PyPedal:
An Open Source project

Release 2.0.0a1

John B. Cole

April 09, 2004

USDA Animal Improvement Programs Laboratory, Bldg 005 Room 306 BARC-West, 10300
Baltimore Avenue, Beltsville, MD 20705-2350

Legal Notice

Disclaimer

CONTENTS

I PyPedal 1

1 Introduction 5
- 1.1 Implemented Features . . . 5
- 1.2 Planned Features . . . 6
- 1.3 Where to get information and code . . . 6
- 1.4 Acknowledgments . . . 7

2 Installing PyPedal 9
- 2.1 Testing the Python installation . . . 9
- 2.2 Testing the Numarray Python Extension Installation . . . 9
- 2.3 Installing PyPedal . . . 10
- 2.4 Testing the PyPedal Python Extension Installation . . . 11
- 2.5 At the SourceForge... . . . 12

3 High-Level Overview 13
- 3.1 Numarray Objects . . . 13
- 3.2 Universal Functions . . . 14
- 3.3 Convenience Functions . . . 14
- 3.4 Differences between numarray and Numeric. . . . 15

4 Tutorial 17
- 4.1 A Few Important Concepts . . . 17
- 4.2 A Gentle Introduction to PyPedal . . . 17

5 API 19
- 5.1 Some Background . . . 19
- 5.2 pyp_classes . . . 19
- 5.3 pyp_utils . . . 19
- 5.4 pyp_nrm . . . 19
- 5.5 pyp_metrics . . . 19
- 5.6 pyp_io . . . 19

6 Glossary 21

Index 23

Part I

PyPedal

PyPedal (“PyPedal”) provides pedigree analysis tools for Python. This part contains all you need to know about “PyPedal” pedigrees and the functions that operate upon them.

CHAPTER

ONE

Introduction

This chapter introduces the PyPedal extensionto Python and outlines the rest of the document.

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. Currently, PyPedal only makes use of information on pedigree structure and does not accomodate genotype data.

Routines are also provided for the decomposition of A and the direct formation of A^1 with and without accountinf for inbreeding. These are of academic interest rather than practical interest, but if a simple script is needed for the inversion of a reasonably-sized pedigree PyPedal is up to the task.

PyPedal is a Python language module that may be called by other Python programs or used interactively from the Python interpreter. The Numerical Python module (http://www.pfdubois.com/numpy/) module is required by PyPedal and must be installed by the end-user. For the stable release of PyPedal 2.0.0, the dependency on Numeric is going to be replaced with a dependency on Numarray. Numarray may be found at http://www.stsci.edu/resources/software_hardware/numarray.

This document is the "official" documentation for pypedal. It is both a tutorial and the most authoritative source of information about pypedal with the exception of the source code. The tutorial material will walk you through a set of manipulations of a simple pedigree. All users of PyPedal are encouraged to follow the tutorial with a working PyPedal installation, working the examples. The best way to learn is by doing — the aim of this tutorial is to guide you along this "doing."

This manual contains:

Installing PyPedal Chapter 2 provides information on testing Python and installing PyPedal.

PyPedal Tutorial Chapter 4 provides information on testing Python and installing PyPedal.

High-Level Overview Chapter 3 gives a high-level overview of the components of the PyPedal system as a whole.

Tutorial Chapter 4 gives provides a walk-through of PyPedal to get you up to speed on the basics.

Glossary Appendix 6 gives a glossary of terms.

1.1 Implemented Features

PyPedal is currently capable of doing the following things:

- 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 using 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.

1.2 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;

1.3 Where to get information and code

PyPedal and its documentation are available at the author's website (sourceforge.net). The Numarray web site is: http://numpy.sourceforge.net/. The Python web site is http://www.python.org/.

1.4 Acknowledgments

PyPedal was initially written to support the author's dissertation research while at Louisiana State University, Baton Rouge (http://www.lsu.edu/). It sat farrow for some time and has recently come under active development again. This is due in part to a request from colleagues at the University of Minnesota that led to the inclusion of new functionality in PyPedal. The author wishes to thank Dr. Paul VanRaden for very helpful suggestions for improving the ability of PyPedal to handle certain computations in very large pedigrees.

Some of the text in this manual is taken verbatim from the Numarray manual. Thanks, guys!

CHAPTER

TWO

Installing PyPedal

This chapter explains how to install and test PyPedal from either the source distribution or from the binary distribution.

Before we can begin the tutorial, we need to make sure that you can install and test Python, the Numeric or Numarray extension, and the PyPedal extension.

2.1 Testing the Python installation

The first step is to install Python if you haven't already. Python is available from the Python project page at http://sourceforge.net/projects/python/. Click on the link corresponding to your platform, and follow the instructions described there. PyPedal requires version 2.3 as a minimum. When installed, starting Python by typing python at the shell or double-clicking on the Python interpreter should give a prompt such as:

```
Python 2.3.3 (#2, Feb 17 2004, 11:45:40)
[GCC 3.3.2 (Mandrake Linux 10.0 3.3.2-6mdk)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
```

If you have problems getting Python to work, consider contacting your local support person or e-mailing python-help@python.org for help. If neither solution works, consider posting on the comp.lang.python newsgroup (details on the newsgroup/mailing list are available at http://www.python.org/psa/MailingLists.html#clp).

2.2 Testing the Numarray Python Extension Installation

The standard Python distribution does not come, as of this writing, with the numarray Python extensions installed, but your system administrator may have installed them already. To find out if your Python interpreter has numarray installed, type '`import numarray`' at the Python prompt. You'll see one of two behaviors (throughout this document user input and python interpreter output will be emphasized as shown in the block below):

```
>>> import numarray
Traceback (innermost last):
File "<stdin>", line 1, in ?
ImportError: No module named numarray
```

indicating that you don't have numarray installed, or:

```
>>> import numarray
>>> numarray.__version__
'0.9'
```

indicating that numarray is installed. If it is installed, you can skip the next section and go ahead to section 2.3. If you don't, you have to get and install the numarray extensions as described on the Numarray website at http://www.stsci.edu/resources/software_hardware/numarray.

2.3 Installing PyPedal

In order to get PyPedal, visit the official website at http://sourceforge.net/projects/pypedal. Click on the "PyPedal" release and you will be presented with a list of the available files. The files whose names end in ".tar.gz" are source code releases. The other files are binaries for a given platform (if any are available).

It is possible to get the latest sources directly from our CVS repository using the facilities described at SourceForge. Note that while every effort is made to ensure that the repository is always "good", direct use of the repository is subject to more errors than using a standard release.

2.3.1 Installing on Unix, Linux, and Mac OSX

The source distribution should be uncompressed and unpacked as follows (for example):

```
gunzip pypedal-2.0.0a12.tar.gz
tar xf pypedal-2.0.0a12.tar.gz
```

Follow the instructions in the top-level directory for compilation and installation. Note that there are options you must consider before beginning. Installation is usually as simple as:

```
python setup.py install
```

or:

```
python setupall.py install
```

There are currently no extra packages for PyPedal.

Important Tip Just like all Python modules and packages, the PyPedal module can be invoked using either the '`import PyPedal`' form, or the '`from PyPedal import ...`' form. Because most of the functions we'll talk about are in the numarray module, in this document, all of the code samples will assume that they have been preceded by a statement:

```
>>> from numarray PyPedal *
```

2.3.2 Installing on Windows

To install numarray, you need to be in an account with Administrator privileges. As a general rule, always remove (or hide) any old version of PyPedal before installing the next version.

Please note that we have **NOT** tested PyPedal on any Win-32 platforms! However, PyPedal should install and run properly on Win-32 as long as the dependencies mentioned above are satisfied.

Installation from source

1. Unpack the distribution: (NOTE: You may have to download an ”unzipping” utility)

```
C:\> unzip PyPedal.zip
C:\> cd PyPedal
```

2. Build it using the distutils defaults:

```
C:\pyPedal> python setup.py install
```

   This installs PyPedal in `C:\pythonXX` where XX is the version number of your python installation, e.g. 20, 21, etc.

Installation from self-installing executable

1. Click on the executable’s icon to run the installer.
2. Click ”next” several times. I have not experimented with customizing the installation directory and don’t recommend changing any of the installation defaults. If you do and have problems, let us know.
3. Assuming everything else goes smoothly, click ”finish”.

Installation on Cygwin

No information on installing PyPedal on Cygwin is available. If you manage to get it working, let us know.

2.4 Testing the PyPedal Python Extension Installation

To find out if you have correctly installed PyPedal, type ‘`import PyPedal`’ at the Python prompt. You’ll see one of two behaviors (throughout this document user input and Python interpreter output will be emphasized as shown in the block below):

```
>>> import PyPedal
Traceback (innermost last):
File "<stdin>", line 1, in ?
ImportError: No module named PyPedal
```

indicating that you don’t have PyPedal installed, or:

```
>>> import PyPedal
>>> PyPedal.__version__
'2.0.0a1'
```

indicating that PyPedal is installed.

2.5 At the SourceForge...

The SourceForge project page for numarray is at http://sourceforge.net/projects/pyedal. On this project page you will find links to:

The PyPedal Discussion List You can subscribe to a discussion list about PyPedal using the project page at SourceForge. The list is a good place to ask questions and get help. Send mail to pyedal-discussion@lists.sourceforge.net. There is also a pypedal-discussion group that you may join.

The Web Site Click on "home page" to get to the PyPedal Home Page, which has links to documentation and other resources.

Bugs and Patches Bug tracking and patch-management facilities is provided on the SourceForge project page.

CVS Repository You can get the latest and greatest (albeit less tested and trustworthy) version of PyPedal directly from our CVS repository.

FTP Site The FTP Site contains this documentation in several formats, plus maybe some other goodies we have lying around.

CHAPTER

THREE

High-Level Overview

In this chapter, a high-level overview of PyPedal is provided, giving the reader the definitions of the key components of the system. This section defines the concepts used by the remaining sections.

3.1 Numarray Objects

The array objects are generally homogeneous collections of potentially large numbers of numbers. All numbers in a numarray are the same kind (i.e. number representation, such as double-precision floating point). Array objects must be full (no empty cells are allowed), and their size is immutable. The specific numbers within them can change throughout the life of the array, however. There is a ”mask array” package (”MA”) for Numeric, but has not yet been implemented for numarray.

Mathematical operations on arrays return new arrays containing the results of these operations performed element-wise on the arguments of the operation.

The size of an array is the total number of elements therein (it can be 0 or more). It does not change throughout the life of the array, unless the array is explicitly resized using the resize function.

The shape of an array is the number of dimensions of the array and its extent in each of these dimensions (it can be 0, 1 or more). It can change throughout the life of the array. In Python terms, the shape of an array is a tuple of integers, one integer for each dimension that represents the extent in that dimension. The rank of an array is the number of dimensions along which it is defined. It can change throughout the life of the array. Thus, the rank is the length of the shape (except for rank 0). **Note:** This is not the same meaning of rank as in linear algebra.

Use more familiar mathematicial examples: A vector is a rank-1 array (it has only one dimension along which it can be indexed). A matrix as used in linear algebra is a rank-2 array (it has two dimensions along which it can be indexed). It is possible to create a rank-0 array which is just a scalar of one single value — it has no dimension along which it can be indexed.

The type of an array is a description of the kind of element it contains. It determines the itemsize of the array. In contrast to Numeric, an array type in numarray is an instance of a NumericType class, rather than a single character code. However, it has been implemented in such a way that one may use aliases, such as ‘`u1`’, ‘`i1`’, ‘`i2`’, ‘`i4`’, ‘`f4`’, ‘`f8`’, etc., as well as the original character codes, to set array types. The itemsize of an array is the number of 8-bit bytes used to store a single element in the array. The total memory used by an array tends to be its size times its itemsize, when the size is large (there is a fixed overhead per array, as well as a fixed overhead per dimension).

Here is an example of Python code using the array objects:

```
>>> vector1 = array([1,2,3,4,5])
>>> print vector1
[1 2 3 4 5]
>>> matrix1 = array([[0,1],[1,3]])
>>> print matrix1
[[0 1]
 [1 3]]
>>> print vector1.shape, matrix1.shape
(5,) (2,2)
>>> print vector1 + vector1
[ 2  4  6  8  10]
>>> print matrix1 * matrix1
[[0 1]                                  # note that this is not the matrix
 [1 9]]                                 # multiplication of linear algebra
```

If this example complains of an unknown name ”array”, you forgot to begin your session with

```
>>> from numarray import *
```

See section **??**.

3.2 Universal Functions

Universal functions (ufuncs) are functions which operate on arrays and other sequences. Most ufuncs perform mathematical operations on their arguments, also elementwise.

Here is an example of Python code using the ufunc objects:

```
>>> print sin([pi/2., pi/4., pi/6.])
[ 1. 0.70710678 0.5       ]
>>> print greater([1,2,4,5], [5,4,3,2])
[0 0 1 1]
>>> print add([1,2,4,5], [5,4,3,2])
[6 6 7 7]
>>> print add.reduce([1,2,4,5])
12                                      # 1 + 2 + 4 + 5
```

Ufuncs are covered in detail in ”Ufuncs” on page **??**.

3.3 Convenience Functions

The numarray module provides, in addition to the functions which are needed to create the objects above, a set of powerful functions to manipulate arrays, select subsets of arrays based on the contents of other arrays, and other array-processing operations.

```
>>> data = arange(10)                  # homolog of builtin range()
>>> print data
[0 1 2 3 4 5 6 7 8 9]
>>> print where(greater(data, 5), -1, data)
[ 0  1  2  3  4  5 -1 -1 -1 -1]         # selection facility
>>> data = resize(array([0,1]), (9, 9)) # or just: data=resize([0,1], (9,9))
>>> print data
[[0 1 0 1 0 1 0 1 0]
 [1 0 1 0 1 0 1 0 1]
 [0 1 0 1 0 1 0 1 0]
 [1 0 1 0 1 0 1 0 1]
 [0 1 0 1 0 1 0 1 0]
 [1 0 1 0 1 0 1 0 1]
 [0 1 0 1 0 1 0 1 0]
 [1 0 1 0 1 0 1 0 1]
 [0 1 0 1 0 1 0 1 0]]
```

All of the functions which operate on numarray arrays are described in chapter **??**. See page **??** for more information about `where` and page **??** for information on `resize`.

3.4 Differences between numarray and Numeric.

This new module numarray was developed for a number of reasons. To summarize, we regularly deal with large datasets and numarray gives us the capabilities that we feel are necessary for working with such datasets. In particular:

1. Avoid promotion of array types in expressions involving Python scalars (e.g., `2.*<Float32 array>` should not result in a `Float64` array).
2. Ability to use memory mapped files.
3. Ability to access fields in arrays of records as numeric arrays without copying the data to a new array.
4. Ability to reference byteswapped data or non-aligned data (as might be found in record arrays) without producing new temporary arrays.
5. Reuse temporary arrays in expressions when possible.
6. Provide more convenient use of index arrays (put and take).

we decided to implement a new module since many of the existing Numeric developers agree that the existing Numeric implementation is not suitable for massive changes and enhancements.

This version has nearly the full functionality of the basic Numeric (only masked array is missing. *Numarray is not fully compatible with Numeric*. (But it is very similar in most respects).

The incompatibilities are listed below.

1. Coercion rules are different. Expressions involving scalars may not produce the same type of arrays.
2. Types are represented by Type Objects rather than character codes (though the old character codes may still be used as arguments to the functions).
3. For versions of Python prior to 2.2, arrays have no public attributes. Accessor functions must be used instead (e.g., to get shape for array x, one must use x.getshape() instead of x.shape). When using Python 2.2 or later, however, the attributes of Numarray are in fact available.

A further comment on type is appropriate here. In numarray, types are represented by type objects and not character codes. As with Numeric there is a module variable Float32, but now it represents an instance of a FloatingType class. For example, if x is a Float32 array, x.type() will return a FloatingType instance associated with 32-bit floats (instead of using x.typecode() as is done in Numeric). The following will still work in numarray, to be backward compatible:

```
>>> if x.typecode() == 'f':
```

or use:

```
>>> if x.type() == Float32:
```

(All examples presume "`from numarray import *`" has been used instead of "`import numarray`", see section **??**.) The advantage of the new scheme is that other kinds of tests become simpler. The type classes are hierarchical so one can easily test to see if the array is an integer array. For example:

```
>>> if isinstance(x.type(), IntegralType):
```

or:

```
>>> if isinstance(x.type(), UnsignedIntegralType):
```

CHAPTER

FOUR

Tutorial

This chapter provides a tutorial for PyPedal. The sample pedigree files may be found in the directory in the distribution. [1]

We are going to start the actual tutorial in this chapter. First, however, we will describe some key concepts that will help you work successfully with PyPedal. You can find a more detailed explanation of PyPedal components in chapter **??**.

4.1 A Few Important Concepts

4.2 A Gentle Introduction to PyPedal

[1] Please let me know of any additions to this tutorial that you feel would be helpful.

CHAPTER

FIVE

API

This chapter provides an overview of the PyPedal Application Programming Interface (API). More simply, it is a reference to the various classes, methods, and procedures that make up the PyPedal module.

5.1 Some Background

5.2 pyp_classes

5.3 pyp_utils

5.4 pyp_nrm

5.5 pyp_metrics

5.6 pyp_io

CHAPTER

SIX

Glossary

This chapter provides a glossary of terms.[1]

pedigree A PyPedal pedigree consists of a Python list containing instances of PyPedal Animal objects.

[1] Please let me know of any additions to this list which you feel would be helpful.

INDEX

P

pypedal (extension module), **3**