PyPedal: Pedigree Analysis Software Written in the Python Programming Language

John B. Cole, PhD

April 2, 2004

Abstract

PyPedal is a software package for animal pedigree analysis. It is intended for use by research and consulting geneticists. PyPedal is designed for exploratory data analysis on small to mediumsized data sets and calculates measures of allelic and genotypic diversity, including average coefficients of inbreeding and relationship, effective number of founders, and effective number of ancestors.

1 Overview

PyPedal (**P**ython **Ped**igree An**al**ysis) is a tool for analyzing animal pedigree files. It calculates several quantitative measures of allelic and genoytpic diversity from pedigrees, including average coefficients of inbreeding and relationship, effective number of founders, and effective number of ancestors. Some qualitative checks are performed in order to catch some common mistakes, such as parents with more recent birthdates or ID numbers than their offspring.

Routines are also provided for the decomposition of A and the direct formation of A^1 with and without taking account of inbreeding. These are of academic interest rather than practical interest, but if a simple script is needed for the inversion of a reasonablysized pedigree PyPedal is quite adequate to the task.

PyPedal is a Python language module that may be called by other Python programs or used interactively with the command line in the Python interpreter. The Numerical Python module (http://www.pfdubois.com/numpy/) module is required by PyPedal, and is included with most Linux distributions.

1.1 Obtaining and Installing PyPedal

PyPedal may be obtained from the author's website at http://www.funjackals.com/ (select "Software" from the menu on the left). For *nix users installation is as simple as untar-and-gunzipping the download and moving the resulting files to a directory that is in your `$PYTHONPATH`. Windows users have an even better time of it – there is a snazzy GUI installer courtesy of Distutils.

1.2 Implemented Features

PyPedal is currently capable of the following operations:

- Reading pedigree files in several formats;
- Checking pedigree integrity (duplicate IDs, parents younger than offspring, etc.);
- Generating summary information such as frequency of appearance in the pedigree file;
- Computation of the numerator relationship matrix (A) from a pedigree file using the tabular method;
- Inbreeding calculations for large pedigrees is provided courtesy of VanRaden's (1992) recursive algorithm;
- Computation of average total and average individual coefficients of inbreeding and relationship;
- Decomposition of A into T and D such that $A = TDT'$;
- Computation of the direct inverse of A (not accounting for inbreeding) using the method of Henderson (1976);
- Computation of the direct inverse of A (accounting for inbreeding) using the method of Quaas (1976);
- Storage of A and its inverse between user sessions as persistent Python objects using the pickle module to avoid unnecessary calculations;
- Computation of effective founder number using the exact algorithm of Lacy (1989);
- Computation of effective founder number using the approximate algorithm of Boichard et al. (1996);
- Computation of effective ancestor number using the algorithm of Boichard et al. (1996);
- Output to ASCII text files, including matrices, coefficients of inbreeding and relationship, and summary information;
- Reordering and renumbering of pedigree files.

1.3 Planned Features

The following features are not yet implemented in PyPedal, but will probably be added in a future release:

- Direct calculation of the inverse of A accounting for inbreeding using the method of Luo and Meuwissen;
- Calculation of theoretical effective population size;
- Calculation of actual effective population size based on the change in population average inbreeding;
- Calculation of some measure of effective family number (inspired by a post of D. Gianola's to the Animal Geneticists Discussion Group email list on 30 January 2001);
- Representation of pedigrees as an algebraic structure (i.e. graphs);
- Identification of disconnected subgroups (if any) in a pedigree;
- Fast operations on graphs;

1.4 Input Files

PyPedal currently reads comma separated value (CSV) pedigree files. Comment lines may be inlcuded in the pedigree file by preceding a line with a hash mark (#) symbol. Users must provide a string in the pedigree file that describes the data found in the file. The format string is preceded by a percent (%) symbol and consists of lowercase letters. For example, `%asd` is the code used to indicate that the file contains three columns, corresponding to animal ID, sire ID, and dam ID, respectively. The available codes are:

- a animal ID number
- s sire ID number
- d dam ID number
- x sex of the animal
- b birth year of the animal
- f coefficient of inbreeding of the animal

At the moment, all formats must begin with the triplet 'asd', and all pedigree files must contain animal, sire, and dam IDs in their first three columns. In addition, they must appear in the format string IN THE ORDER IN WHICH THEY APPEAR IN THE LIST ABOVE. For example, a file with four columns, consisting of animal,sire, and dam IDs plus the coefficient of inbreeding of the animal, would have the format string `% asdf`. A five-column file of animal,

sire, dam, sex, coefficient of inbreeding will ALWAYS be written as `% asdxf` and **never** as `% asdfx`. In future revisions, the use of regular expressions will probably eliminate the order dependency in the format string.

A pedigree file may be written as follows:

```
# Sample pedigree file for John's Jerseys.
# '0' indicates an unknown parent.
% asdx
1,0,0,m
2,0,0,f
3,1,2,m
```

Sex codes may be written as either uppercase or lowercase "F"s or "M"s. Any other character is ignored and the sex inferred from progeny records or set to "U" for unknown.

2 Application Programming Interface (API)

Automatically-generated API documentation is provided as HTML files in the source code distribution. If you are a Windows user you can install PyPedal using the GUI installer but you must also download and unzip the source code distribution in order to access the documentation. There is an HTML file for each `.py` file in the distribution that contains an API for the procedures in that file. NOTE: the HTML files in the source code distribution are MORE CURRENT than this document. You should consult the online API as well as this written document.

2.1 Notes on Parameters

Some notes are in order with respect to the parameters required by functions detailed in the API. The *inputfilename* parameter refers to a file from which a pedigree should be read.

The *pedigreehandle* parameter refers to a named list of Animal() objects. Animal() objects are created from the records in the pedigree file when the **preprocess()** function is called. *filetags* are descriptive strings prepended to output file names for easy identification. *a* parameters refer to A matrices created by a call to the **a_matrix()** function. A sample program illustrating the use of these parameters may be found in the appendix.

2.2 preprocess(*inputfilename*)

The **preprocess()** function performs two key functions. First, reads the contents of the pedigree file and creates a list of Animal() objects if the record format code is valid. Second, it performs some simple checks on pedigree integrity to make sure that parents are older than offspring, ID codes are unique, etc. The resulting list of Animal()s is referred to as a pedigree, and is returned

by the function. Multiple pedigrees may be open at the same time as long as the system running the software has enough RAM and disc space to store everything in memory.

2.3 ped_summary(*pedigreehandle,filetag*)

ped_summary() produces summary statistics on the pedigree file such as number of sires, number of dams, founders, and most frequently appearing individuals by sex. Output is written to a file named *filetag*_summary_.dat.

2.4 a_matrix(*pedigreehandle,filetag*)

Output is written to a file named *filetag*_summary_.dat.

2.5 a_matrix_to_file(pedigreehandle,filetag,a)

2.6 a_matrix_from_file(inputfilename)

2.7 a_decompose(*pedigreehandle,filetag,a*)

Output is written to a file named *filetag*_summary_.dat.

2.8 form_d_nof(*pedigreehandle*)

2.9 a_inverse_dnf(*pedigreehandle,filetag*)

Output is written to a file named *filetag*_summary_.dat.

2.10 a_inverse_df(*pedigreehandle,filetag*)

Output is written to a file named *filetag*_summary_.dat.

2.11 a_inverse_to_file(*pedigreehandle,filetag,a*)

2.12 a_inverse_from_file(inputfilename)

2.13 a_coefficients(*pedigreehandle,filetag,a*)

Output is written to a file named *filetag*_summary_.dat.

2.14 a_effective_founders_lacy(*pedigreehandle,filetag,a*)

Output is written to a file named *filetag*_summary_.dat.

2.15 a_effective_founders_boichard(*pedigreehandle,filetag,a*)

Output is written to a file named *filetag*_summary_.dat.

2.16 a_effective_ancestors_boichard(*pedigreehandle,filetag,a*)

Output is written to a file named *filetag*_summary_.dat.

3 Methodology

This needs to be written.

Appendix

The following sample program demonstrates the use of all of the functions detailed in the API. Some of the functions are not useful for daily analysis, but their use is demonstrated for the sake of completeness.

```
#########################################################
# Sample program to demonstrate the use of PyPedal
#########################################################

# we must import the functions in the pypedal modules
from PyPedal import *

# read the pedigree file, create a list of Animal()
# objects, and return the pedigree, which will be
# called 'lacy'

lacy = preprocess('lacy.ped')
lacy_a = a_matrix(lacy,'lacy')  # form the numerator relationship
                                # matrix, A, from the pedigree
                                # 'lacy'

l_t,l_d = a_decompose(lacy,'lacy',lacy_a)  # decompose A into matrices D and
                                           # T such that A=TDT'; a tuple of
                                           # the form {T,D} is returned.

l_dnf = a_inverse_dnf(lacy,'lacy')  # form the direct inverse of
                                    # A not accounting for inbreeding

l_df = a_inverse_df(lacy,'lacy')    # form the direct inverse of
                                    # A accounting for inbreeding

a_coefficients(lacy,'lacy',lacy_a)  # compute average coefficients
                                    # of inbreeding and relationship
                                    # and individual coefficients of
                                    # inbreeding
```

```
a_effective_founders_lacy(lacy,'lacy')  # compute the effective number of
                                        # founders using the method of Lacy

a_effective_founders_boichard(lacy,'lacy')  # compute the effective number of
                                            # founders using the method of
                                            # Boichard et al.

a_effective_ancestors_boichard(lacy,'lacy')  # compute the effective number of
                                             # ancestors using the method of
                                             # Boichard et al.
```

Note that PyPedal programs write most of their output directly to files and write very little output to STDOUT. This is convenient when scripting because you only have to redirect the output to a log file or to /dev/null if all you really want are the output files.