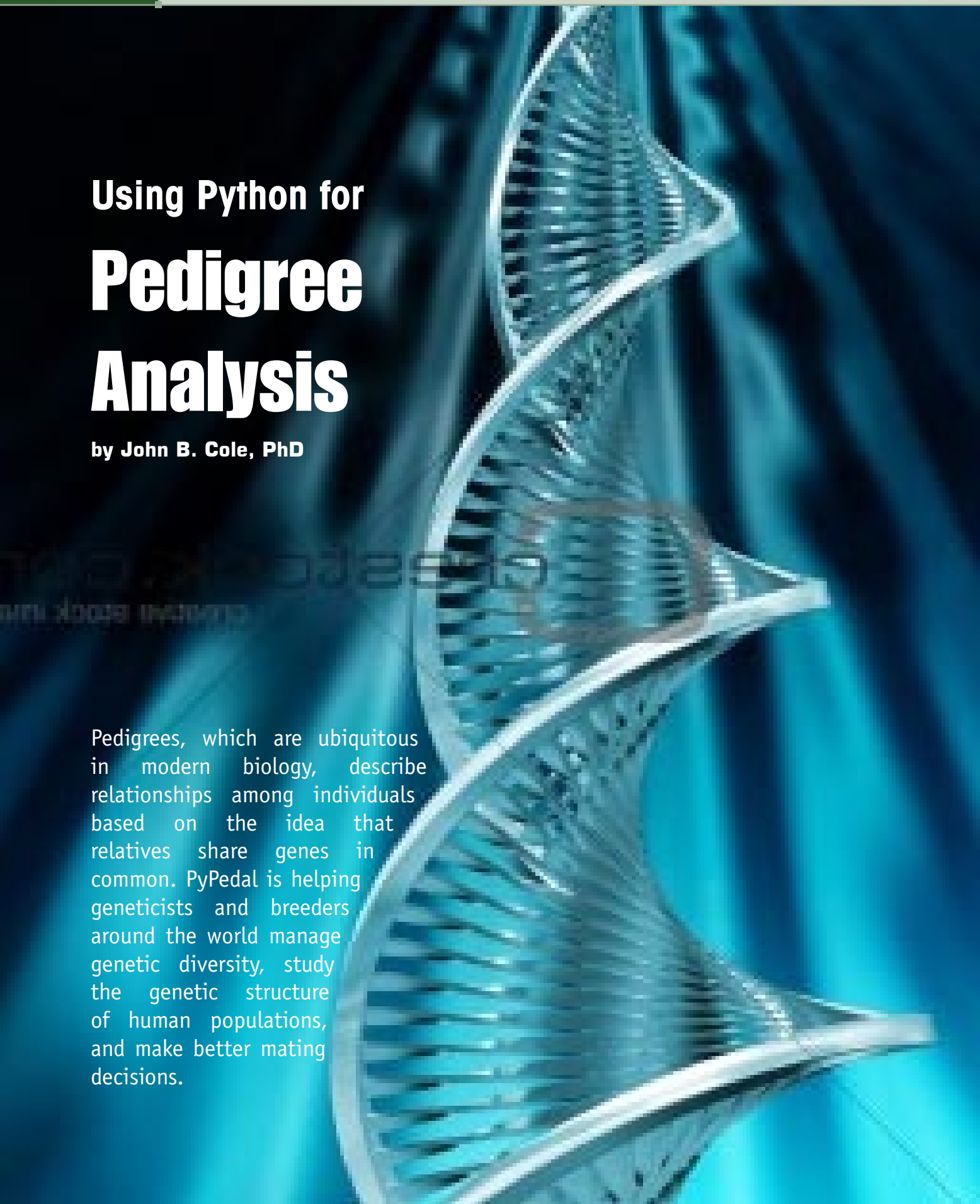
Using Python for Pedigree Analysis

by John B. Cole, PhD



Pedigrees, which are ubiquitous in modern biology, describe relationships among individuals based on the idea that relatives share genes in common. PyPedal is helping geneticists and breeders around the world manage genetic diversity, study the genetic structure of human populations, and make better mating decisions.

A *pedigree* is a way of describing a population of people or animals in terms of genetic relationships among individuals. Pedigrees are of interest to many people, including scientists, animal and plant breeders, and genealogists. They are used to assess the diversity of populations, in combination with performance records in genetic evaluation programs, and to trace the inheritance of beneficial or harmful alleles in a population. Many people also construct pedigrees in conjunction with an interest in academic or familial lineages.

Related individuals resemble one another more closely than unrelated individuals because they share genes in common. As noted above, pedigrees are a tool for describing genetic relationships, and they can be used to calculate the proportion of genes shared by two individuals in a population. Pedigree relationships are based on the fundamental assumption that parents and offspring share one-half of their genes in common. A number of different measures of population structure and diversity can be constructed from that assumption.

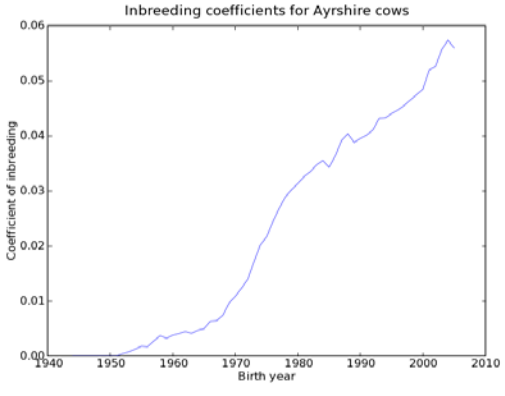
Human pedigrees are typically used to determine the risk that a child will inherit an undesirable recessive condition of which both parents are carriers, and to document family histories. Animal pedigrees are commonly used to trace the flow of genes through a population in order to assess levels of genetic diversity, and to calculate breeding values for use in genetic selection programs.

PyPedal was originally developed as a tool to support my research and as a learning tool, supposing that the best way to understand a calculation is to implement it in software. At the time there were (and still are) a number of commercial and free packages available for working with pedigrees. These packages can be divided into two broad groups: pedigree management software for genealogists and animal (e.g., horse and dog) breeders, and pedigree analysis software for use by

LISTING 1

```
#!/usr/bin/python

######################################################################
# NAME: new_hartl.py
# VERSION: 2.0.0b12 (15MAY2060)
# AUTHOR: John B. Cole, PhD (jcole@aipl.arsusda.gov)
# LICENSE: LGPL
######################################################################

from PyPedal import pyp_graphics
from PyPedal import pyp_newclasses
from PyPedal import pyp_nrm

if __name__ == '__main__':

    # Load the pedigree using the options defined in the file
    # new_hartl.ini.
    example = pyp_newclasses.loadPedigree(optionsfile='new_hartl.ini')

    # Calculat coefficients of inbreeding for the individuals in the
    # example pedigree.
    example_inbreeding = pyp_nrm.inbreeding(example)
    print example_inbreeding
```

FIGURE 1



REQUIREMENTS

PYTHON: *2.4*

Other Software:

- PyPedal - http://pypedal.sourceforge.net
- ADODB - http://phplens.com/lens/adodb/adodb-py-docs.htm
- elementree - http://effbot.org/zone/element-index.htm
- Graphviz - http://www.graphviz.org
- matplotlib - http://matplotlib.sourceforge.net
- NetworkX - http://networkx.lanl.gov
- NumPy - http://www.numpy.org
- pydot - http://code.google.com/p/pydot
- PyGraphviz - http://networkx.lanl.gov/pygraphviz
- PyParsing - http://pyparsing.wikispaces.com
- Python Imaging Library - http://effbot.org/zone/pil-index.htm
- ReportLab - http://www.reportlab.org

Related Links:

- Animal Improvement Programs Laboratory, ARS, USDA - http://www.aipl.arsusda.gov
- Psyco - http://psyco.sourceforge.net/
- The Seeing Eye, Inc. - http://www.seeingeye.org

scientists and genetic counselors. There is some overlap between the two groups – e.g., both sorts of package often calculate coefficients of inbreeding – but there tend to be substantial differences in both usability (graphical versus command line interfaces) and function. One important point is that commercial pedigree management systems tend to be standalone packages, while scientific packages often consist of collections of tools which can be easily scripted, in keeping with the Unix philosophy. Over time, PyPedal evolved to incorporate features of both sorts of program.

Representing animals and pedigrees

Python was chosen over other programming languages such as Fortran (Pedig), Visual Basic (ENDOG), and Visual C++ (CFC) because of its support for procedural and object-oriented programming paradigms, its rich data structures, the availability of third-party libraries, speed of development, and support for the Linux operating system. PyPedal performs well on pedigrees of hundreds to thousands of animals, and is capable of processing pedigrees of hundreds-of-thousands or millions of records; Figure 1 shows the change in inbreeding over time for approximately 600,000 Ayrshire dairy cattle. Memory can be an issue for extremely large datafiles, as pedigrees are stored entirely in RAM for processing.

Input pedigrees are read from ASCII flat files or database tables, and loaded into pedigree objects.

LISTING 2

```
# new_hartl.ini
messages = verbose
pedfile = hartlandclark.ped
pedname = Pedigree from van Noordwijck and Scharloo (1981)
pedformat = asdb
pedigree_is_renumbered = 1
```

LISTING 3

```
# Pedigree from van Noordwijck and Scharloo (1981) as presented
# in Hartl and Clark (1989), p. 242.
1 0 0 1900
2 0 0 1900
3 0 0 1900
4 1 2 1910
5 1 2 1910
6 3 4 1920
7 3 4 1920
8 5 0 1930
9 6 0 1930
10 7 0 1930
11 8 0 1930
12 9 11 1940
13 7 12 1950
14 10 11 1960
15 13 14 1970
```

Pedigrees may also be simulated or read from directed graphs. Heuristics are used to improve data completeness when minimal information is provided; for example, PyPedal can infer the sex of individuals if that information is not included in the pedigree file. Pedigree objects contain a list of instances of animal objects, a pedigree metadata object, and an optional numerator relationship matrix (NRM). The NRM, which describes the proportion of genes shared by two individuals, may be stored in the pedigree to avoid repeating time-consuming calculations at the cost of greater memory consumption. Metadata are collected when a pedigree is loaded and are used by other routines to avoid unnecessary pedigree traversal. Pedigree objects are passed by reference to procedures in PyPedal modules; NRM are instances of NumPy matrix objects, which are densely stored.

As an example, we are going to consider the pedigree of a population of Great Tits, *Parus major*, described by Noordwijk and Scharloo (1981). The pedigree file, shown in Listing 3, contains a record for each bird in the pedigree. By default, an ID of *0* indicates an

LISTING 4

```
[INFO]: Logfile hartlandclark.log instantiated.
[INFO]: Preprocessing hartlandclark.ped
[INFO]: Opening pedigree file hartlandclark.ped
  [INFO]: Renumbering pedigree at Tue Feb 17 13:11:59 2009
    [INFO]: Reordering pedigree at Tue Feb 17 13:11:59 2009
    [INFO]: Renumbering at Tue Feb 17 13:11:59 2009
    [INFO]: Updating ID map at Tue Feb 17 13:11:59 2009
  [INFO]: Assigning offspring at Tue Feb 17 13:11:59 2009
[INFO]: Creating pedigree metadata object
  [INFO]:  Instantiating a new PedigreeMetadata() object...
  [INFO]:  Naming the Pedigree()...
  [INFO]:  Assigning a filename...
  [INFO]:  Attaching a pedigree...
  [INFO]:  Setting the pedcode...
  [INFO]:  Counting the number of animals in the pedigree...
  [INFO]:  Counting and finding unique sires...
  [INFO]:  Counting and finding unique dams...
  [INFO]:  Setting renumbered flag...
  [INFO]:  Counting and finding unique generations...
  [INFO]:  Counting and finding unique birthyears...
  [INFO]:  Counting and finding unique founders...
  [INFO]:  Counting and finding unique herds...
  [INFO]:  Detaching pedigree...
Metadata for Pedigree from van Noordwijck and Scharloo (1981)
(hartlandclark.ped)
  Records:    15
  Unique Sires:  9
  Unique Dams:    5
  Unique Gens:    1
  Unique Years:  8
  Unique Founders:  3
  Unique Herds:  1
  Pedigree Code:    asdb
[DEBUG]: method =  dense
{'fx': {1: 0.0, 2: 0.0, 3: 0.0, 4: 0.0, 5: 0.0, 6: 0.0, 7: 0.0, 8: 0.0,
        9: 0.0, 10: 0.0, 11: 0.0, 12: 0.015625, 13: 0.078125,
        14: 0.015625, 15: 0.14453125},
 'metadata': {'nonzero': {'f_max': 0.14453125, 'f_avg': 0.0634765625,
                          'f_rng': 0.12890625, 'f_sum': 0.25390625,
                          'f_min': 0.015625, 'f_count': 4},
              'all': {'f_max': 0.14453125, 'f_avg': 0.016927083333333332,
                      'f_rng': 0.14453125, 'f_sum': 0.25390625,
                      'f_min': 0.0, 'f_count': 15}}}
```

unknown parent (missing or unknown parents are common in animal pedigrees). This pedigree has the format **asdb**, indicating that the columns contain identification numbers for individuals, their father, and their mother (subsequently referred to as animals, sires, and dams, respectively), as well as the animal's birth year. Pedigree files must contain animal, sire, and dam IDs, and may contain a number of other data (eighteen different kinds, as of this writing). This pedigree can be loaded with two lines of code (shown in context in Listing 1):

```
>>> from PyPedal import *
>>> example = pyp_newclasses.loadPedigree(
...     optionsfile='new_hartl.ini')
>>> example
<PyPedal.pyp_newclasses.NewPedigree object at 0x...>
```

Using `from PyPedal import` should not pollute the namespace too badly — you still need to prefix the appropriate module name to access PyPedal functions. The call to `loadPedigree()` instantiates a `NewPedigree` object based on the settings in the parameter file (Listing 2) and populates it with `NewAnimal` instances which correspond to the data in the pedigree file (Listing 3). By default, PyPedal provides verbose output (Listing 4). You must pass either a dictionary of program options or a parameter file name to `loadPedigree()`, and the options file or dictionary must include `pedformat` and `pedfile` entries:

```
>>> example2 = pyp_newclasses.loadPedigree()
[ERROR]: pyp_newclasses.loadPedigree() was unable to
instantiate and load the pedigree
>>> options = { 'pedfile':'hartlandclark.ped', \
... 'pedformat':'asdb' }
>>> example2 = pyp_newclasses.loadPedigree(options)
>>> example2
<PyPedal.pyp_newclasses.NewPedigree object at 0x...>
```

If you do not have a real pedigree you may simulate one, and a number of options are provided to produce pedigrees with structures of interest. Populations may be closed or open to migration, the ratio of males to females can be changed from the default of 50:50, the number of founders of each sex and number of generations specified, and parent-offspring and full-sib matings can be allowed. Simulated datasets are useful for studying pedigrees with tools commonly used in network analysis. Pedigrees are directed graphs, and network analysis may provide new tools for identifying influential animals and members of outgroups for better management of genetic diversity. The following example code simulates a pedigree of 25 individuals:

```
options = {}
options['simulate_pedigree'] = 1
options['simulate_n'] = 25
options['pedname'] = 'Simulated Pedigree'
ped = pyp_newclasses.loadPedigree(options)
```

Note that you do not need to provide `pedformat` or `pedfile` parameters when you simulate a pedigree. All simulated pedigrees have the format code **asdxg**, which specifies a pedigree with animal, sire, and dam IDs, the animal sex, and a generation code.

> A single bug report is far more valuable to the user experience than a hundred testimonials as to the brilliance of PyPedal.

Plain ASCII text formats are used for most input (pedigree) and output (results) files, although some graphics and matrix routines write binary files. Animal IDs may be provided as either integers or strings; strings are hashed to integers internally. Comments and user-specified column delimiters may be included in pedigree files. Pedigree errors — including duplicate animal IDs, animals appearing as both sires and dams, animals older than their parents, and animals with the same ID as a parent — are detected and the user notified. Pedigree records are automatically generated for animals that appear only as parents. The Python `logging` module is used to create log files when a PyPedal program is run. Program options, such as the pedigree format code, may be set in the program or read from an options file. A pedigree format code and pedigree file name must be provided; additional parameters may be provided to override defaults. For example, the `set_sexes` option enables the sex-inference heuristic, `renumber` requests that the pedigree be reordered and renumbered, and `pedcomp` indicates that

pedigree completeness (also known as the ancestor loss coefficient) should be calculated for each animal in the pedigree.

Pedigrees may also be loaded from, and stored to, MySQL, PostgreSQL, and SQLite databases using the Python bindings to the ADODB database abstraction library, which is included in the PyPedal distribution. The data are stored in a table keyed on animal ID, and include sire and dam information, measures of genetic variation such as inbreeding and pedigree completeness, and demographic information that includes sex, birth date, and founder status. Loading a pedigree from a database is similar to reading it from a file, but you need to provide some database-specific options. The following code will load a pedigree from an SQLite database (the default database engine):

```
options = {}
options['database_name'] = 'test_pypedal_save'
options['dbtable_name'] = 'test'
options['pedfile'] = ''
example = pyp_newclasses.loadPedigree(options,
    pedsource='db')
```

In the current version you must provide a pedfile parameter, even if it's an empty string.

Finally, PyPedal can import data stored in the GEDCOM format, which is commonly used in human genealogy, and can export pedigrees to that format. Only a subset of the format is supported, but it should be sufficient for getting pedigree data into PyPedal for calculations not typically supported in genealogy software.

```
options = {}
options['pedfile'] = 'ged3.ged'
options['pedname'] = 'European Royalty pedigree'
options['pedformat'] = 'ASD'
test = pyp_newclasses.loadPedigree(options,
    pedsource='gedcomfile')
```

Not all elements in the GEDCOM standard are supported, and if you import and then export a pedigree using PyPedal then you will lose data from the original pedigree. The data that are lost are important to genealogists, but are not important for the sorts of calculations performed by PyPedal.

Working with pedigrees

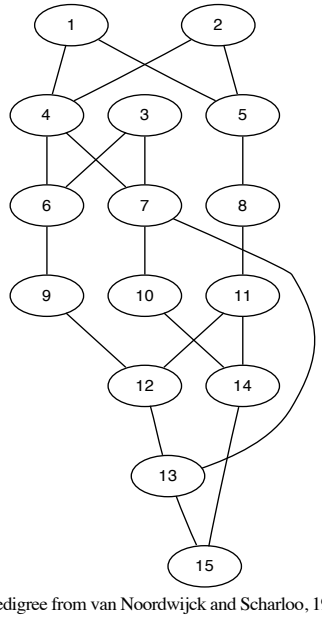
Routines for calculating a number of measures of genetic variation are included in PyPedal, including effective founder numbers and founder genome equivalents (Lacy, 1989), effective ancestor numbers (Boichard et al., 1997), average coefficients of inbreeding and relationship (Wright, 1922), theoretical and realized effective population sizes (Falconer and MacKay, 1996), and pedigree completeness (Cassell et al., 2003).

Founder alleles are simulated and segregated through the pedigree to calculate the effective number of founder genomes (MacCluer et al., 1986), but molecular data (e.g., DNA marker data) are not otherwise utilized. Routines that return values for each animal in the pedigree also return summary statistics such as means, minima, and maxima. Tools are also provided for calculating the additive relationship between two individuals, calculating the inbreeding of a given mating (coefficient of kinship), identifying common ancestors, and calculating generation lengths and generation intervals. Results are returned in dictionaries that are easily passed to other routines for additional computation, plotting, or reporting. Most routines also write results to a file automatically.

Coefficients of relationship and inbreeding are calculated using the method of VanRaden (1992) in which pedigrees for individual animals are extracted from the full pedigree and relationship matrices calculated using the tabular method (Emik and Terrill, 1949). Diagonals may be adjusted for the inbreeding of the parents. Inverse NRM ignoring or accounting for inbreeding are formed directly using the methods of Henderson (1976) and Quaas (1976).

The pyp_db module uses the ADODB library for

FIGURE 2



Pedigree from van Noordwijck and Scharloo, 1981

working with relational databases. PyPedal pedigrees can be stored in a database and accessed using command line tools or bindings to many different programming languages, which is of great value to the user in that data are not bound to a particular application or proprietary data storage format. In conjunction with the `pyp_reports` module, which allows users to create reports in Adobe's Portable Document Format, users have the tools to easily define custom reports. Basic reports are provided in the `pyp_reports` module, and the `pyp_reports` template module provides a template for use in writing custom reports.

Demonstrating pedigree analysis

The easiest way to demonstrate the abilities of PyPedal is by example, so I will again consider the Great Tit pedigree. Recall that we can load the pedigree by calling `loadPedigree()`. You will see a series of messages telling you about the pedigree that's being loaded, including a summary listing, among other things, the number of animals in the pedigree, the format code (see Listing 4 for complete output).

```
>>> from PyPedal import *
>>> example = pyp_newclasses.loadPedigree(
...                     optionsfile='new_hartl.ini')
[INFO]: Logfile hartlandclark.log instantiated.
[INFO]: Preprocessing hartlandclark.ped
[INFO]: Opening pedigree file hartlandclark.ped
[INFO]: Renumbering pedigree at Tue Jul 29 14:49:49
2008
[cut like the author from junior high soccer]
Metadata for Pedigree from van Noordwijck and
Scharloo (1981) (hartlandclark.ped)
        Records:                15
        Unique Sires:           9
        Unique Dams:            5
        Unique Gens:            1
        Unique Years:           8
        Unique Founders:        3
        Unique Herds:           1
        Pedigree Code:          asdb
```

In the preceding output, you will see a message that the pedigree is being renumbered, and it is important to understand why we do this. Pedigrees are reordered such that parents always precede their children, and they are renumbered so that arbitrary animal IDs are replaced with IDs that can be used as list indices. Dictionaries that map between original and renumbered IDs, as well as between original and renumbered animal names, are provided as attributes of `NewPedigree` objects, but are not discussed further here. Note that PyPedal does not renumber pedigrees by default, so that you can load pedigrees that are already renumbered. The pedigree in Figure 2 did not require renumbering, and the reordering function agreed, so we do not need to worry about renumbered versus original IDs when referring to this particular output.

Now that the preliminaries are out of the way, we can ask some interesting questions. For example, how inbred are these birds, that is, do they have lots of genes that came from the same ancestor? Are they closely-related to one another, that is, do two animals share lots of genes in common because they share mutual ancestors? Let's find out by calling the `inbreeding` function from `pyp_nrm`, which returns one dictionary that includes individual coefficients of inbreeding (COI) as well as summary statistics for the population (`example_inbr`) and a second that includes summary statistics for coefficients of relationship (`example_rels`). A coefficient of inbreeding is the probability that the two alleles at an arbitrary location in the genome are identical because they came from the same ancestor. Inbreeding has harmful effects on vitality, fertility, and productivity, but is an inevitable consequence of genetic selection. Coefficients of relationship measure the proportion of genes that two animals share in common because they're related. The relationship between a parent and child is 50% (0.50) because individuals receive half of their genes from each parent.

Summary statistics about relationships among animals in the population are contained in `example_rels`, including counts, sums, and averages for all relationships and for only non-zero relationships. In populations with many unrelated animals the relationship matrix is sparse (contains mostly zeroes) and the average relationship is heavily influenced by all of the zero values.

```
>>> example_inbr, example_rels = pyp_nrm.inbreeding(
...     example,method='vanraden', rels=1)
>>> example_rels
{'r_nonzero_count': 98,
 'r_nonzero_avg': 0.27838010204081631,
 'r_nonzero_sum': 27.28125, 'r_min': 0.0,
 'r_sum': 54.5625, 'r_avg': 0.45468750000000002,
 'r_max': 0.68359375, 'r_count': 120,
 'r_rng': 0.68359375}
```

The inbreeding dictionary, which is probably the most complex object returned by a PyPedal function, contains two dictionaries: `fx`, which includes the COI for each animal in the pedigree, and `metadata`, which

includes dictionaries of summary statistics for all animals in the pedigree (all), and for only the inbred animals (nonzero).

```
>>> example_inbr
{'fx': {1: 0.0, 2: 0.0, 3: 0.0, 4: 0.0, 5: 0.0,
  6: 0.0, 7: 0.0, 8: 0.0, 9: 0.0, 10: 0.0, 11: 0.0,
  12: 0.015625, 13: 0.078125, 14: 0.015625,
  15: 0.14453125},
 'metadata': {'nonzero': {'f_max': 0.14453125,
   'f_avg': 0.0634765625, 'f_rng': 0.12890625,
   'f_sum': 0.25390625, 'f_min': 0.015625,
   'f_count': 4},
  'all': {'f_max': 0.14453125,
   'f_avg': 0.016927083333333332,
   'f_rng': 0.14453125, 'f_sum': 0.25390625,
   'f_min': 0.0, 'f_count': 15}}}
```

One of the useful things you can do easily with PyPedal is ask what-if questions. We animal breeders routinely ask questions such as, "What would the COI be if we mated birds 14 and 15?" We can easily obtain that answer:

```
>>> pyp_metrics.mating_coi(14, 15, example)
0.326171875
```

We wouldn't normally expect to see a COI that high, so let's look and see if the two animals share any pedigree in common.

```
>>> pyp_metrics.common_ancestors(14, 15, example)
[1, 2, 3, 4, 5, 7, 8, 10, 11, 14]
```

It is easy to see from Figure 2 that bird 14 is bird 15's father, but in pedigrees with dozens or hundreds of animals it is often difficult to see how closely two animals are related from looking at a plot. Dog guide pedigrees contain thousands of records, and dairy cattle pedigrees may contain millions of animals.

Livestock breeders are used to looking at COI as a measure of genetic diversity, but there are many more ways to measure diversity, some of them better-suited to small populations than COI. Consider Lacy's (1989) effective founder number, which is a measure of how founders (the animals at the beginning of a pedigree) have contributed to the current population. Our bird pedigree contains three founders (birds 1, 2, and 3). Have they made equal genetic contributions to the current generation?

```
>>> pyp_metrics.effective_founders_lacy(example)
Effective founder number (f_e): 3.0
```

In this case, the answer is, "yes", but in many populations the effective founder number is much smaller than the number of actual founders. The function shown above returns a dictionary (not shown) containing the effective founder number and a few other pieces of information.

Finally, the pedigree shown in Figure 2 can be produced with a single line of code:

```
>>> pyp_graphics.draw_pedigree(example,
...     gfilename='hartlandclark',
...     gtitle='van Noordwijck and Scharloo (1981)',
...     gorient='p', gname=0, gdirec='',
...     gfontsize=12, garrow=0)
```

PyPedal's users, and their requests

PyPedal has been used to solve a number of real-world use problems for both scientists and laypeople, working with a variety of species and to different ends. Some examples include:

- An Australian geneticist working with beef cattle pedigrees used PyPedal to import and renumber pedigrees to create NRM, which were then used for the calculation of optimal contributions and mate selection.

- An Italian geneticist working with human data used PyPedal to calculate coefficients of kinship for 6,000 couples, as well as for all possible combinations of 30,000 individuals, reporting, "I think the 6,000 couples were finished in under an hour. The program we were using would have taken a month for that."

- A dog breeder wrote a web front-end for PyPedal that he uses to compute coefficients of inbreeding for animals in his pedigree database, as well as for hypothetical matings.

Based on feedback from those and other users PyPedal's value is based in large part on its Unix tools philosophy, which allows it to be used as part of a larger tool chain. If PyPedal were a GUI-based program, as many pedigree analysis programs are, it would not as useful to its current users.

One of the advantages of interacting with your users is that they often ask you to add features to the program. I think this is exciting, because I've often found that people use my programs differently than I do, and features that are often easy to add provide the end-user with lots of value. For example, I've put a lot of work into the reports module, but the most-requested feature I've ever added has to be database support.

Initially, PyPedal used SQLite only to create some reports, and users had no direct access to database functionality. Now there is basic support for MySQL, Postgres, and SQLite that users can get their hands on. The key lesson here is that people do not want their data wrapped up in such a way that they cannot easily access them.

Deploying, documenting, optimizing, and extensibility

At present, there is only one developer actively working on the PyPedal source tree, and, while nightly backups are taken, no revision control system is used. The philosophy of "release early, release often" has been followed, which explains the 19 alpha and 23 beta releases, but tarballs and a changes file don't scale well.

The documentation for PyPedal includes a lengthy manual that is part user's manual and part developer's reference, and an automatically generated API. The philosophy underlying the manual is that it should both teach users how to manipulate pedigrees in a manner that's interesting to them, as well as explain to programmers how, and sometimes why, PyPedal works the way it does. For example, there's an entire chapter in the manual devoted to the creation of new printed reports. The manual is typeset using LaTeX, and the API documentation is generated automatically using Doxygen, both of which are excellent tools. The HTML and PDF versions of the manual are created using `latex2html` and `ps2pdf`, respectively. I chose LaTeX over OpenOffice.org because I feel that it provides superior tools for managing bibliographies and creating indexes and tables of contents.

Performance and optimization are complex topics about which I claim no particular expertise, although I take to heart the words of Donald Knuth, who noted in his 1974 Turing Award lecture that "Premature optimization is the root of all evil (or at least most of it) in programming". I'll speak quite plainly in order to avoid any confusion: PyPedal is not optimized code. I have not engaged in code refactoring or algorithm tweaking in order to improve the performance of the package. I have made use of syntactic tools like list comprehensions because they are expressive tools, rather than because they are faster than more rudimentary

References

- Cole, J. B. 2007. PyPedal: A package for pedigree analysis using the Python programming language. *Computers and Electronics in Agriculture* 57:107-113. Available: http://dx.doi.org/10.1016/j.compag.2007.02.002 (Accessed 23 October, 2008).
- Falconer, D. S., and T. F. C. MacKay. 1996. *Introduction to Quantitative Genetics (4th ed.)*. John Wiley & Sons, Inc., New York, NY.
- Henderson, C. R. 1976. A simple method for computing the inverse of a numerator relationship matrix used in prediction of breeding values. *Biometrics* 32:69-83.
- Lacy, R. C. 1989. Analysis of founder representation in pedigrees: founder equivalents and founder genome equivalents. *Zoo Biol.* 8:111-123.
- MacCluer, J. W., J. L. VandeBerg, B. Read, and O. A. Ryder. 1986. Pedigree analysis by computer simulation. *Zoo Biol.* 5:147-160.
- Quaas, R. L. 1976. Computing the diagonal elements of a large numerator relationship matrix. *Biometrics* 32:949-953.
- Sargolzaei, M., H. Iwaisaki, and J.-J. Colleau. 2005. A fast algorithm for computing inbreeding coefficients in large populations. *J. Anim. Breed. Genet.* 122:325-331.
- van Noordwijk, A. J., and W. Scharloo. 1981. Inbreeding in an island population of the Great Tit. *Evolution* 35:674-688.
- VanRaden, P. M. 1992. Accounting for inbreeding and crossbreeding in genetic evaluation of large populations. *J. Dairy Sci.* 75:3136-3144.
- Wright, S. 1922. Coefficients of inbreeding and relationship. *Am. Nat.* 56:330-338.

approaches. However, the alert reader of code may note that I use dictionaries rather than lists in a few places where a list is the obvious data structure to use, because dictionary lookups are fast.

That said, PyPedal clearly could benefit from an extensive code review, careful optimization, and the addition of unit tests, as a few examples may demonstrate. Several of the procedures in the graphics module have accumulated parameters and additional functionality by accretion, and the code is quite messy and difficult to manage. Inbreeding calculations are slow for large pedigrees, although faster algorithms have been described in the literature (e.g., Sargolzaei et al., 2005). There are cases where regressions which could have been prevented by proper unit tests were introduced into releases.

Community and the Future

PyPedal is written entirely in pure Python, which means that it should be easily extensible, although it does use some compiled extensions (e.g., Numpy, ReportLab). It is designed to make use of the Psyco optimizing compiler for Python if it's available. It is possible that the speed of some calculations would be improved by implementing them in a compiled language such as C or Fortran, but some preliminary work with inbreeding calculations suggest that poor algorithms are a bigger issue. Running a poorly-performing algorithm a little faster by using compiled code doesn't solve the real problem. However, those concerns have been far outweighed by the ease and speed of development in Python. I am a scientist by profession, not a programmer, and Python lets me quickly solve problems. Its ease-of-use is extremely important to me.

The Sourceforge project statistics for the 8-week period April 19 through June 17, 2008, shows about 150 web visits each day, but only a few downloads (typically 1 or 2, up to a maximum of 10 on the busiest day). There were no bug reports posted over that time period, and no posts to any of the project mailing lists. I do occasionally receive e-mails directly from users reporting bugs or asking for help with a problem, but I have been unsuccessful in building a user community around the Sourceforge site. PyPedal is clearly a niche product, so there aren't many prospective community members in the first place, but I greatly underestimated the difficulty of getting users to provide feedback. I suspect that there are many users were unable to install early versions of PyPedal and gave up on it without asking for help, which is very unfortunate. A single bug report is far more valuable to the user experience than a hundred testimonials as to the brilliance of PyPedal. I am fortunate that I have had a few users provide lots of feedback, resulting in dramatic improvements in usability.

Plans for the 2.1 series are currently focused on performance improvements, internal overhauls to the graphics routines, and greater use of unit testing. There are also a number of subroutines which interact poorly with the user, and much of that behavior can be fixed without changing the API. Features will be added as needed for my work or requested by users, subject always to time constraints.

While the development of a user community has proven to be much more challenging than anticipated, PyPedal has benefited considerably from the feedback of several users. Any deficiencies in design and implementation, and I'm sure there are many, are the author's alone. Continued feedback and feature requests will hopefully help PyPedal mature into a tool that is useful to a wide community of researchers, population managers, and animal breeders.

Acknowledgments

Work on PyPedal was partially supported by a grant from The Seeing Eye Inc., Morristown, NJ. The author is particularly grateful to a number of people, including E. Hagen, B. Heins, T. von Hassel, and M. Kelly for their feedback, bug reports, and sample datasets.

Mention of trade names or commercial products in this article is solely for the purpose of providing specific information and does not imply recommendation or endorsement by the U.S. Department of Agriculture.

JOHN B. COLE is a Research Geneticist (Animal) in the Animal Improvement Programs Laboratory, a part of the Agricultural Research Service, which is the United States Department of Agriculture's in-house research arm. He's been developing with Python for 10 years in research and production environments, including as a graduate student at the University of Minnesota and Louisiana State University and as the Data Manager for the Southern Regional Climate Center. When he's not doing research or adding new features to PyPedal he geocaches with his wife and two sons and washes and waxes Bowie Volunteer Fire Department apparatus.