele set associated with a unique marker name. Methods associated with the Population object can calculate the summary statistics such as `pi`, `theta`, `heterozygocity` by processing each Individual in the set.

A Marker is the name given to a polymorphic region of the genome. Markers are represented by a `Bio::PopGen::Marker` object which can contain information such as allele frequencies in a population. Derived subclasses of the main `Bio::PopGen::Marker` are used to store specialized information about markers where supported by data formats. This is done particularly in the `Bio::Pedigree` objects which are a set of modules derived from `Bio::PopGen` and intended to handle the case of interrelated individuals.

3. Building Populations

Although a typical user will want to obtain data for analysis from files or directly from databases we will describe briefly how to create Individuals with Genotypes and Populations of Individuals directly in the code to illustrate the parameters used and access to the data stored in the objects.

A genotype is a triple of a marker name (string), an individual id (string or int), and set of alleles (array of string). The individual_id field is optional as it is explicitly set when a genotype is added to and individual. We can instantiate a Genotype object by using the following code.

```
use Bio::PopGen::Genotype;
my $genotype = Bio::PopGen::Genotype->new(-marker_name   => 'D7S123',
                                          -individual_id => '1001',
                                          -alleles       => ['104','107'],
                                          );
```

To get the alleles back out from a Genotype object the

`get_Alleles`

method can be used. To replace alleles one must call the

`reset_Alleles`

and then

`add_Allele`

with a list of alleles to add for the genotype.

This genotype object can be added to an individual object with the following code which also builds an individual with an id of '1001'.

```
use Bio::PopGen::Individual;
my $ind = Bio::PopGen::Individual->new(-unique_id  => '1001',
                                       -genotypes  => [$genotype]
                                       );
```

There is no restriction on the names of markers nor is there any attempted validation that a genotype's individual_id is equal to the id of Individual is has been associated with it.

Additional genotypes can be added to an individual with the add_Genotype method as the following code illustrates.

```
$ind->add_Genotype(Bio::PopGen::Genotype->new(
                     -alleles       => ['102', '123'],
                     -marker_name   => 'D17S111'
                                       )
```

```
                    );
```

A population is a collecion of individuals and can be instantiated with all the individuals at once or individuals can be added to the object after it has been created.

```
use Bio::PopGen::Population;
my $pop = Bio::PopGen::Population->new(-name        => 'pop name',
                                       -description => 'description',
                                       -individuals => [$ind] );
# add another individual later on
$pop->add_Inidividual($ind2);
```

Using these basic operations one can create a `Population` of individuals. `Bio::PopGen::Marker` objects are intended to provide summary of information about the markers stored for all the individuals.

Typically it is expected that all individuals will have a genotype associated for all the possible markers in the population. For cases where no genotype information is available for an individual empty or blank alleles can be stored. This is necessary for consistency when running some tests on the population but these blank alleles do not get counted when evaluating the number of alleles, etc. Blank alleles can be coded as a dash ('-'), as a blank or empty (' ', or ''), or as missing '?'. The 'N' allele is also considered a blank allele. The regexp used to test if an allele is blank is stored in the `Bio::PopGen::Genotype` as the package variable $BlankAlleles. The following code resets the blank allele pattern to additionally match '.' as a blank allele. This code should go BEFORE any code that calls the `get_Alleles` method in `Bio::PopGen::Genotype`.

```
use Bio::PopGen::Genotype;
$Bio::PopGen::Genotype::BlankAlleles = '[\s\-N\?\.]';
```

`Bio::PopGen::Marker` is a simple object to represent polymorphism regions of the genome.

4. Reading and Writing Population data with Bio::PopGen::IO

Typically one wants to get population data from a datafile.

To read data in CSV format

The CSV format is a comma delimited format where each row is for an individual. The first column gives the individual or sample id and the rest of the columns are the alleles for the individual for each marker. The names of the markers in these rows are listed in the header or which is the very first line of the file.

```
SAMPLE,D17S1111,D7S123
1001,102 123,104 107
1002,105 123,104 111
```

To read in this CSV we use the `Bio::Popgen::IO` object and specify the csv format. We can call `next_individual` repeated times to get back a list of the individuals (one is returned after each time the iterator is called). Additionally the `next_population` is a convience method which will read in all the individuals at once and create a new `Bio::PopGen::Population` object containing all of thse individuals. The CSV format assumes that ',' is the delimiter between columns while '\s+' is the delimiter between alleles. One can override these settings by providing the -field_delimiter and -allele_delimited argument to Bio::Popgen::IO when instantiating. Additionally

a flag called -no_header can be supplied which specifies there is no header line in the report and that the object should assign arbitrary marker names in the form 'Marker1', 'Marker2' ... etc.

Pretty Base format

Phase and hapmap format

5. Allele data from Alignments using `Bio::AlignIO` and `Bio::PopGen::Utilities`

Often one doesn't already have data in SNP format but want to determine the polymorphisms from an alignment of sequences from many individuals. To do this we can read in an alignment and process each column of the alignment determine if it is polymorphic in the individuals assayed. Of course this will not work properly if the alignment is bad or with very distantly related species. It also may not properly work for gapped or indel columns so we might need to recode these as Insertion or Deletion depending on the questions one is asking.

The modules to parse alignments are part of the `Bio::AlignIO` system. To parse a clustalw or clustalw-like output one uses the following code to get an alignment which is a `Bio::SimpleAlign` object.

```
use Bio::AlignIO;
my $in = Bio::AlignIO->new(-format => 'clustalw', -file => 'file.aln');
my $aln;
if( $aln = $in->next_aln ) { # we use the while( $aln = $in->next_aln ) {}
                             # code to process multi-aln files
      # $aln is-a Bio::SimpleAlign object
}
```

The `Bio::PopGen::Utilities` object has methods for turning a `Bio::SimpleAlign` object into a `Bio::PopGen::Population` object. Each polymorphic column is considered a *Marker* and as assigned a number from left to right. By default only sites which are polymorphic are returned but it is possible to also get the monomorphic sites by specifying -include_monomorphic => 1 as an argument to the function. The method is called as follows.

```
use Bio::PopGen::Utilities;
# get a population object from an alignment
my $pop = Bio::PopGen::Utilities->aln_to_population(-alignment=>$aln);
# to include monomorphic sites (so every site in the alignment basically)

my $pop = Bio::PopGen::Utilities->aln_to_population(-alignment=>$aln,
                                                    -include_monomorphic =>1);
```

In the future it will be possible to just ask for the sites which are synonymous and non-synonymous if one can assume the first sequence is the reference sequence and that the sequence only contains coding sequences.

6. Summary Statistics with `Bio::PopGen::Statistics`

Pi or average pairwise differences is calculated by taking all pairs of individuals in a population and computing the average number of differences between them. To use pi you need to either provide a `Bio::PopGen::PopulationI` object or an arrayref of `Bio::PopGen::IndividualI`. Each of the individuals in the population need to have the same complement of Genotypes for the Markers with the same name.

```
use warnings;
use strict;
use Bio::PopGen::IO;
use Bio::PopGen::Statistics;
my $stats= Bio::PopGen::Statistics->new();
my $io = Bio::PopGen::IO->new(-format => 'prettybase',
         -fh     => \*DATA);
if( my $pop = $io->next_population ) {
    my $pi = $stats->pi($pop);
    print "pi is $pi\n";

    # to generate pi just for 3 of the individuals;
    my @inds;
    for my $ind ( $pop->get_Individuals ) {
 if( $ind->unique_id =~ /A0[1-3]/ ) {
     push @inds, $ind;
 }
    }
    print "pi for inds 1,2,3 is ", $stats->pi(\@inds),"\n";
}
# pretty base data has 3 columns
# Site
# Individual
# Allele
__DATA__
01 A01 A
01 A02 A
01 A03 A
01 A04 A
01 A05 A
02 A01 A
02 A02 T
02 A03 T
02 A04 T
02 A05 T
04 A01 G
04 A02 G
04 A03 C
04 A04 C
04 A05 G
05 A01 T
05 A02 C
05 A03 T
05 A04 T
05 A05 T
11 A01 G
11 A02 G
11 A03 G
11 A04 A
11 A05 A

01 out G
02 out A
04 out G
05 out T
11 out G
```

Waterson's theta - `theta`

Tajima's D can be calculated with the function `tajima_D` which calculates the D statistic for a set of individuals. These can be provided as `Bio::PopGen::Population` objects or as an arrayref of `Bio::PopGen::Individuals`.

The companion function `tajima_D_counts` can be called with just the number of samples (N), number of segregating sites (n), and the average number of pairwise differences (pi) in that order.

Fu and Li's D can be calculated with the function `fu_and_li_D` which calculates D statistic for a set of individuals and an outgroup. The function takes 2 arguments both of which can be either an arrayref of `Bio::PopGen::Individual` objects or a `Bio::PopGen::Population` object. The outgroup is used to determine which mutations are derived or ancestral. Additionally if the number of external mutations is known they can be provided as the second argument instead of a `Population` object or arrayref of `Individuals`.

The companion method `fu_and_li_D_counts` allows one to just provide the raw counts of the number of samples (N) number of segregating sites (n)and number of external mutations (n_e).

Fu and Li's D^* can be calculated with the function `fu_and_li_D_star` calculates the D* statistics using the number of samples, singleton mutations (mutations on external branches) and total number of segregating sites. It takes one argument which is either an array reference to a set of `Bio::PopGen::Individual` objects (which all have a set of Genotypes with markers of the same name) OR it takes a `Bio::PopGen::Population` object which itsself is just a collection of `Individuals`.

The companion method `fu_and_li_D_star_counts` can be called with just the raw numbers of samples (N), site (n), and singletons (n_s) as the arguments (in that order).

Fu and Li's F can be calculated with the function `fu_and_li_F` and calculates the F statistic for a set of individuals and an outgroup. The function takes 2 arguments both of which can be either an arrayref of `Bio::PopGen::Individual` objects or a `Bio::PopGen::Population` object. The outgroup is used to determine which mutations are derived or ancestral. Additionally if the number of external mutations is known they can be provided as the second argument instead of a `Population` object or arrayref of `Individuals`.

The companion method `fu_and_li_F_counts` can be called with just the raw numbers of samples (N), average pairwise differences (pi), number of segregating sites (n), and the number of external mutatiosn (n_e) as the arguments (in that order).

*Fu and Li's F** can be calculated with the `fu_li_F_star` and calculates the F^* statistic for a set of individuals. The function takes one argument an arrayref of `Bio::PopGen::Individual` or a `Bio::PopGen::Population` object.

The companion method `fu_and_li_F_star_counts` can be called with just the raw numbers of samples (N), average pairwise differences (pi), number of segregating sites (n), and the number of singleton mutations (n_s) the arguments (in that order).

Linkage Disequilibrium *composite_LD* from Weir

7. Population Statistics using `Bio::PopGen::PopStats`

Wright's F_{st} can be calculated for populations using the `Fst` in `Bio::PopGen::PopStats`.

```
use Bio::PopGen::PopStats;
# @populations - are the sets of Bio::PopGen::Population
# objects
# @markernames - set of Marker names to use in this analysis
```

```
my $fst = $stats->Fst(\@populations,\@markernames);
```

8. Coalescent Simulations

The `Bio::PopGen::Simulation::Coalescent` module provides a very simple coalescent simulation. It builds a tree with individual.

Some very simple usage is to generate a few random coalescents and calculate some summary statistics. We separate the topology generation from throwing the mutations down on the tree. So depending on your question, you may want to generate a bunch of different topologies with mutations thrown down randomly on them. Or if you want to look at a single topology with mutations thrown down randomly many different times.

```
use Bio::PopGen::Simulation::Coalescent;
use Bio::PopGen::Statistics;
# generate 10 anonymous individuals
my $sim = Bio::PopGen::Simulation::Coalescent->new(-sample_size => 10);
# generate 50 different coalescents, each with
# potentially a different topology and different mutations
# Let's throw down 12 mutations
my $NumMutations = 12;
my @coalescents;
for ( 1..50 ) {
    my $tree = $sim->next_tree;
    $sim->add_Mutations($tree,$NumMutations);
    # we'll pull off the tips since that is all we want out of the
       # coalescent for summary statistics
    push @coalescents,  [ $tree->get_leaf_nodes];
}
# for each of these coalescents we can then calculate various statistics
my $stats = Bio::PopGen::Statistics->new;
for my $c ( @coalescents ) {
    printf "pi=%.3f theta=%.3f Tajima's D=%-6.3f Fu and Li's D*=%-6.3f ",
 $stats->pi($c), $stats->theta($c), $stats->tajima_D($c),
 $stats->fu_and_li_D_star($c);

    printf "Fu and Li's F*=%-6.3f\n", $stats->fu_and_li_F_star($c);
}

print "Stats for a single topology but mutations thrown re-down\n";
# if you wanted to look at just one topology but mutations thrown
# down many times

my $tree = $sim->next_tree;
for ( 1..50 ) {
    $sim->add_Mutations($tree,$NumMutations);
    my $c = [ $tree->get_leaf_nodes];
    printf "pi=%.3f theta=%.3f Tajima's D=%-6.3f Fu and Li's D*=%-6.3f ",
    $stats->pi($c), $stats->theta($c), $stats->tajima_D($c),
    $stats->fu_and_li_D_star($c);

    printf "Fu and Li's F*=%-6.3f\n", $stats->fu_and_li_F_star($c);
}
```

Bibliography

[1] “ Disentangling the effects of demography and selection in human history ”. Jason E Stajich and Matthew W Hahn. “Mol Biol Evol”. Copyright © 2005 . 22(1):63-73.

[2] “ Population genetic and phylogenetic evidence for positive selection on regulatory mutations at the factor VII locus in humans ”. Matthew W Hahn, Matthew V Rockman, Nicole Soranzo, David B Goldstein, and Greg A Wray. “Genetics”. Copyright © 2004 . 167(2):867-77.

[3] “ Positive selection on a human-specific transcription factor binding site regulating IL4 expression ”. Matthew V Rockman, Matthew W Hahn, Nicole Soranzo, David B Goldstein, and Greg A Wray. “Current Biology”. 13(23):2118-23. Copyright © 2003 .