DOCUMENTATION FOR THE CRITTRZ POPULATION SIMULATION KIT, VERSION 0.9.5

Revised January 10, 2011

From documentation for previous versions of CRITTRZ

By Tom Olivier

Green Creek Paradigms, LLC

Schuyler, VA 22969 USA

tom.olivier@alumni.duke.edu

Project Home Page

www.greencreekparadigms.com/CRITTRZ.htm

Table of Contents

Licenses

CRITTRZ is released under an open source license commonly known as an 'MIT license'. Here it is.

Module rv, in the CRITTRZ\libs directory, was written by Ivan Frohne (1998). Ivan Frohne holds the copyright to module rv and has released rv under an open source license. Copyright notice and licensing details for rv will be found both in the module source and in the documentation file in the CRITTRZ\docs directory.

Introduction

CRITTRZ is a population simulation construction kit. It is intended for modeling animal populations containing a few dozen to several thousand individuals. It supports modeling populations containing subpopulations with little internal structure as well as subpopulations with complex, dynamic structures seen in some mammalian social groups. CRITTRZ permits examination of interactions of demographic, genetic, social and infectious disease processes. The current release includes modules for construction of models of some Old World monkey populations. The author hopes the CRITTRZ kit will prove extensible to modeling populations of other taxa. The kit includes an interface to a geographic information system (GIS). Through this interface, spatial structures and landscape states of areas occupied by simulated populations can be incorporated into model processes. A simulation may contain multiple populations, representing different species.

The home page for the CRITTRZ population simulation kit project is www.greencreekparadigms.com/CRITTRZ.htm. This page provides links to documentation, reports on applications of the modeling kit and links for downloading current CRITTRZ distributions. You can download some previous CRITTRZ distribution file sets at http://sourceforge.net/projects/crittrz.

Version 0.7.0, the initial version, was released in June 2004. It contained core modules, some sample applications, inputs and brief documentation. Version 0.7.1 was released in August 2004. It added tools and some maintenance changes. Version 0.7.2, released in February 2005, added density-dependent modeling of survival and birth processes and a variety of other refinements. Version 0.7.3 was released in August 2005. It added a model of a monkey population with density-dependent birth and survival processes, a module for tracing descendants of female members of a simulation founding population and some other changes. Version 0.7.4 added a simple, general base class for infectious disease modeling and separate distribution files for Windows computers without a GIS, Windows computers with the Idrisi GIS and Linux computers without a GIS. Version, 0.7.5, added a raster module that reads and writes Idrisi raster files, a home range modeling module, a new population framework that integrates aspects of density-dependent survival and reproduction and a GIS-based monkey density-dependent population application. The high-level organization of the agesex_dd_idrisi application was revised in a form consistent with the new Idrisi-based monkey simulation. Additions to version, 0.8.0, released in July 2007, include module infection_idrisi, which permits modeling of infectious disease transmission where infection likliehoods are affected by numbers of infected individuals sharing landscape areas, a new application with two monkey species that live in the same area but use space differently and transmit an infectious disease across species boundaries and some maintenance changes. Version 0.8.5, released in August, 2008, added the ability to turn log outputs off or on from gin file entries. It also added the ability to generate gene frequency surfaces in a GIS-enabled monkey population model. Version 0.9.0, includes a new utility, logsum.py, that summarizes log structural and event information in tym files. The summaries are in a comma-delimited format suitable for input to spreadsheets and statistical packages. Version 0.9.5, the current version, adds a new simulation harness module, harness_breaks.py, that allows running of subsets of simulation series. The current version also includes a new module for modeling mitochondrial DNA transmission, mtDNA.py. Module migration_idrisi now includes a method that outputs distances between groups in a population. Logging of migration in cercomm family models has been modified so that movements of newly adult males out of their natal groups and movements of already adult males between groups is distinguished.

Version 0.9.5 is a second beta release. Version 0.9.0 is the first beta release. All releases prior to 0.9.0 are considered to be of alpha test development status.

CRITTRZ is open-source software; anyone receiving a copy is free to redistribute it, extend it and modify it, as explained in the license. The author intends that the open-source release will make CRITTRZ readily available to scientists anywhere who might have use for it.

CRITTRZ is intended as a tool for biologists engaged in basic research and conservation.

The kit is designed to allow biologists who are occasional programmers to build simulation applications from preexisting modules. The architecture of the system is highly extensible, though construction of entirely new modules generally will require sophisticated software development skills.

This modeling kit is an outgrowth of simulations of population genetic processes in social groups of savanna baboons and rhesus monkeys built by the author in the 1970's and 1980's. One could see then that computer simulations allowed portrayal of complex population processes observed in real populations, processes that were too complex for ready mathematical treatment. At the same time, construction of simulations of populations exhibiting different kinds of complexity raised new issues regarding general means of representing population structures and events that change them. Olivier (1984, 1985, 2003a, 2003b, 2006) provide background on the biological modeling issues that led to the construction of CRITTRZ and details on organization and applications of earlier development versions. Olivier (2004) reports on an application of CRITTRZ in a study of interactions between behavioral, genetic and demographic processes in simulated populations of Old World monkeys. Olivier (2005) examines matriline numbers, sizes and genetics in simulated monkey populations. Olivier (2007) explained the GIS interface in CRITTRZ and presented examples of its use in applications. Olivier(2008) presented models of population genetic and disease transmission processes in spatially structured Old World monkey populations. Olivier(2010) examined genetic differentiation levels Old World monkey populations structured by both space and social groupings. In this study, different simulation series employed different group fusion and fission size thresholds, resulting in different average group sizes in the different run series.

Organization and Implementation

The modeling kit is written in Python, an open source, object-oriented language. (see http://www.python.org/). Object-oriented languages allow definition of classes of objects that combine data attributes and behaviors. Many instances of a particular object class can exist simultaneously in a program. Objects of one class can be embedded as data objects in another class. New classes of objects can be derived from already defined classes. These properties of objects make them convenient for many tasks in modeling population processes. CRITTRZ uses object-oriented methods extensively.

Python is a dynamically typed language. The types of objects referenced by names can vary during program execution. Also, the types of objects stored in containers such as lists and hash tables (dictionaries in Python) can be varied quite freely. This means, for example, that numbers and kinds of data fields associated with individuals in simulations can be varied readily to suit the purposes of different applications.

Python also emphasizes dynamically sized data containers and automatic release of memory of data structures no longer in use. As a result, there are virtually no fixed limits on numbers of individuals, subpopulations and so on that might be created in a CRITTRZ simulation.

An application written with CRITTRZ usually contains a simulation object with one or more population objects. In turn, each population contains one or more subpopulation objects. Subpopulations may be social groups with members belonging to different elements within the subpopulation. For example, subpopulations in the sample monkey application are social groups divided internally into group elements, one containing immigrant adult males and a natal segment containing adult females and immature offspring of both sexes.

In an application, a simulation steps populations through a series of discrete time periods, with births, deaths and other biological events potentially taking place in each time period. Individual life histories are simulated in detail.

Unique identifiers are given to subpopulations within each population, group elements of subpopulations and individuals. Each identifier consists of a text string that identifies population component type and an integer number. Internal organization of subpopulations is represented in keyed data structures. These may be represented in syntactically organized character strings, Python dictionaries or indented text blocks. See Olivier (1985) and Olivier (2003a, especially Appendix A) for more information on keyed structures, examples of their use in representing subpopulations with simple and complex internal structures and the use of template strings to asses structural correctness.

CRITTRZ architecture isolates many features of population structures and processes. Framework objects store structural information on populations and subpopulations. Separate event modeling objects simulate births, deaths, migrations and other biological events (group fissions, fusions; diploid inheritance; transmission of diseases). A population model is created by binding a population framework with objects for modeling biological events of interest. By design, these bindings occur at a high level in the source code, simplifying code changes required to build models with varying biological assumptions.

Object class derivations are used to extend and specialize modeling kit components for particular applications. For example, a basic population framework that makes no assumptions about age or sex distinctions of population members is specialized in a derived framework to handle populations structured by age and male and female sexes. Additional derivations are used to specialize this age and sex aware framework, and some event modeling classes, to customize them for use in the sample Old World monkey applications.

In basic CRITTRZ models, no spatial structuring of relationships among subpopulations is assumed. However, geographic space can add structure to many population processes. In a simple spatial model, subpopulations might be arranged on a homogeneous, Euclidean plane, with isolation by distance influencing processes such as the migration of individuals. More realistically, landscapes may vary greatly in structure, with costs of movement, habitat suitability, disease risks and other factors varying irregularly or in complex patterns with location.

To facilitate sufficient realism in modeling spatial and landscape effects, one of the CRITTRZ for Windows distribution files includes a run-time interface to the Idrisi GIS (Clark Labs, 2009). At present, this interface provides some foundation functionality, a module that models migration between subpopulations occupying subareas of a landscape where costs of movement vary over the landscape, as well as survival and reproduction modules where these events are affected by resources available in areas occupied by subpopulations.

Idrisi was chosen as a foundation for this initial representation of spatial and landscape effects because this software emphasizes raster representation of data, which seem well suited to many conservation projects, the programming interface is relatively simple and quite powerful analytical tools are included. Modules idripop, geo_idrisi, migration_idrisi, survival_dd_idrisi, reproduction_dd_idrisi, agesex_popframe_idrisi, applications agesex_base_dd_idrisi_app and cercomm_dd_idrisi_app and harness agesex_base_dd_idrisi_harness require presence of an Idrisi GIS system. Modules raster_idrisi, infection_idrisi and home_range_idrisi are written in Python and directly access and manipulate Idrisi GIS files, without use of a separate GIS program.

CRITTRZ is designed for creation of models that run as unattended batch applications. A simulation object is created to oversee each model run. The simulation object manages most file input and output. Text file inputs and outputs are used primarily, for portability. A simulation object obtains the run name from the command line. At the start of model execution, the simulation opens and manages reading of an initialization file (run name + ". gin"). Simulations generate various standard output files. These standard files are written to the same directory as the gin file and have the same name but different extensions (extensions "fin", "tym", "ro" and "err").

The simulation opens a temporal log file (run name + ".tym") that remains open until simulation termination. Various CRITTRZ objects, including logs, are able to write entries to this tym file, with entries potentially including details of biological events, summaries of population states and test data. At normal termination, the simulation opens and manages writing of a file with information on the terminal state of the simulation (name + ".fin"). The terminal state files are useful for biological analyses and correctness checking.

The formats of the gin and fin files for a particular application are identical.   These files are organized into labeled, hierarchically structured text blocks. Each block consists of or one or more lines. The simulation block is outermost. At the beginning, it contains simulation related information (for example, simulation name, time periods, random number initialization).   The simulation block then contains information blocks for one or more constituent populations. Each population block normally includes parameter values for methods it employs, a key structure template used for periodic checks of subpopulation structural correctness, a list of subpopulations with keyindent representations of their internal organizations and possibly other data, a list of attributes (or data fields) for individuals currently present in the population and kinship records for population members. A fin file can be converted readily to a gin file, should the user wish to extend a simulation run.

A single line of text in one of the CRITTRZ I/O files may be too long for convenient inspection. Long, logically single, lines of text may be broken into several physical lines in I/O files by the use of a line continuation string, "[\]". Module texio, which handles most file input and output, automatically handles these line continuations.

Should a simulation abort on detection of an error condition, the simulation object writes an error file (run name + ".err") with information on the time and location of the error and some messages regarding other circumstances.

As a programming guideline, CRITTRZ classes should provide methods for any file input and output that they require.

At times, CRITTRZ classes may require I/O of extensive, possibly binary, data that are not sensibly placed a standard CRITTRZ file. One approach to such cases is input and output of these data via separate files, with the name of the separate file indicated in gin and fin files. For example, at the end of each run, the rv random number generator library module saves its internal state by writing a table of values to a file with extension 'ro' (run name + "ro"). Rather than write this table into the fin file, the fin file simply contains a tagged line with the name of the ro file. Should one wish to extend a simulation run, by converting its fin file to a gin file, on restart, CRITTRZ will automatically locate and read the ro file and use its contents to initialize the random number generator for the extension of the run.

Module pathserver provides each simulation with a mechanism for locating and naming input and output files other than previously discussed standard files. These other files are referred to as auxiliary files in the remainder of this document. Several considerations led to creation of patherserver: 1) increasing use of application-specific GIS files to store information on model landscape states 2) need to dynamically create data files related to subpopulations that appear during simulations 3) wish to specify input and output files in standard locations by name, without use of full pathnames.

By default, pathserver looks for auxiliary files in the same directory as standard files. An optional directory field in gin files allows the user to specify that auxiliary files will be found in a subdirectory ("auxstuff") of the standard file directory or be stored in a subdirectory with a name derived by pathserver from the simulation name.   Input file names may be specified as entries in gin files with only file names and extensions. When read, pathserver appends a file name to an auxiliary file path based on the simulation directory option. To support creation of new files during a simulation, pathserver methods can generate new file names that include substrings derived from combinations of simulation, population and subpopulation names.

Discussion of CRITTRZ Packages, Modules and Classes

CRITTRZ is structured as a Python package, with subpackages built of Python modules. The packages are organized hierarchically and follow the structure of the CRITTRZ directory. The use of packages permits firmer control of module importations and facilitates coexistence of different versions of CRITTRZ on a single computer.

A brief explanation of some interrelationships between modules follows. Although tedious, this discussion should be useful to anyone who wishes to examine kit code.

Module CRITTRZ_labels, found in the CRITTRZ package directory, assigns values to variables that are used as constants widely throughout the CRITTRZ system.

Next, I consider modules found in the mods subpackage. Module simulation defines a simulation class that manages much file input and output, manages the progression of time periods and oversees error handling. A pathserver object is embedded in each simulation object. By convention, population objects are created as attributes of simulation instances.

Module base_popframe defines base classes for population and subpopulation frameworks. Embedded in base_popframe class instances are objects that manage the assignment of unique identification numbers to simulated individuals, groups and other subpopulation elements (module idnserver), maintain a database on attributes of individuals present (module indfieldserver), maintain a database on parent-offspring links (module kinshipserver) and test structural correctness of subpopulations (modules template, templateserver and keystring). Instances of class base_subpopframe each contain an instance of class keygraph (module keygraph). Keygraphs store subpopulation key structural information using Python dictionaries and provide many tools for deterministic manipulation of the structures. Module keyindent provides a means of convenient file input and output of key structures, using the indented text block format.

Module agesex defines classes agesex_popframe and agesex_subpopframe. These are derived from counterparts in module base_popframe and add some handling for populations with age structuring and male and female sexes.

Modules geo, geoidrisi and idripop support modeling space and landscape features by providing an interface to the Idrisi GIS. Modules raster_idrisi and home_range_idrisi also support use of Idrisi GIS raster files to represent states of landscapes occupied by simulated populations, but do not require presence of Idrisi software.

Turning to biological events, module survival models survival, with probabilities fixed by age and sex. Module reproduction models reproduction, with reproduction rates fixed by female age. Poisson and binomial birth distribution models are provided. Modules survival_dd and reproduction_dd provide density dependent versions of these events. Modules survival_dd_idrisi and reproduction_dd_idrisi provide density dependent modeling of these events, with resources that influence survival and reproduction obtained from GIS rasters of subpopulation locations and resource distributions.

Module migration models a random migration process, with no spatial structuring. Module migration_idrisi uses a GIS raster representation of a cost-of-movement surface to model migration with distance effects. Diploid inheritance (module diploid) also is modeled. Module infection, provides a basic representation of infectious disease processes. Module diploidanalyst provides some tools for analyses of gene frequency distributions in subpopulations.

Fissions of subpopulation group elements are modeled by modules keyfission and linfission. The latter divides a subpopulation or group element along unilineal, genealogical lines (for example, along matrilines). Linfission requires that the population framework on which it acts be equipped with a uniline kinship data manager. Class uniline is derived from class kinshipserver and adds the ability to trace unilineal pedigree relationships through males or females. Module uniline in turn employs classes in modules squarematrix and unimat. Module femfounders traces descent of simulated individuals from females present in the population at the start of simulation. Module mtDNA models models mitochondrial DNA transmission. This model makes the simplifying assumption that mtDNA in each individual is of one type.

Some utility modules are used widely by other CRITTRZ components. Module texio provides many functions for input, output and other processing of textual information, particularly lines labeled with tag identifiers. Module log provides machinery for writing labeled entries in text logs. Entries to log files are written in Python dictionary format. Python scripts can easily read and analyze log contents; people can read logs as well, though perhaps not so easily. Biological event modules generally contain embedded log objects that can be turned on to write data for code testing or analyses of simulated biological events. The logsum tool provides simple quantitative summaries of event log entries in each time period in tym files. Module tally provides methods for tallying individual field values in the entire population, subpopulation or elements of subpopulations. Module tally is a useful foundation class for the construction of objects that statistically summarize population states. CRITTRZ uses the "rv" random number generating package by Frohne (1998). Module randy provides a light wrapper to the rv package. .

Modules in the cercomm subpackage directory (cercomm_fission, cercomm_fusion, cercomm_migration, cercomm_diploidanalyst, cercomm_popframe, cercomm_dd_events_idrisi) provide components for models of populations of Old World cercopithecine monkeys with multi-male groups. These cercomm modules are built using extensions and combinations of more basic CRITTRZ modules. Module cercomm_dd_events_idrisi derives GIS-enabled versions of cercomm fission, fusion and migration classes and a home range class that supports home range fissions and fusions needed in cercomm models.

The tools subpackage includes demograteplay, a script that allows exploration of population growth given fixed, age-specific survival and birth rates. The harness_base tool provides a Python class that can serve as a foundation for creation of scripts that oversee conduct of simulation run series. The harness_breaks tool, derived from harness_base, allows a simulation series to be run in segments. Tool seedset uses the built-in Python random number generator, with initialization from the computer hardware state, to create a file with a list of random numbers. Seedset output files are used to seed the random number generator for the different runs in simulation series overseen by scripts built with the harness_base tool. Tool demogratedd deterministically models density dependent population growth. Tool dd_extract, extracts population size data from tym files and exports them to comma-delimited files. Tool logsum summarizes population structure and event log information in tym files and exports summaries to comma-delimited files.

Classes and methods in CRITTRZ modules generally have been written with internal "docstrings". These docstrings provide brief comments on the function of the class or method and are accessible within the Python interpreter. I have tried to use descriptive names for most classes and methods and in many cases the names of classes and methods are about as informative as the current docstrings.

Installation

Three compressed distribution files are available for CRITTRZ Version 0.9.5. CRITTRZ.0.9.5.Win.zip and CRITTRZ.0.9.5.WinIdr.zip are intended for installations on computers running the Windows operating system. CRITTRZ.0.9.5.Lin.tar.gz, is intended for installation on computers running the Linux operating system. Files CRITTRZ.0.9.5.Win.zip and CRITTRZ.0.9.5.Lin.tar.gz contain core modeling files and are nearly identical in content. They differ in that scripts in the Windows file are of the batch type and scripts in the Linux file are bash shells and may vary slightly in name. Also, end of line characters in text files differ in accordance with host operating system conventions. CRITTRZ.0.9.5.WinIdr.zip includes files present in CRITTRZ.Win.0.9.5.zip plus others that provide a run-time interface to the Idrisi GIS or use Idrisi raster files directly. Use of this interface requires installation on the computer of a recent version of the Idrisi GIS. Idrisi is sold by Clark Labs (2009; http://www.clarklabs.org/) and runs only on computers with the Windows operating system. CRITTRZ 0.9.5 has been developed using the initial Taiga version of Idrisi. It has not yet been tested with the more recent Taiga Idrisi release. The WinIdr distribution file also includes additional example programs that make use of the Idrisi interface and related script and input files. The terms ‘Win’, ‘Lin’ and ‘WinIdr’ are used elsewhere in this document to refer to these different distribution files or their resultant installations.

Using Windows Distribution Files

At present, the Windows version of CRITTRZ is intended for use on computers running the Microsoft Windows XP operating system. Running CRITTRZ requires presence of a Python interpreter, preferably version 2.5.4(downloadable at http://www.python.org/). In addition to requiring installation of a recent version of the Idrisi GIS, use of the WinIdr distribution file with the Idrisi interface also requires installation of Mark Hammond’s “Python for Windows Extensions” (Hammond, 2004; downloadable at http://sourceforge.net/projects/pywin32). This package contains an integrated development environment (a combination editor, debugger and console window) that CRITTRZ users may find helpful even if they do not run Idrisi-dependent modules.

CRITTRZ has been designed to allow flexible location of the package on file systems and coexistence of multiple versions on a single computer. CRITTRZ can be installed as a top-level directory on a hard drive (e.g. C:\CRITTRZ) or a subdirectory (e.g. C:\Models\CRITTRZ).

To install CRITTRZ, unzip the distribution file, specifying the directory in which you wish CRITTRZ installed, indicating that directories should be preserved. You must tell the Python interpreter where to look for CRITTRZ. This can be accomplished by creating a user environment variable named PYTHONPATH in the control panel with value set to the directory in which CRITTRZ is installed (do not include ‘CRITTRZ’ at the end of this variable value). If PYTHONPATH already exists, append the location of the CRITTRZ package to the end of it.

Alternately, one can create a “path file” in the directory in which the Python interpreter is installed. Directories specified in a path file are added to the list of directories on the Python interpreter search path. A text file named ‘CRITTRZ.pth’, containing one line with the directory in which CRITTRZ is installed has worked. Note that path file syntax requires directories be indicated with double, rather than single, slashes. If CRITTRZ is installed in directory “Models” on drive “C”, a CRITTRZ path file would contain the line “C:\\Models”. See the Python documentation for additional discussion on making the Python interpreter aware of packages such as CRITTRZ.

Spaces in command line application argument pathnames (as in "My Documents") can be parsed incorrectly, triggering exceptions and execution aborts. In the PythonWin IDE, placing quotes and the beginning and end of each pathname argument cause correct parsing. Example application batch files have been revised to correctly handle installation in paths where directory names contain spaces.

Using the Linux Distribution File

CRITTRZ has been tested on a computer running the SuSE Version 10.1 Linux distribution and bash shell scripting. In its current form, the author expects CRITTRZ to generally be compatible with other Linux distributions. Running CRITTRZ requires presence of a Python interpreter. The Linux version of CRITTRZ has been test run using Python version 2.4.2. Many Linux distributions include Python. Information on downloading a Python interpreter is available at http://www.python.org/. Some download links are to interpreter source files that must be compiled and linked before use. You may find an RPM package built for your Linux distribution the most convenient means of installing Python, but you may have to look around to find one.

CRITTRZ has been designed to allow flexible location of the package on file systems and coexistence of multiple versions on a single computer. You may find it convenient to install a CRITTRZ distribution in a subdirectory of your home directory (for example '~/CRTZ'.

To install CRITTRZ, place the distribution file in the directory in which you wish the distribution installed. Extract the distribution file using the tar command with '-xzf' as options (for example 'tar –xzf CRITTRZ.0.9.0.Lin.tar.gz'). At this point, the distribution files should be installed in expected subdirectories.

You must tell the Python interpreter where to look for CRITTRZ. One way to do this is to create in each session a PYTHONPATH shell variable set to the location in which the CRITTRZ distribution resides (for example with command'PYTHONPATH=~/CRTZ') and then export the variable (with command 'export PYTHONPATH'). Alternately, you can place a command in the '.bashrc' configuration file in your home directory (for example 'export PYTHONPATH=~/CRTZ').

Notes for All Distribution Files

CRITTRZ applications and data sets may reside outside of the CRITTRZ directory tree.

The CRITTRZ distribution files contain a copy of the rv.py open source random number generating module, written by Frohne (1998). This module and its documentation are located in the CRITTRZ\libs directory.

Sample Applications

Nine sample applications with inputs are provided in all distribution files; five more are present in the WinIdr file. Scripts for execution of sample applications are found in package directory CRITTRZ\apps. Before executing any of these scripts, open a command prompt or console window and use the change directory command to make the CRITTRZ\apps directory your current directory. To execute a Windows batch file, simply type its name at the command line. To execute a Linux bash shell file, type 'bash ' and the name of the shell file on the command line. Output from a simulation is generated in the same directory as the gin file for the application or its auxiliary data directory. Remarks in the script files, displayed on the console at execution, note the files that contain distinctive output from each sample simulation.

The agesex_base application models a population with multiple subpopulations and age and sex structuring. Migration event logging is turned on; log output is contained in the application tym file. The agesex_base_dd application models a population with density-dependent survival and reproduction. Resources available to each subpopulation that influence survival and reproduction rates are read from the gin file. Population sizes at each time period are displayed on the console and entered in the tym file. Application agesex_dd_idrisi also models a population with density-dependent survival and reproduction. However, in this model, resources for survival and reproduction are determined from GIS raster files that represent landscape states. The agesex_diploid application adds diploid genetics to the base agesex model. The presence of genotype fields for individuals can be seen in the gin and fin files. Test logging in the diploid genetics modeler is turned on, with detailed testing output present in the application tym file. Application agesex_infection uses the infection module to model infectious disease transmission. A tally object is added to the population to summarize numbers of individuals infected, previously infected and not infected in the population. These tallys are displayed on the console as the simulation proceeds through time periods. Application agesex_mtDNA models mitochondrial DNA in an agesex population. The tym file displays tallies over time of mtDNA types present in the population and records of mtDNA mutations. Application two_pops presents a simulation with two distinct agesex populations. The gin and fin files illustrate some I/O formatting for a multi-population simulation.

The cercomm applications model a population of Old World monkeys divided into multimale social groups. The base cercomm application logs group fission and fusion events in its tym file. Comparison of gin and fin files reflects changes in groups present detailed in the group fission and fusion logs. Application cercomm_diploid adds diploid genetics to the base cercomm model. A tally object is created for the population and used to create subpopulation genotype counts that are written to the application tym file. Application cercomm_base_dd models a monkey population with density dependent birth and survival events. Derived fission and fusion classes in the application code manage allocation of demographically important resources during group fissions and fusions. Application cercomm_dd_idrisi_app models a monkey population with density dependent birth and survival events, using GIS analyses and rasters to assist in modeling population interactions with landscapes. Population size numbers are written to the console and tym files. The auxiliary data directory for the application contains rasters representing subpopulation home ranges, population resources and demand, as well as costs of movement. Home ranges may shrink, expand, shift, fission and fuse, in concert with population events. Application cercomm_dd_infection_idrisi models infectious disease transmission in a spatially-structured cercomm population. Counts of individuals in different infection states are shown on the console and a raster with counts of infected individuals in different locations is generated. Application cercomm_dd_diploid_idrisi models genetic dynamics in a spatially structured population. This model provides Fst statistics to the console during the run and allele frequency surface rasters at simulation termination.

Application_cercomm_two_pop_dd_infection_idrisi_app models disease transmission within and between populations of two socially subdivided monkey species. The model employs Idrisi raster layers to represent landscape states. The two species populations occur within the same area but use space differently. Groups of one species are distributed throughout the landscape. Groups of the second species population occupy a Y-shaped area that could represent a network of riparian areas.

Simply changing the random number generator seed value will produce different simulation results. This value is set in the gin file on the line with the tag "[random_initialization_seed:]" The number of time periods in a simulation run can be altered by changing the time limit value, in the gin file on the line with the tag "[time_limit:]".

The base agesex and cercomm applications both use the survival and reproduction modules, though agesex uses the Poisson birth model and the cercomm models use the binomial birth model. Both applications use the migration module, though the in the cercomm application, migration is used as a foundation for a more complex and specialized cercomm_migration module. The cercomm application adds group fission and fusion processes. Note that in the applications using diploid genetics, the initialization of the main population class includes a parameter reference to a diploid genetic event modeling class. In the main loop of these models, after births have occurred, a call is added to simulate gene transmission from parents to newborns. In the agesex_infection application one sees, similarly, the addition of a disease modeling class to the parameter list for the population initialization method and a calls to disease transmission methods in the main simulation loop, between reproduction and survival modeling. Within the population initialization routines, one sees creation of instances of the event modeling classes and a binding of the population and event modeling instances to each other. The cercomm_dd_idrisi_app makes use of multiple class inheritance for the migration event and population framework

Perusal of the different application sources will reveal many similarities. User-developed applications could be structured similarly.

Three examples of simulation series scripts built with the harness_base tool are provided in directory harns in all distribution files. They provide examples of 1) a basic agesex model 2) an agesex model with density-dependent modeling of survival and reproduction 3) cercomm population simulations that include diploid gene transmission. The WinIdr distribution includes an agesex model using GIS representation of resources influencing density-dependent survival and reproduction, as well as migration affected by distance and costs of movement that vary with location on the landscape.

A script for command line execution of the three harness scripts that do not require a GIS (most_harness_examples.bat for the Win distribution installation; Lin_apps.sh for the Lin distribution) is found in directory harns. Before executing it, open a command prompt or console window and use the change directory command to make the harns directory your current directory. In the WinIdr installation, script agesex_dd_idrisi_harness.bat executes the example that utilizes the Idrisi GIS. Script call_dd_idrisi.bat is an auxiliary script used to invoke the Idrisi harness example and is not designed to be called directly from the command line.

Console output from the agesex harnesses is minimal.   However, the density-dependent models record population sizes at different times in the run tym files. The Idrisi-based density dependent harness model uses a base subdirectory to store raster files required for simulation runs. These base directory files are copied by the harness to an auxiliary file directory, in keeping with the directory option specified in the example base gin file.

The cercomm harness application uses the diploidanalyst module to send FST gene frequency differentialion measures to the console. Mean, minimum and maxium FST values for each simulation and for the series of simulations are provided.

Note that the harness scripts and IO directories lies outside the CRITTRZ package directory structure. This is done by design to allow relatively free location of harness scripts and IO within the user's computer file system.

Development History and Plan

Design of CRITTZ began in August 2002, with coding begun in September 2002, with the initial public release in June 2004. Version 0.9.5 is the tenth release of CRITTRZ.   This version includes the full range of functions planned for Version 1.0. Version 0.9.5 is viewed as a beta test release. I aim to release Version 1.0 during 2011. A major rewrite of the documentation, including a modeling tutorial, is planned for Version 1.0.

Starting in early 2003, CRITTRZ modules have been tested frequently using unit test modules and the PyUnit test harness (Purcell, 2001). Unit tests greatly facilitated refactorings that occurred, particularly during earlier releases.

The remainder of this section offers miscellaneous further thoughts on changes and extensions that may be made in the future.

At present, CRITTRZ represents landscape states through an interface to a GIS system that runs only under Microsoft Windows operating systems.   With the addition of raster_idrisi and home_range_idrisi modules CRITTRZ can read, analyze, modify and write Idrisi raster files without use of the Idrisi GIS. These modules might provide a basis for a tiny GIS within CRITTRZ that would support raster-based landscape modeling in applications without need for calls to a separate GIS program.

At some point, replacement of the rv random number generator module with a generator written for the current version of Python may be needed. CRITTRZ module randy wraps and isolates the random number generator, potentially simplifying the tasks involved in swapping generators.

Creation of a tool to extract simulation kinship link files from CRITTRZ is likely, so that stand-alone kinship analysis programs can analyze kinship relationships in simulated populations.

The calculation of genealogical relationships by methods of module uniline slows rapidly with increases in the number of individuals among which relationships are traced. Unilineal relationships are traced, for example, as a prelude to group fissions in the cercomm models. The matrix methods in module squarematrix that are used by modules unimat and uniline are only slightly optimized. No doubt, far more efficient implementations are possible.

Input consistency checks could be extended and messages on detection of some errors could be made more specific.

Some code refinements are likely in coming releases. If you scan module source code, you will find comments that include '***'. These indicate sections of code that are candidates for revision. You also will find some module method docstrings contain the word 'DEPRECATED' (for example in module keygraph). These methods may disappear in later versions and should not be used in new code.

Comments on potential extensions from readers are welcome.

Changes in this Version

A new simulation series management module, harness_breaks, has bee added. This allows a simulation series to be run in segments or subsets. Use of harness_breaks allows calls to a hard disk defragmenter to be interspersed with simulation runs. Hard drive fragmentation can lead to increases in run times as simulation series proceed.

A module, mtDNA.py, for modeling transmission, mutation and population dynamics of mitochondrial DNA has been added. This model makes the simplifying assumption that mtDNA in each individual is of one type. An application employing this module, agesex_mtDNA_app.py, also has been added.

A method that writes to a file distances between subpopulations, reckoned using the cost of movement surface for the landscape occupied by a model population, has been added to module migration_idrisi.py.

Event logging of migration of individuals cercomm family models has been modified, primarily to distinguish movements of newly adult males from their natal groups and movements of already adult males between groups. The logsum utility has been modified to process the revised cercomm migration logging.

This documentation has been revised.

Acknowledgments

The author thanks his wife, Wren, for encouraging the development of CRITTRZ. I also thank my son, David for prompting me to incorporate unit testing into the development process and for reviewing a draft of this documentation. Thanks also to Jeff Rogers for suggesting I incorporate a mitochondrial DNA modeling module.

References

Clark Labs. 2009. Idrisi Ver. 16.00 Taiga. Worcester

Frohne, I. 1998. rv. Vers. 1.1 No known web download site now. Copy of source and documentation included in CRITTRZ distribution zip file.

Hammond, M. 2004. "Python for Windows Extensions", also referred to as "pywin32". Home page at http://starship.python.net/crew/mhammond/win32. Current download site at http://sourceforge.net/projects/pywin32.

Olivier, T. J. 1984. "Some Issues in the abstraction of genetic structures of monkey societies." J. Social Biol. Struct. 7, 61-73.

Olivier, T. J. 1985. "Use of Keyed Character String Data Structures and Operators in Models of Primate Gropus," J. Theor. Biol. 115, 539-549.

Olivier, T.J. 2003a. "A Population Simulation System Written in Python." Paper presented to the PyCon Conference, Washington DC, 28 March 2003. Link to Microsoft Word file at www.greencreekparadigms.com/CRITTRZ.htm.

Olivier, T.J. 2003b. "Design Goals and Application of a Simulation System for Conservation Biology." Poster presentation at the Society for Conservation Biology Meetings, 29 June – 2 July, 2003, Duluth, MN. Link to PDF file at www.greencreekparadigms.com/CRITTRZ.htm.

Olivier, T.J. 2004. "Interactions between Social, Genetic and Demographic Processes in Simulated Monkey Populations." Poster presentation at the Society for Conservation Biology Meetings, 29 July – 2 August, 2004, New York, NY. Link to PDF file at www.greencreekparadigms.com/CRITTRZ.htm

Olivier, T.J. 2005. "Genetics and Population Dynamics of Matrilines in Simulated Monkey Populations." Oral presentation at the Society for Conservation Biology Meetings, 17 July,2005, Brasilia, Brazil. Link to HTML file at www.greencreekparadigms.com/CRITTRZ.htm.

Olivier, T.J. 2006. "Experiences with Computer Simulations of Primate Populations." Oral presentation at NKS 2006 Wolfram Science Conference, 16 June, 2006, Washington, DC. Link to PDF file: www.wolframscience.com/conference/2006/presentations/materials/olivier.pdf

Olivier, T.J. 2007. "Use of a Geographic Information System to Represent Landscape and Population States During Population Simulations." Oral presentation to the Eighteenth Annual Virginia GIS Conference, Virginia Beach, VA. Link to PDF file at www.greencreekparadigms.com/CRITTRZ.htm.

Olivier, T.J. 2008. "Use of the CRITTRZ Simulation System to Model Genetic and Disease Processes in Populations with Dynamic Spatial and Social Structures." Poster presentation at the Society for Conservation Biology Meetings, 14-18 July, 2008. Chgattanooga, TN. Link to PDF file at www.greencreekparadigms.com/CRITTRZ.htm.

Olivier, T.J. 2010 "Genetic Differentiation in Simulated Monkey Populations with Spatial Substructure and Varying Group Fusion and Fission Size Thresholds." Poster presented at the Twenty-Third Congress of the International Primatological Society, 12-18 September, 2010, Kyoto, Japan. Link to PDF file at www.greencreekparadigms.com/CRITTRZ.htm.

Purcell, S. 2001. PyUnit Testing Framework. Ver. 1.4.1 Home page at http://pyunit.sourceforge.net.

Module: **agesex.py**

class: **agesex_popframe**

[age_classes:]: Number of age classes. Should be an integer >= 1.
[initial_age_class:]: Numeric value of initial age class. May be 0 or 1.

Module: **agesex_popframe_idrisi.py**

class: **agesex_popframe_idrisi_dd**

[individual_optimal_resources:] : Real number representing an optimum level of resources per individual. This value is used in modeling density-dependent processes. Levels of available resources greater than the optimum are treated as equal to the optimal.

[population_resource_raster:]: Name of population resource raster. Should include extension, '.rst'. Should not include directory. Normally, this raster is read at beginning of simulation and must be provided by the user. Each raster cell represents resources available to members of population. Resources of a cell are accessible to members of subgroups that include the cell in their home ranges.

[population_resource_demand_raster:]: Name of population resource demand raster. Should includ extension '.rst'. Should not include directory. Versions of this file are created during simulations. The user normally does not need to supply a raster of this name to start a simulation. Each cell represents the number of individuals occupying that location. Occupation (or demand) is calculated by summing demand factors for each cell over all subpopulations that include the cell in their home ranges. The per cell demand factor for each group is calculated by dividing the number of individuals in the subpopulation by the number of cells in the group home range.
[resources_power:]: Real number with positive value. The ratio of resources available per individual to the optimal level is employed in modeling some density dependent process. This variable repreents the power to which the ratio may be raised.

Module: **base_popframe.py**

class: **base_popframe**

[key_io_mode:]: String that specifies mode of input and output of key structures. Should be 'indent, 'graph' or 'string.' Indented text blocks are used for input of key structures in example applications. The 'graph' option specifies use graphs portrayed in strings with the syntax of Python dictionaries. The 'string' option refers to strings with key string syntax.

[population_name:]: String that specifies name of population. May be used in log entires or as substring in names of files created by CRITTRZ in Idrisi-enabled models. For the latter, short names composed of letters and numerals will be most trouble-free.

[population_name_logging_flag:] : String that sets inclusion of population name in log entries on or off. Value must be ‘on’ or ‘off’. Optional, with default of ‘off’. Inclusion of population name is useful for simulations with multiple populations.

[subpopulation_number:]: Integer with number of subpopulations in population. In gin file, should match number of subpopulations represented by key structures at start of simulation. Value is updated during simulations where subpopulations may appear and disappear, with updated values written to fin files.

Logging on/off flags: This module provides code to optionally read event and test log settings of application modules. The block of log settings begins with tag '[logging_flags_start]' and ends with tag '[logging_flags_end]'. Logs under control of flags in this block are identified in application code. Flag values must be ‘on’ or ‘off’. Examples could be ‘[migration_test_logging_flag:] off’ and ‘[reproduction_logging_flag:] on’. When set from the gin file, the population name logging flag should be set within this block.

Module: **cercomm_dd_events_idrisi.py**

class **cercomm_migration_idrisi**

[distance_weighting:]: Real number representing the weighting given to geographic distance in selection of group to which an emigrant immigrates. Value should range from 0 to 1.

Module: **cercomm_diploidanalyst.py** - No input variables.

Module: **cercomm_fission.py**

class **cercomm_fission**

[cercomm_group_fission_size_threshold:]: Integer with group size that triggers initiation of group fission process.

Module: **cercomm_fusion.py**

class **cercomm_fusion**

[cercomm_group_fusion_size_threshold:]: Integer with upper limit for sizes of groups that are candidates for fusion with another group.

Module: **cercomm_migration.py**

class **cercomm_migration**

[minimum_female_adult_age:]: Integer with minimum age for female to be tallied as adult. Adult sex ratios influence attractiveness of groups to immigrants.

[natal_male_emigration_age:]: Integer with age for male emigration from natal group.

Module: **cercomm_popframe.py**

class **cercomm_population_framework**

[adult_male_element_label:]: String with label for key structure containing adult males.

[group_label:]: String with label for key structures that represent groups.

[natal_element_label:]: String with label for key structure containing natal members of group.

Module: **diploid.py**

class: **diploid**

[diploid_loci:]: Integer with number of loci modeled in population.

Module: **diploidanalyst.py** - No input variables.

Module: **femfounders.py** - No input variables.

Module: **geo.py** - No input variables.

Module: **geoidrisi.py** - No input variables.

Module: **home_range_idrisi.py**

class **home_range_idrisi**

[add_distance_maximum:]: Integer representing maximum distance from home range center for cell that is candidate for addition to home range. Distance is calculated as maximum of a) absolute difference between x coordinate of cell and x coordinate of home range center b) absolute difference between y coordinate of cell and y coordinate of center. Value should be positive.

[add_occupancy_maximum:]: Real number representing maximum occupancy of cell that is candidate for addition to home range.

[adjoining_cell_add_fraction:]: Fraction of cells adjoing home range edge considered for addition to home range during home range update process.

[cell_bound_number:]: Integer representing maximum number of raster cells allowed in a subpopulation home range.

[resource_demand_multiplier:]: Real number multiplied times individual optimal resources value (see agesex_popframe_idrisi). This product affects whether a cell adjoining a home range is added during an update process.

[update_expand_shrink_passes:]: Integer representing number of edge expansion/shrink passes conducted in a home range update.

Module: **idnserver.py** - No input variables

Module: **idripop.py** - No input variables

Module: **indfieldserver.py**

class **indfieldserver**

[individual_fields:]: Line with data fields for an individual alive in the population. Line begins with identification number of individual, followed by a comma, followed by a substring in Python dictionary format with field keys or names and values. Here is an example: "[individual_fields:] 457, {'age': 4, 'sex': 'm'}".

Fields for all live individuals are listed in sequence in a block of lines that starts after the tag "[individual_fields_start]" and ends immediately before the tag "[individual_fields_end]".

During a simulation, individual field records an individual in the population are erased from memory when that individual dies. Note that kinship links are stored in separate data structures. Kinship links are retained after simulated individuals die.

Module: **infection.py**

class **infection**

[base_infected_mortality_probability:]: Real number that represents probability that in an individual who is infected will die in an infection mortality trial. Value should range from 0 through 1.

[base_infected_to_previous_probability:]: Real number that represents probability that in an individual who is infected will overcome the infection and become previously infected in an infection state transition trial. Value should range from 0 through 1.

[base_never_to_infected_probability:]: Real number that represents probability that in an individual who never has been infected will become infected in an infection state transition trial. Value should range from 0 through 1.

[base_previous_to_infected_probability:]: Real number that represents probability that an individual who was previously infected will become infected again.in an infection state transition trial. Value should range from 0 through 1.

Module: **infection_idrisi.py** - No input variables.

Module: **keyfission.py**

class **key_fission**

[key_fission_event_label:]: Text string used to identify in gin/fin files event modeled by key fission instance.

[paired_allocation_fraction:]: Real number representing fraction of items in list allocated through random, pariwise allocations to fission product key elements. Value should range from 0 through 1.

Module: **keygraph.py** - No input variables.

Module: **keyindent.py** - No input variables.

Module: **keystring.py** - No input variables.

Module: **kinshipserver.py**

class **kinshipserver**

[kinship_links:]: Line that identifies father and mother of an individual in the population. Line begins with identification number of individual, followed by a comma, followed by a substring in Python dictionary format with field keys or names and values. The keys are ‘fa’ for father identity and ‘mo’ for mother identity. Values should be identification numbers of the father and mother.. Here is an example: “[kinship_links:] 412, {'fa': 309, 'mo': 306}”. Note that if the father or mother of an individual is not known, the value for that field should be ‘None’. Kinship links are retained in the kinship links dictionary after a simulated individual dies. This contrasts with the invidual fields entries, which are deleted when a simulated individual dies.

Kinship links are listed in sequence in a block of lines that starts after the tag “[kinship_links_start]” and ends immediately before the tag “[kinship_links_end]”.

Module: **linfission.py**

class **lineal_fission**

[paired_allocation_fraction:]: Real number representing fraction of lineages allocated through random, pariwise allocations to fission product key elements. Value should range from 0 through 1.

[lineage_trace_depth:]: Integer representing number of ancestral links of individuals used during assemblage of lineages.

Module: **listfunctions.py** - No input variables

Module: **log.py** - No input variables

Note that module base_popframe provides code that allows reading and writing of some log settings.

Module: **migration.py**

class **migration_age_sex_simple**

[destination_group_element_type:] : String with label type of destination key structure.

[emigrant_group_element_type:]: String with label type of emigrant key structure.

[emigrate_probability:]: Probability that a potential emigrant will emigrate. Value should range from 0 through 1

.

[maximum_age:]: Integer with maximum age of emigrants. Optional

[migration_sex:]: String with sex of candidates for emigration. Value should be ‘m’ or ‘f’. Optional.
[minimum_age:]: Integer with minimum age of emigrants. Optional

[minimum_remainder_count:]: Integer representing minimum number of potential emigrants that must remain. Optional.
[minimum_remainder_fraction:]: Real number representing minimum fraction of potential emigrant pool that must not emigrate. Value should vary from 0 through 1. Optional.

When both minimum remainder count and minimum remainder fraction both are assigned, the value representing the larger number of nonmigrants is used.

Module: **migration_idrisi.py**

class **migration_idrisi**

[population_cost_surface_raster:]:

Module: **pathserver.py**

class **pathserver**

[directory_option:]: Optional. String that specifies location of auxiliary files (such as rasters) for simulation. Value should be 'flat', 'aux' or 'aux_name'. A value of 'flat' indicates auxiliary files occur in same directory as gin and fin files. A value of 'aux' specifieds that auxiliary files will be found in a subdirectory named 'auxstuff' of the directory containing gin and fin files. A value of 'aux_name' leads to creation of an auxiliary files directory in the parent of the directory with gin and fin files, with the name "aux_" + the name of the gin and fin file directory.

Module: **randy.py**
class **randy**

The random number generator may be seed from a single seed value or a file in format expected by the rv.py module.

[random_initialization_seed:]: Integer seed for random number generator.

[random_initialization_file:]: Name of file with sequence of integer values that can be used to initialize the random number generator. Should occur in same directory as gin file for simulation. Extension normally is'.ro'. These files are generated by rv random number generator at end of simulation to save generator state.

Module: **raster_idrisi** - No input variables.

Module: **reproduction.py**

class **reproduction_fixed_age_bir_rndadm**

[minimum_paternal_age:]: Integer representing minimum age of potential fathers.

[fraction_male_births:]: Real number representing fraction of male births.

class **births_Poisson**

[births_Poisson_means:]: Real number with mean number of births for a female in a time period. Means vary with age class.

class **births_binomial**

[births_binomial_rates:]: Real number with births for a female in a time period. Births are spread out over a series of binomial trials, with probability of birth in one trial equal to births per time period divided by binomial trials per period. Value of births per time period divided by binomial trials should range from 0 through 1.

[binomial_trials_per_period:]: Integer with number of birth binomial trials per female per time period.

Module: **reproduction_dd.py**

class **reproduction_dd**

[reproduction_individual_optimal_resources:]: Real number representing an optimum level of resources per individual for reproduction. Levels of available resources greater than the optimum are treated as equal to the optimal.

[reproduction_resources_available:]: Real number with resources affecting reproduction available to a subpopulation.

[reproduction_resources_power:]: Real number with positive value. The ratio of resources available per individual to the optimal level is employed in modeling density dependent reproduction. This variable repreents the power to which the ratio may be raised.

Module: **reproduction_dd_idrisi.py** - No input variables.

Note that class reproduction_dd_idrisi normally shares an individual optimal resources value with class survival_dd_idrisi in module survival_dd_idrisi.py. These classes also share use of a population resources raster. These shared variables are read by methods in class agesex_popframe_idrisi_dd in module agesex_popframe_idrisi.py. This sharing differs from class reproduction_dd in module reproduction_dd.py, which does not share an optimal resource value with class survival_dd in module survival_dd.py.

Module: **simulation.py**

class **simulation**

[current_time:]: Integer with current time period. Value is input as part of initial simulation conditions. Value is revised during simulation executions.

[run_name:]: String with name of a simulation run.

[time_limit:]: Integer with final time period of simulation.

Module: **squarematrix.py** - No input variables.

Module: **survival.py**

class **survival_fixed_age_sex_probabilities**

[female_survival:]: String in format of Python dictionary representing probabilities of survival of female individuals to next age class. Keys of dictionary are integers representing age classes. Values represent survival probabilities and should range from 0 through 1. Value of last age class should be 0.

[male_survival:]: String in format of Python dictionary representing probabilities of survival of male individuals to next age class. Keys of dictionary are integers representing age classes. Values represent survival probabilities and should range from 0 through 1. Value of last age class should be 0.

Module: **survival_dd.py**

class **survival_dd**

[female_survival:]: String in format of Python dictionary representing probabilities of survival of female individuals to next age class. Keys of dictionary are integers representing age classes. Values represent survival probabilities and should range from 0 through 1. Value of last age class should be 0.

[male_survival:]: String in format of Python dictionary representing probabilities of survival of male individuals to next age class. Keys of dictionary are integers representing age classes. Values represent survival probabilities and should range from 0 through 1. Value of last age class should be 0.

[survival_individual_optimal_resources:]: Real number representing an optimum level of resources per individual for survival. Levels of available resources greater than the optimum are treated as equal to the optimal.

[survival_resources_available:]: Real number with resources affecting survival available to a subpopulation.

[survival_resources_power:]: Real number with positive value. The ratio of resources available per individual to the optimal level is employed in modeling density dependent survival. This variable represents the power to which the ratio may be raised.

Module: **survival_dd_idrisi.py** - No input variables.

Note that class reproduction_dd_idrisi normally shares an individual optimal resources value with class survival_dd_idrisi in module survival_dd_idrisi.py. These classes also share use of a population resources raster. These shared variables are read by methods in class agesex_popframe_idrisi_dd in module agesex_popframe_idrisi.py. This sharing differs from class reproduction_dd in module reproduction_dd.py, which does not share an optimal resource value with class survival_dd in module survival_dd.py.

Module: **tally.py** - No input variables

Module: **template**

class **template**

[individual_label:]: String with label used for individual key structures.

[template_start], **[template_end]**: These tags bound one or more lines with string variables containing subpopulation structural templates.

Module **templateserver.py** - No input variables

Module **texio.py** - No input variables

Module: **uniline.py**

class **uniline**

[ancestor_trace_depth:]: Integer representing number of ancestors of live individuals traced and used in identifying lineage membership sets.

[lineage_trace_sex:]: String with sex through which lineages are traced. Value should be 'm' or 'f'

[maximum_apical_clips:]: Integer with maximum number of generations from living apical ancestor used in search for coexisting sublineages.

Module: **unimat.py** - No input variables.
