GENERALIZED
ANIMAL
POPULATION
PROJECTION
SYSTEM

Version 3.0
September 1986

USER'S MANUAL

Conceived and Designed by:

Richard B. Harris
Montana Cooperative Wildlife Research Unit
University of Montana
Missoula, MT 59812

Lee. H. Metzgar
Departments of Zoology and Wildlife Biology
University of Montana
Missoula, MT 59812

Collin D. Bevins
Systems for Environmental Management
Missoula, MT 59801

Programmed by:

Collin D. Bevins
Systems for Environmental Management
Missoula, MT 59801

Disclaimer: We have attempted to find and correct all "bugs" in this program. However, no program can be guaranteed error-free. Users are cautioned to examine results carefully, and to convince themselves that the program has executed its functions correctly.
Disk Contents

The disk you receive with this document should contain the following files:

GAPPS.EXE
   The GAPPS simulation system
INPUT
   Command file options
A.DBF
SA.DBF
POSA.DBF
   Rate file templates
POPSTR.DBF
   Population file structure
CONTENTS.DOC
   This list
POPR.DBF
POPK.DBF
   Two sample population files

In sub-directory /GJ, you should also find a complete set of files needed to replicate the example simulations of Chapter 2 of this manual.

/GJ
GJIN
GJPOP.DBF
GJBREED.DBF
GJMATE.DBF
GJ.TT1.DBF
GJLIT2.DBF
GJLIT3.DBF
GJLIT4.DBF
GJLIT5.DBF
GJGENDER.DBF
GJWEAN.DBF
GJSURV.DBF
GJPOP.RPT
Funding for this project came from:

THE U.S. FISH AND WILDLIFE SERVICE

Office of Grizzly Bear Recovery Coordinator

and

THE U.S. FOREST SERVICE

Office of Grizzly Bear Habitat Coordinator

The assistance of C. Servheen, H. Salwasser and J. Weaver in securing funding is gratefully acknowledged.

Many persons have helped us develop this computer simulation tool. In particular, we wish to thank F. Allendorf, J. Ballou, N. Ryman, J. Glassy, R. Redmond, and M. Gilpin. Special thanks are due A. J. Boyce for allowing us access to his algorithm for computing inbreeding coefficients.

For continuing administrative assistance, we also thank J. Spencer, N. Bernius, G. Schwartz, V. Johnston and S. Fogerty.

This program is public domain. Copying the program and/or this manual is permitted. Publications and reports using this computer tool should contain an appropriate citation.

Questions may be directed to the authors, care of the Montana Cooperative Wildlife Research Unit, University of Montana, Missoula, Montana, 59812.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INTRODUCTION</strong></td>
<td>7</td>
</tr>
<tr>
<td>The history of GAPPS</td>
<td>9</td>
</tr>
<tr>
<td>Questions and Answers</td>
<td>10</td>
</tr>
<tr>
<td>Basic Requirements for using GAPPS</td>
<td>16</td>
</tr>
<tr>
<td>Hardware</td>
<td>16</td>
</tr>
<tr>
<td>Software</td>
<td>16</td>
</tr>
<tr>
<td>How to use this manual</td>
<td>17</td>
</tr>
<tr>
<td><strong>OVERVIEW: A QUICK SAMPLE SESSION</strong></td>
<td>19</td>
</tr>
<tr>
<td>The problem</td>
<td>19</td>
</tr>
<tr>
<td>Available data</td>
<td>19</td>
</tr>
<tr>
<td>Life history data summary</td>
<td>21</td>
</tr>
<tr>
<td>Command file</td>
<td>22</td>
</tr>
<tr>
<td>Rate files</td>
<td>23</td>
</tr>
<tr>
<td>Output</td>
<td>26</td>
</tr>
<tr>
<td><strong>FILE HANDLING IN GAPPS</strong></td>
<td>27</td>
</tr>
<tr>
<td><strong>THE COMMAND FILE</strong></td>
<td>31</td>
</tr>
<tr>
<td>INPUT</td>
<td>31</td>
</tr>
<tr>
<td>Command file concepts</td>
<td>38</td>
</tr>
<tr>
<td>Editing the Command file</td>
<td>41</td>
</tr>
<tr>
<td>PARENTHESES</td>
<td>41</td>
</tr>
<tr>
<td>SUBCOMMANDS</td>
<td>43</td>
</tr>
<tr>
<td>COMMENTS</td>
<td>45</td>
</tr>
<tr>
<td>COMMAND ORDER</td>
<td>45</td>
</tr>
<tr>
<td>REPETITION</td>
<td>47</td>
</tr>
<tr>
<td><strong>THE POPULATION FILE</strong></td>
<td>49</td>
</tr>
<tr>
<td>The use of traits in GAPPS</td>
<td>49</td>
</tr>
<tr>
<td>Creating and editing population files</td>
<td>51</td>
</tr>
<tr>
<td><strong>RATE FILES</strong></td>
<td>55</td>
</tr>
<tr>
<td>Types</td>
<td>55</td>
</tr>
<tr>
<td>Format</td>
<td>56</td>
</tr>
<tr>
<td>Function CON (Constant)</td>
<td>59</td>
</tr>
<tr>
<td>Function SL (Straight line)</td>
<td>59</td>
</tr>
<tr>
<td>Function MM (Modified Michaelis-Menton)</td>
<td>60</td>
</tr>
<tr>
<td><strong>HOW TO RUN GAPPS</strong></td>
<td>64</td>
</tr>
<tr>
<td><strong>MODULES IN GAPPS</strong></td>
<td>66</td>
</tr>
<tr>
<td><strong>ROOT MODULES</strong></td>
<td>66</td>
</tr>
<tr>
<td>BEGIN ()</td>
<td>66</td>
</tr>
<tr>
<td>END ()</td>
<td>67</td>
</tr>
<tr>
<td>RESEED ()</td>
<td>68</td>
</tr>
<tr>
<td>SET ()</td>
<td>70</td>
</tr>
<tr>
<td>ABS(arg1) Function</td>
<td>71</td>
</tr>
</tbody>
</table>
CEIL(arg1) Function 71
EXP(arg1) Function 71
FLOOR(arg1) Function 72
LN(arg1) Function 72
LOG(arg1) Function 72
MAX(arg1, arg2) Function 72
MIN(arg1, arg2) Function 72
MM(arg1, arg2, arg3, arg4, arg5) Function 72
NRAND(arg1, arg2) Function 72
ROUND(arg1) Function 73
SORT(arg1) Function 73
URAND(arg1, arg2) Function 73

SETUP() 74
SUMMARY() 75

EVENT MODULES NOT REQUIRING RATE FILES 77
AGE() 77
ALLELES() 78
CENSUS() 80
INBRED() 82
PACK() 84
PURGE() 86

EVENT MODULES REQUIRING RATE FILES 87
BIRTH() 87
BREED() 89
HUNT() 90
IMMIGRATE() 92
LITTER() 94
MATE() 95
READ() 96
SURVIVE() 97
WEAN() 99
WRITE() 100

COMMAND-SUBCOMMAND CROSS REFERENCE TABLE 102

OUTPUT FILES 104
AGESEX FILES 104
CASES FILES 105
REPORT FILES 106
REPORT Files 106
GENETICS REPORT Files 107
INBREED REPORT Files 107
SUMMARY REPORT Files 107

TRACE FILES 108

EXAMPLES 109
Example 1: Comparing age-structures from hunts and those from the standing population 109
Example 2: A population with separate density-dependent functions for males and females. 110
Example 3: An inversely density-dependent function for breeding. 111
Example 4: Bringing in an average of 0.5 112
Example 5: Examining the loss of genetic diversity at the hypothetical locus.

Example 6: Examining the variation in the loss of genetic diversity given the same set of demographic events.

Example 7: Examining the loss of genetic diversity by using the increase in average population inbreeding level.

Example 8: Simulate inbreeding depression among inbred newborns.

Example 9: An animal that lives more than 30 years: Elephants.

Example 10: Some animals breed more than once yearly.
INTRODUCTION

GAPPS (Generalized Animal Population Projection System) is a computerized system that allows simulating animal populations under a wide variety of circumstances and with a wide variety of options. GAPPS is not a simulation model itself; rather, it is a system that allows the user to create a simulation model without having to program it him/herself. GAPPS can be thought of as a simple population simulation language. It has a syntax, contains various rules and limitations, and takes some time to learn. However, it is much easier to create a population projection simulation using GAPPS than to start from scratch using a programming language (e.g. FORTRAN, BASIC, PASCAL, etc.) because GAPPS already knows how to do many things. You need only know how to tell it what you want it to do.

The keynote of GAPPS is flexibility. For example, in GAPPS, you can arrange the life-history events that typically enter a population projection model (e.g. birth, death, immigration) in any order you wish. Further, you can repeat these functions numerous times within each cycle, and each time the event occurs the rates governing them may be different. Flexibility extends to the rates themselves; in addition to assigning a single value to a rate, functions that relate the rate to a population-wide characteristic (e.g. census population) may be defined. Flexibility also characterizes the output of GAPPS. Data files containing the results of many different events may be produced.

GAPPS primarily projects a single population. It is not useful for creating, for example an ecosystem or a predator-prey model. It's primary use is the investigation of the properties of a specific population. However, we realize that no population exists in a vacuum. Therefore, external forces that may
influence the population in question may be modeled. Examples include hunting, carrying capacity, and immigration of conspecifics. However, the focus is always on the population of interest, so that, for example, while hunting can be modeled in a number of ways, the population of hunters cannot be explicitly modeled.

GAPPS deals with individual animals, not with entire cohorts. Because of this feature, GAPPS is stochastic, not deterministic. Rates for all events that affect the population (e.g., survival, breeding, etc.) are more properly considered probabilities. The algorithm underlying all population events is the choosing of a random number from a uniform distribution between 0 and 1. For each event, each individual has a critical value which is set by a rate file. When the individual is subjected to the event, the individual's critical value is compared with the chosen uniform random number, and a decision is made. Thus, in GAPPS, one cannot demand that exactly a specified number of individuals die, or give birth, or immigrate into the population. One can specify a value (i.e., a mean) that, over enough repeated runs, will characterize the event. But in any one trial, one can never be certain of exactly the number of animals that will do a certain thing (unless, of course, one specifies the probability of the event occurring as 1.0 or 0.0).

Because GAPPS deals with individual animals, its best uses will be for simulating relatively small populations. In small populations, the random nature of discrete events is important. In large populations, the 'law of averages' tends to diminish the importance of random, discrete events. Additionally, because GAPPS deals with discrete individuals, it runs more slowly than simulation programs that apply rates to entire cohorts or populations. In developing GAPPS, we have considered its primary use to be with vertebrate populations. Its advantages will be more pronounced when the species being modeled is typified by small population size, extended parental care, or complex age-structures.
There are numerous advantages to working with animals individually. Each individual can be assigned a number of traits, and these traits can be used in differentiating segments of the population, or can be useful in generating certain kinds of data. For example, for each animal, GAPPS knows not only its sex and its age, but who its mother and father were, whether it is still under its mother's care, and if female, the identity of its offspring. Entire pedigrees can thus be traced through time and inbreeding coefficients computed for each individual. In addition, the random component of each event adds a degree of reality that is lost in completely deterministic models. Beyond simply asking the question: 'what will most likely happen?', one can also ask: 'how likely are other outcomes'?

The history of GAPPS

GAPPS evolved from a simulation model built for the purpose of evaluating age-structures of grizzly bear (Ursus arctos) populations. We required the model to be stochastic because we were interested in the sample sizes necessary to detect age-structure changes in the face of random variability. The original simulation was built in FORTRAN and designed to run on a large, main-frame computer.

Because grizzly bears are long-lived, have extended parental care, and complex age-structures, the model was quite involved. We realized that, with only a bit more complexity, it would be possible to model almost any long-lived, seasonally breeding vertebrate species. When we decided to expand the capabilities of the original grizzly bear model, it was a relatively short step to create a totally flexible program that would handle a variety of species.

At this point, we made the decision to change the language of the program from FORTRAN to C, and the computers to operate it from main-frames to micros. Extensive re-programming was necessary, as more and more of the particular items we needed
were removed from the computer code and put into user-accessible locations. Our constant goal was to reduce the number of decisions that had to be 'hard-wired' into the program, and thus give the user more and more flexibility to mold the program at his/her will. The final result was GAPPS, conceptually different from the original grizzly bear model, but able to emulate it perfectly.

Questions and Answers

Who is GAPPS intended for?

GAPPS is intended for biologists, wildlife managers, biometricians, researchers, students; anyone with an interest in modelling a specific situation. All that is required is a modest amount of computer skill (e.g. the ability to use a screen editor), and access to an appropriate computer. Generally, its best use will be for a specific application, where specific questions are being asked. Simpler models may perform better for analyses that are more general or heuristic.

What are some typical uses for a model using GAPPS?

One might simply want to project a population to view its rate of growth (and the distribution of probable rates of growth). However, GAPPS is also most useful for examining auxiliary information. For example, one can view the sex and age structure of the population at any time. One might want to experiment with different types of harvests to see their effects on the population. Or, one might be interested in the age-structure of the harvested animals, since that is often a data source useful to wildlife managers. One can also investigate the amount of inbreeding likely to occur under a variety of assumptions, using the pedigree analysis module. In fact, one can estimate the genetically effective population size (Hartl 1980) during a run by using the population-wide F statistic. We would emphasize that a clearly specified objective in using GAPPS
should always precede actual modelling.

What should I use GAPPS for?

Any specific application where having individual animals treated discretely is an advantage. We do not recommend GAPPS for highly general or 'heuristic' applications, where a simple series of equations might do the trick.

Why does GAPPS treat only individual animals?

Because the demographic stochasticity produced by treating discrete events is important in the small populations that typify many vertebrate species. Users are also able to create innumerable subdivisions of the populations, using the population attributes. Treating individual animals also allows incorporation of genetics.

Can GAPPS treat cohorts, or entire populations?

No.

Why isn't GAPPS a 'model'?

The term 'model' implies that many assumptions about the way a population works have been made, and that rates governing births, deaths, etc. have been estimated. GAPPS does not make the assumptions, it allows the user to make them. GAPPS does not estimate rates governing life-history events, or even demand the form they be in. It allows the user to estimate rates, and decide whether rates are constants, or vary in some way with population density, carrying capacity, weather, etc. Thus GAPPS is a framework, allowing the user to build his/her own 'model' without having to program everything from scratch.

Why shouldn't I just write my own model from scratch?

Maybe you should. There is certainly much to be learned about how you conceive of your population operating from writing a model from scratch. Additionally, you may simply enjoy the process of constructing a model that accurately reflects your theoretical world.
challenge of computer programming. However, programming takes much time, and if you're not already proficient, errors can arise that have nothing to do with the biology of interest. You may find yourself spending much time finding a bug in the program you've written, leaving little time for thinking about the implications of the model itself. The theory underlying GAPPS is that it should largely free you from the technical tasks involved in programming, allowing you to focus your attention on the biology involved in modeling.

Are there things GAPPS won't do?
Certainly. Every computer program has limitations. We've already mentioned the fact that GAPPS will not treat cohorts as entities, i.e., that it cannot completely emulate a Leslie-matrix. As presently written, it does not treat the spatial component of populations. Individuals within the population of interest are assumed to mingle with each other at will. Depending on your specific application, this may or may not be a severe limitation.

How large a population will GAPPS model?
No more than approximately 16,000 individuals may exist during any 1 simulation run. This 16,000 figure can be exceeded by running a small population for a very large number of years, or by modeling a very large population. If it is exceeded, new animals will be given identification numbers that will overwrite existing animals (e.g. the 16,001st animal will be assigned #1, thus erasing all information about the already existing animal #1). In general, there is little to be gained by simulating populations larger than a few thousand, because the dynamics are likely to be similar to those of a simpler deterministic model.

Will GAPPS model density-dependent relationships?
Yes. The user may specify up to 4 parameters which can be
used to identify a function relating a rate-probability to a population-wide characteristic (e.g. population size). You may also specify the form of the density-dependent function. In its current implementation, GAPPS already recognizes 2: the straight-line function and a version of a Michaelis-Menton equation. Other functions may be built by users using the SET command.

Can GAPPS be used to model the genetics of small populations?

Yes. In GAPPS, each individual animal carries with it 2 alleles at a hypothetical locus. Newborns receive 1 allele from their mother and 1 from their father in simple Mendelian fashion. Both the total number of alleles in the population, and the population-wide heterozygosity may be reported. In addition, GAPPS will perform pedigree analysis on all newborns to determine their inbreeding coefficient.

Will GAPPS recognize different rates for different age/sex classes?

Yes. GAPPS allows up to 30 age-classes, but no more than 2 sexes.

But I don't have enough data on my population to do this sort of modeling.

Few of us do. One is never forced by GAPPS to devise complex rate structures; one is simply given the freedom to do so if one wishes. However, there may be situations in which you may wish to take your best guess at various age-specific rates, or density-dependent functions, using whatever data are available, because leaving them out completely may seem less realistic than using reasonable guesses.

What kind of computer will GAPPS run on?

Any IBM-compatible (PC or AT 'look-alike') with the minimum memory and storage requirements should be able to run GAPPS.
Are there memory or storage requirements?

Yes, but they will depend in part on how large a population is modeled, and how complex your model is. We recommend running GAPPS on a machine with 640K available memory, although it can be run on a machine with 512K, with fewer than 16,000 animals being allowable. Storage requirements will also vary with the specific application. We generally recommend operating GAPPS with a hard disk, although the program itself will fit on a standard 360K floppy disk. It is possible to create enormous data output files with GAPPS, and large storage areas may be required to handle them.

Do I need any other software?

Yes. GAPPS is designed to run under MS-DOS or PC-DOS 2.+, so you'll need that type of operating system. Some sort of screen editor will be necessary to manage the Command file. Most DOS systems come with one; you may use a word-processor as well if the resultant file can be outputted in ASCII format. Additionally, rate files in GAPPS are created and maintained by dBASEII\(^1\), so you'll need to have a copy of that. If you want to write your own modules to add to the existing code of GAPPS, you'll need the Computer Innovations C86 compiler for the C programming language, to link together the new module with the existing code. The vast majority of applications will not require extensions to the existing code, and so will not require a C compiler.

Is GAPPS written in dBASEII?

No, it simply uses dBASEII as an editor for the rate files. GAPPS knows how to read files written in the dBASEII language. GAPPS is written in the C programming language.

\(^1\)dBASE is a registered trademark of Ashton-Tate, Inc.
Why the C programming language?

C has a number of advantages for this kind of program. Foremost among these is that it is useful for the kind of modular programming characterizing GAPPS. Everything that happens in GAPPS is in its own little module. This way, modules can be switched in order, removed, or new ones added, and no undesired changes in the code occur. Additionally, C is amenable to screen-oriented micro-computers, it is terse (so that code doesn't take up scarce storage space), and it runs quickly on most PC's. Unfortunately, most biologists don't understand C, but perhaps after GAPPS this will change!

Do I have to understand C to use GAPPS?

No, you only have to be able to edit the command file, and deal with the rate files through dBASEII. Only if you want to add modules to GAPPS that don't already exist must you understand the C programming language. Most applications will not require adding modules.

What kind of output does GAPPS produce?

GAPPS produces tabular output of many sorts. Output from GAPPS can be voluminous, but it will always be in a form that is easily accessible to data base managers, statistical software, or user produced analysis programs. See the section on output in the chapter on file handling for details.

Why doesn't GAPPS have graphics output?

To keep flexibility at a maximum, we have kept output tabular. If graphics were incorporated, what would we graph? Total population size? Population size of males only? Males above a certain age? Population inbreeding coefficient? Number of 3-year old females that have never bred dying from spring hunts? As you can see, the choice of variables would be enormous for any graphing. Users with programming expertise are free to devise graphic routines, either within or external to GAPPS. For
a simple model, we refer the interested reader to SIMCON, developed at the University of British Columbia by Dr. Carl Walters. SIMCON graphs variables specified by the user, who must also develop the specific equations governing the simulation.

How long does GAPPS take to run?

It depends on a number of factors, including the type of machine you're using, the size of the population you're modeling, the complexity of your particular model, and how much data input and output is required.

Basic Requirements for using GAPPS

Hardware.

GAPPS is designed to run on any IBM-PC, IBM-AT, or compatible. Your copy will originally be on a floppy disk. We recommend making a copy of this disk, of course. Users with a hard disk available will most often want to copy GAPPS to the hard disk. A hard disk is not essential, but we believe, most convenient, especially for the storage of large quantities of output data.

You must also have an 8087 type math co-processor on your machine.

Software.

The following is required to run GAPPS:

- The GAPPS executable program, called GAPPS.EXE.
- A command file. The command file that you receive with the program is called simply INPUT. We recommend that you make a few copies of this and store them in a safe place. Then, you may copy INPUT to other file names, and edit them at will when actually running GAPPS.
- A screen editor, to edit the Command file.
- dBASEII, for accessing and editing data files.
- A series of files governing the rates in your simulation.
These will be written in dBASEII, using its editor. You may create as many as 256 of these files, and name them anything you want.

- MS-DOS or PC-DOS 2.00 or greater operating system.

Additionally, if you wish to add modules to the existing GAPPS code, you will need a C compiler and a linker.

How to use this manual

Presumably, if you've gotten this far you've proven that you're willing to read User's Manuals. This manual does not have a learning or tutorial section. All the requirements of using GAPPS to develop a population simulation are described in their appropriate section. We have taken the perspective that users best learn how to operate a particular part of a program when they really need that part. The index at the back should help if you get lost.

After reading this introduction, take some time to study the next section, entitled Overview: a quick sample session. It will give you a feel for how GAPPS works, although you probably won't understand everything at first. You can also look directly at some of the complete Examples, toward the end of the Manual. Both sections should provide you with the general picture. Next, take some time to familiarize yourself with the concept of a population file. Similar to this is the rate file, both using dBASEII. Finally, the section on modules is fairly specific.

Rather than trying out a complex model at first, you might want to try some very simple structures, and gradually work your way into using multi-parameter rates, lots of output, complex yearly cycles, etc.

For users comfortable with slightly more technical language, we refer you to GAPPS: Generalized Animal Population Projection System Technical Documentation by Collin D. Bevins. Please write
to the Montana Cooperative Wildlife Research Unit, University of Montana, Missoula, Montana 59812 if you desire a copy of this technical documentation.
OVERVIEW: A QUICK SAMPLE SESSION

This section presents just one application for which GAPPs might be used. It is not intended to be thorough, rather it should give you the flavor of how GAPPs works. The example is a simple one, and is rather silly. However, getting comfortable with GAPPs may require writing simple simulations before you are ready to include the sophistication and detail which GAPPs can handle. All the files you need to re-create this example are included on your disk in a sub-directory called 3GT. After you have learned how to operate GAPPs, a good idea would be to try to recreate this example.

The problem:

Four male and six female Gappjumpers (Electronicus fecundensis) have been established on a remote desert island from which no escape is possible. No other competing species exist on the island, nor are there any predators. The habitat is known to be optimal for Gappjumpers, and a rapid increase in numbers can be expected. For a short time, we can expect there to be little if any response in vital rates to increasing density. We wish to know how many Gappjumpers will exist on the island in 10 years. Further, we would like to estimate the parameter in a statistical sense; that is, know the distribution of years at which 100 animals exist on the island.

Available data:

From zoo studies, estimates of age-specific fecundity and survivorship are available. The following information summarizes our knowledge of Gappjumper population biology:

- Breeding season is in the spring. Births occur the following summer.

- Most females first breed at age 2 3/4, although a few
breed at age 1 3/4. Thus, some females give birth on their second birthday, although most do so on their 3rd birthday. No observations were made of initial reproductive activity earlier than 1 3/4 years of age, and all females had bred by age 2 3/4. Males first participate in breeding at age 2 3/4.

- Litter size averages 4, with roughly half of all litters consisting of 4 and about one-quarter 3 and one-quarter 5.

- Young stay with their mother for over a year. Adult females breed only every other year, not every year. Young are not known to stay with the mother for over 1 1/2 years. Thus, 2 years is the known breeding cycle.

- We have no reason to suspect that, once adulthood is attained, there is any relation between age and litter size, breeding cycle, or probability of breeding.

- We have no data to suggest deviations from a 1:1 sex ratio of young Gappjumpers at birth.

- Most natural mortality is caused by Gappjumpers getting stuck in the mud that forms around the seasonal water holes on the island during the dry season. Thus, we can reasonably model mortality as a once-per-year phenomenon. Young Gappjumpers, being weaker, are less able to extricate themselves from mud, and thus have higher mortality rates. Similarly, very old Gappjumpers (>10 years old) become senile, and are occasionally known to wallow in mud seemingly with great enjoyment. They also have higher mortality rates.

- Because this is an uninhabited island, neither immigration nor hunting are applicable.

- The best time to census Gappjumpers is at the end of the dry season, just after most of the mortality has occurred.
Life history data summary:

<table>
<thead>
<tr>
<th>Age</th>
<th>Survival Probability</th>
<th>Breeding Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Females</td>
<td>Males</td>
</tr>
<tr>
<td>0</td>
<td>0.75</td>
<td>0.75</td>
</tr>
<tr>
<td>1</td>
<td>0.85</td>
<td>0.85</td>
</tr>
<tr>
<td>2</td>
<td>0.85</td>
<td>0.85</td>
</tr>
<tr>
<td>3-9</td>
<td>0.85</td>
<td>0.85</td>
</tr>
<tr>
<td>10+</td>
<td>0.50</td>
<td>0.50</td>
</tr>
</tbody>
</table>

Note that the breeding probability for females is only for those females whose previous young have left.

For females of all ages:

<table>
<thead>
<tr>
<th>Litter Size</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td>0.25</td>
<td>0.50</td>
<td>0.25</td>
</tr>
</tbody>
</table>

and all young are 'weaned' at age 1.5.

The initial transplants are as follows:

<table>
<thead>
<tr>
<th>Age</th>
<th>Females</th>
<th>Males</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The 4 zero-year old animals are offspring of the 3-year old female.
To run GAPPs, we set up the following Command file, and call it GJIN:

```
SETUP(
  TITLE 0.0  ("Gappjumpers run amok")  1.0
BEGIN(
  RUN 0.1  (50)
)
READ(
  INPUT (GJPOP.DBF)
; Comment: start simulation;
BEGIN(
  YEAR (10)
; Comment: These events happen each year;
)
BREED(
  RATES (GJXBREED.DBF)
)
MATE(
  RATES (GJMATE.DBF)
)
LITTER(
  RATES(GJLIT1.DBF,GJLIT2.DBF,GJLIT3.DBF,
        GJLIT4.DBF,GJLIT5.DBF)
)
AGE()
; Comment: Animals ages are incremented by 1 year at this point;
BIRTH(
  RATES (GJGENDER.DBF)
  VAR (BIRTHS)
)
SURVIVE(
  RATES (GJMORT.DBF)
  VAR (DEATHS)
)
CENSUS(
  VAR (POP))
WEAN(
  RATES (GJWEAN.DBF))
SUMMARY(
  REPORT (GJPOP.RPT)
  SYMBOLS (RUN,YEAR,BIRTHS,DEATHS,POP))
END()
PURGE()
END()
```

The SETUP group of commands includes a title for screen display. The READ command tells GAPPs to search for a population file.
file (which will usually have a .DBF extension), in this case called GJPOP.DBF.

Comments are allowed anywhere in the command file as long as they are enclosed on both sides by a semi-colon (;). They can be useful to remind yourself what is going on at any point.

Commands that are actually executed in the running of the simulation are situated after the BEGIN command. The 2 important subcommands under BEGIN are RUN and YEAR. RUN simply informs GAPPS to repeat every command following for as many times as specified (50 in this case). YEAR similarly informs GAPPS to repeat all the events following for the specified number of years (10 in this case).

The yearly events specified in this model of Gappjumpers are BREED, MATE, LITTER, AGE, BIRTH, SURVIVE, CENSUS, WEAN, and SUMMARY. They will always occur in exactly this order. CENSUS and SUMMARY are information transfer modules; in this application, they do not affect the other population processes. AGE increments each animal's age by 1 year (notice that it occurs just before BIRTH; in an organism that breeds at a similar time each year, females give birth on their birthday). All the other modules must have rate files, which contain the rates governing their occurrence. Notice that for LITTER, there is a .DBF for each of the 5 possible litter sizes.

The VAR command is used to store event results under variable names which may be used in rate files or the SUMMARY module. All specified variables may appear in a file as part of the summary output if they are specified by the SYMBOLS command within SUMMARY. In this case, a summary report file called GJPOP.RPT will be created, and will have data on the number in the population, and the number of births and deaths.

Rate files:
To run this simulation, rate files GJBREED.DBF, GJMATE.DBF, GJLIT1.DBF, GJLIT2.DBF, GJLIT3.DBF, GJLIT4.DBF, GJLIT5.DBF, GJGENDER.DBF, GJMORT.DBF, and GJWEAN.DBF must be in existence. Shown below are the first few lines from each of
(Litter size 3)

<table>
<thead>
<tr>
<th>Age</th>
<th>Function</th>
<th>PARM1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CON</td>
<td>0.2500</td>
</tr>
<tr>
<td>2</td>
<td>CON</td>
<td>0.2500</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Litter size 4)

<table>
<thead>
<tr>
<th>Age</th>
<th>Function</th>
<th>PARM1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CON</td>
<td>0.5000</td>
</tr>
<tr>
<td>2</td>
<td>CON</td>
<td>0.5000</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...(Litter size 5)

<table>
<thead>
<tr>
<th>Age</th>
<th>Function</th>
<th>PARM1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CON</td>
<td>0.2500</td>
</tr>
<tr>
<td>2</td>
<td>CON</td>
<td>0.2500</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

... from GJGENDER.DBF. Entries under PARM1 show the probability of a newborn being a female, by age of the mother.

<table>
<thead>
<tr>
<th>Age</th>
<th>Function</th>
<th>PARM1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CON</td>
<td>0.5000</td>
</tr>
<tr>
<td>2</td>
<td>CON</td>
<td>0.5000</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...(Litter size 5)

<table>
<thead>
<tr>
<th>Age</th>
<th>Function</th>
<th>PARM1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CON</td>
<td>0.2500</td>
</tr>
<tr>
<td>2</td>
<td>CON</td>
<td>0.2500</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

... from GJMORT.DBF. Entries under PARM1 show the probability of surviving each mortality event, by sex, age, and family status.

<table>
<thead>
<tr>
<th>Parents</th>
<th>Offspring</th>
<th>Sex</th>
<th>Age</th>
<th>Function</th>
<th>PARM1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>CON</td>
<td>0.7500</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>CON</td>
<td>0.8500</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>CON</td>
<td>0.8500</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>CON</td>
<td>0.7500</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>CON</td>
<td>0.8500</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>CON</td>
<td>0.8500</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

... from GJWEAN.DBF. Entries under PARM1 show the probability of a litter being cast off by a mother, by age of the offspring.
<table>
<thead>
<tr>
<th>Age</th>
<th>Function</th>
<th>PARM1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CON</td>
<td>0.0000</td>
</tr>
<tr>
<td>2</td>
<td>CON</td>
<td>0.1000</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Output:**

After being run, the user will have a new file called GJPOP.RPT. This file will contain each the number of births, deaths, and the final population in each year, for each run. The beginning of GJPOP.RPT is produced below.

<table>
<thead>
<tr>
<th>RUN</th>
<th>YEAR</th>
<th>BIRTHS</th>
<th>DEATHS</th>
<th>POP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>13</td>
<td>6</td>
<td>17</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>18</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>22</td>
<td>12</td>
<td>28</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>27</td>
<td>12</td>
<td>45</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>18</td>
<td>11</td>
<td>52</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>45</td>
<td>18</td>
<td>79</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>31</td>
<td>24</td>
<td>86</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>60</td>
<td>22</td>
<td>124</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>48</td>
<td>23</td>
<td>149</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>13</td>
<td>2</td>
<td>21</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>9</td>
<td>3</td>
<td>27</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**GJPOP.RPT** can now be used to answer the original question: What is the expected distribution of population sizes at year 10 of this growing Gappjumper population? At this point, the user can either determine this by hand, use an already existing statistics program, or write another computer program that will calculate the desired summary data. For our example, the mean population size at year 10 was 156.74, with a standard deviation of 57.43.
This chapter details the different kinds of files involved with running GAPPS. A generalized flow-chart of file-handling is shown on Fig. 3.1.

The heart of the program is the set of executable functions that know how to interpret all the other data given it. This resides in GAPPS.EXE. You can visualize GAPPS.EXE as a kind of traffic cop, directing the flow of information from the population file to the output data file, using information about how to structure the simulation found in the command file, and information about the particular population, found in the rate files (.DBF files). Short of adding modules to the basic program, you will have no reason to deal with GAPPS.EXE.

The Command file structures the simulation. By convention, the Command file does not have an extension as part of its name, although it could. It tells GAPPS such necessary information as how many years the simulations are to last, how many replicate loops to run, what sorts of output are desired, etc. More importantly, it is within the Command file that the biology of the organism is actually "modeled". The life-history events that are to occur, how often each year, and their exact order are set in the Command file. Chapter 4 details options available with the Command file.

The rates that govern the simulation (and thus make GAPPS simulate a specific population of interest) are contained in rate files. Rate files are written in, and accessed by dRASETT. A separate Rate file is necessary for each event module called in the Command file. (An event module is one with the capacity to change characteristics of individuals in some way, such as BIRTH, or HUNT. Note that if event module LITTER is called, up to 12 Rate files must be created, 1 for each possible litter size. Rate files are covered in detail in Chapter 6.
To run a simulation, a population file must be in existence. GAPPS cannot create a population out of thin air. It can alter existing population files, and these new files may be used as population files, but there must exist a file containing 'founder' individuals. Population files are explained in detail in Chapter 5.

Output files may take a variety of forms, and may be named almost anything. Four forms that are possible in almost all of the modules are CASES, AGSEX, REPORT and TRACE.

A CASE file consists of records of individuals as they are processed by that particular module. For example, a CASE file from the SURVIVE module consists of all the animals that died during the run. An AGSEX file consists of a matrix containing the animals as in the CASE file, but specified by age and sex. For each simulation, for each year, 2 records are produced, 1 for females, 1 for males. Within each record are the numbers in each age-class processed by that module, from 0 to 29. For example, the AGSEX file produced by the CENSUS module is simply the sex/age matrix of all animals censused during that year, while the AGSEX file produced by SURVIVE is the sex/age matrix of all animals dying during that particular mortality event. A TRACE file gives summary information about the number of individuals processed by the module.

Other modules have some specialized types of outputs which are accessed through the REPORT subcommand. These are detailed in the section on output. SUMMARY produces a file of exactly what is seen on the screen as the simulation progresses.
Fig. 3.1. Generalized flow chart showing file handling in GAPPS.
To aid in clarity of file handling, we suggest the following protocol in naming files:

<table>
<thead>
<tr>
<th>Type of files</th>
<th>Extension name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CASE</td>
<td>.CAS</td>
</tr>
<tr>
<td>AGESEX</td>
<td>.DAT</td>
</tr>
<tr>
<td>REPORT</td>
<td>.RPT</td>
</tr>
<tr>
<td>TRACE</td>
<td>.TRC</td>
</tr>
<tr>
<td>RATE</td>
<td>.DBF</td>
</tr>
<tr>
<td>POPULATION</td>
<td>.DBF</td>
</tr>
<tr>
<td>COMMAND</td>
<td>no extension</td>
</tr>
<tr>
<td>ERROR</td>
<td>.ERR</td>
</tr>
</tbody>
</table>
THE COMMAND FILE

Structuring the simulation is accomplished via the Command file. The first thing GAPPS does when executed is to read the specified Command file. It will look for syntax errors and alert you to any it finds. It will not proceed with the simulation if any errors are detected.

Many modern computer programs attempt to be "user friendly" by running via menus. We have deliberately omitted menus in GAPPS, because the number and complexity of menus needed to allow the full flexibility in modeling would be overwhelming, and likely more confusing than editing a simple command file. We make the assumption that users of GAPPS are capable of using a screen editor to mold the Command file so that the resulting simulation will do just what it should.

With your copy of GAPPS, you receive a Command file called INPUT. It is shown in Fig. 4.1. INPUT contains every possible command option. It is not necessary to include them all. Much simpler Command files are possible (it is possible to create even more complex files by repeating commands).

Fig 4.1. File INPUT, the complete listing of commands and subcommand in GAPPS.

; GAPPS 3.0 LANGUAGE SYNTAX ;

; AGE() - increments animal's ages ;
AGE   
   AGESEX   (   filespec    )
   CASES    (   filespec    )
   PAUSE    (  [ON : OFF]  )
OFF]

   START    (   integer    )
   STEP     (   integer    )
   STOP     (   integer    )
   SWITCH   (  [ON : OFF]  )
   TRACE    (   filespec    )
   VAR      (   symbol     ) ; stores population count ;
; ALLELES() - initializes animals' alleles for MIN : MAX
diversity & zygosity;

ALLELES(
  AGESEX (filespec)
  CASES (filespec)
  DIVERSITY ([MIN : MAX])
  HETERO ([MIN : MAX])
  PAUSE ([ON : OFF])
  START (integer)
  STEP (integer)
  STOP (integer)
  SWITCH ([ON : OFF])
  TRACE (filespec)
  VAR (symbol) ; no of unique alleles in population ;
)

; BEGIN() - marks beginning of RUNS or YEAR ;

BEGIN(
  PAUSE ([ON : OFF])
  RUN (integer) ; either RUN or YEAR required ;
  YEAR (integer) ; either RUN or YEAR required ;
)

; BIRTH() - creates new animal records for mothers with litters

BIRTH(
  AGESEX (filespec) ; births by mother's age class ;
  CASES (filespec)
  PAUSE ([ON : OFF])
  RATES (filespec) ; required ;
  REPORT (filespec)
  SEED (integer) ; reseed genetic random number gen.
  START (integer) ; 0 = use clock;
  STEP (integer)
  STOP (integer)
  SWITCH ([ON : OFF])
  TRACE (filespec)
  VAR (symbol) ; stores number born ;
)

; BREED() - selects females to be bred ;

BREED(
  AGESEX (filespec) ; bred by age class ;
  CASES (filespec)
  DEPRESS (real : symbol) ; inbreeding depression factor;
  PAUSE ([ON : OFF])
  RATES (filespec) ; required ;
  REPORT (filespec)
  START (integer)
  STEP (integer)
  STOP (integer)
  SWITCH ([ON : OFF])
)
TRACE ( filespec )
VAR ( symbol ) ; stores number bred;

; CENSUS() - takes census of current population;
CENSUS(
AGES ( integer, integer, ... , integer)
AGESEX ( filespec ) ; entire population;
CASES ( filespec )
PAUSE ( [ON : OFF] )
SEXES ( [BOTH : FEMALES : MALES] )
START ( integer )
STEP ( integer )
STOP ( integer )
SWITCH ( [ON : OFF] )
TRACE ( filespec )

; *
VAR ( symbol ) ; census count - required;

; END() - marks end of RUN or YEAR;
END ()

; GENETICS() - determines population allelic diversity and zygosity;
GENETICS(
AGESEX ( filespec ) ; entire population;
CASES ( filespec )
PAUSE ( [ON : OFF] )
REPORT ( filespec )
START ( integer )
STEP ( integer )
STOP ( integer )
SWITCH ( [ON : OFF] )
TRACE ( filespec )
VAR ( symbol ) ; stores population heterozygosity;
)

; HUNT() - selects and removes animals due to hunting mortality
HUNT(
AGESEX ( filespec ) ; killed animals only;
CASES ( filespec )

; *
NUMBER ( [symbol : real] ) ; required;
PAUSE ( [ON : OFF] )

; *
RATES ( filespec ) ; required;
REPORT ( filespec )
START ( integer )
STEP ( integer )
STOP ( integer )
SWITCH ( [ON : OFF] )
TRACE ( filespec )
VAR ( symbol ) ; stores number killed;
; IMMIGRATE() - creates and adds immigrants to population;
IMMIGRATE(
    AGESEX   ( filespec ) ; immigrants only;
    BASE     ( [symbol : real] ) ; allele base;
    CASES    ( filespec )
    DIVERSITY( [MIN : MAX] )
    HETERO   ( [MIN : MAX] )
    ;*
    NUMBER   ( [symbol : real] ) ; required;
    PAUSE    ( [ON : OFF] )
    ;*
    RATES    ( filespec ) ; required;
    REPORT   ( filespec )
    START    ( integer )
    STEP     ( integer )
    STOP     ( integer )
    SWITCH   ( [ON : OFF] )
    TAG      ( integer )
    TRACE    ( filespec )
    VAR      ( symbol ) ; stores number of immigrants created;
)

; INBREED() - determines animals inbreeding coefficient F;
INBREED  ( AGESEX   ( filespec ) ; entire population;
    BASE     ( filespec )
    PAUSE    ( [ON : OFF] )
    REPORT   ( filespec )
    START    ( integer )
    STEP     ( integer )
    STOP     ( integer )
    SWITCH   ( [ON : OFF] )
    TRACE    ( filespec )
    VAR      ( symbol ) ; mean population F;
)

; LITTER() - determines litter size for bred females;
LITTER   ( AGESEX   ( filespec ) ; females in age class with litters;
    BASE     ( filespec )
    PAUSE    ( [ON : OFF] )
    ;*
    RATES    ( filespec [,filespec ... ,filespec]); required;
    REPORT   ( filespec )
    START    ( integer )
    STEP     ( integer )
    STOP     ( integer )
    SWITCH   ( [ON : OFF] )
    TRACE    ( filespec )
    VAR      ( symbol ) ; stores number of litters;

; MATE() - selects mates for bred females and stores mate's alleles:
MATE
  AGESEX (filespec)
  CASES (filespec)
  DEPRESS (real : symbol); inbreeding depression factor;
  PAUSE ([ON : OFF])
  RATES (filespec); required;
  REPORT (filespec)
  START (integer)
  STEP (integer)
  STOP (integer)
  SWITCH ([ON : OFF])
  TRACE (filespec)
  VAR (symbol); stores number of unique mates;
)

; PACK() - condenses animal id's into contiguous list:
PACK
  AGESEX (filespec)
  CASES (filespec)
  PAUSE ([ON : OFF])
  START (integer)
  STEP (integer)
  STOP (integer)
  SWITCH ([ON : OFF])
  TRACE (filespec)
  VAR (symbol); stores population count;
)

; PURGE() - removes population from memory:
PURGE
  AGESEX (filespec); purged animals only
  CASES (filespec)
  PAUSE ([ON : OFF])
  START (integer)
  STEP (integer)
  STOP (integer)
  SWITCH ([ON : OFF])
  TAG (number); if not 0, removes only tagged pop
  TRACE (filespec)
  VAR (symbol); stores population count;
)

; READ() - reads population file and adds animals to population
; READ
  AGESEX (filespec)
  CASES (filespec)
  INPUT (filespec); required
  PAUSE ([ON : OFF])
  START (integer)
  STEP (integer)
STOP       ( integer )
SWITCH     ( [ON : OFF] )
TAG        ( number ) ; if non-zero, is new population tag ;
TRACE      ( filespec )
VAR        ( symbol ) ; stores number of animals added ;

; RESEED() - reseeds random number generator using system clock ;
RESEED ( 
PAUSE      ( [ON : OFF] )
SEED       ( integer ) ; 0 = use system clock ;
START      ( integer )
STEP       ( integer )
STOP       ( integer )
SWITCH     ( [ON : OFF] )
TRACE      ( filespec )
VAR        ( symbol ) ; stores new seed ;

; SET() - GAPPS mathematical expression analyzer ;
SET ( symbol = expression )
Legal operators: (+,-,*,/,^,%,=)
ABS        ( expression ) ; absolute value ;
CEIL       ( expression ) ; round up ;
EXP        ( expression ) ; exponent base e ;
FLOOR      ( expression ) ; round down ;
LN         ( expression ) ; log base e ;
LOG        ( expression ) ; log base 10 ;
MAX        ( expression, expression ) ; minimum of two expressions ;
MIN        ( expression, expression ) ; maximum of two expressions ;
MM         ( indvar, min, max, p50, p95)
NRAND      ( mean, cv) ; N-distributed random number ;
ROUND      ( expression ) ; round to nearest whole number ;
SQRRT      ( expression ) ; square root ;
URAND      ( a, b) ; uniform random number a ≤ n ≤ b ;

; SETUP() - initializes GAPPS runtime parameters ;
SETUP ( 
TEST       ( [ON : OFF] ) ; if ON, no runtime performed ;
TITLE      ( "literal string") ; run title ;
)

; SUMMARY() - displays requested variable values ;
SUMMARY ( 
PAUSE      ( [ON : OFF] )
REPORT     ( filespec )
START      ( integer )
STEP ( integer )
STOP ( integer )
SWITCH ( [ON : OFF] )
TRACE ( filespec )

; *;
SYMBOLS ( symbol [, symbol ..., symbol] ) ; up to 13 variables;
)

; SURVIVE() - selects and removes animals due to natural mortality;
SURVIVE( ; determines natural mortality;
AGEX ( filespec ) ; number died;
CASES ( filespec )
DEPRESS ( real : symbol ) ; inbreeding depression factor;
PAUSE ( [ON : OFF] )

; *;
RATES ( filespec ) ; required;
REPORT ( filespec )
START ( integer )
STEP ( integer )
STOP ( integer )
SWITCH ( [ON : OFF] )
TRACE ( filespec )
VAR ( symbol ) ; stores number of deceased animals;
)

; WEAN() - determines which young are weaned from their mother;
WEAN ( AGEX ( filespec ) ; number weaned;
CASES ( filespec )
PAUSE ( [ON : OFF] )

; *;
RATES ( filespec ) ; required;
REPORT ( filespec )
START ( integer )
STEP ( integer )
STOP ( integer )
SWITCH ( [ON : OFF] )
TRACE ( filespec )
VAR ( symbol ) ; stores number of weaned young;
)

; WRITE() - writes current population to a DBF disk file;
WRITE ( AGEX ( filespec )
CASES ( filespec )
MODE ( [APPEND : CREATE] ) ; CREATE deletes existing file;

; *;
OUTPUT ( filespec ) ; required;
PAUSE ( [ON : OFF] )
START ( integer )
STEP ( integer )
STOP ( integer )
SWITCH ( [ON : OFF] )
TAG (number); if not 0, write only if this tag;
TRACE (filespec)
VAR (symbol); number of animal records written;

The individual modules are described in detail in Chapter 8. Here, some basic concepts that underlie the Command file are summarized.

**Command file concepts**

A Command file consists of commands, subcommands, and specifications. Specifications may be numbers, strings, or file names, depending on the circumstance. Commands and subcommands are key words; GAPPs knows to look for them and knows what they mean. You are not allowed to use these keywords elsewhere in your modeling (to name a variable, for example). Most commands have associated subcommands; most subcommands have associated specifications.

Commands occur in blocks. At the head of each block must be the name of a module, such as ALLELES or MATE (the terminology "Command" and "module" are used interchangeably in this manual). Most commands have at least 1 subcommand that is required. In the case of modules that depend on a Rate file to instruct it, the RATES subcommand will always be required. In a few cases, all subcommands are optional, i.e. it is legal to specify only the command with no subsequent subcommands (these will always be modules that do not require RATE files).

Subcommands cannot be repeated within a command. Doing so will result in only the last of the identical subcommands being invoked.

**Example:**

```
BEGTN(
    YEAR (100)
) : Incorrect. The simulation
    will loop for 10, not 100 years.
```
IMPORTANT NOTE: In this, and following sections, the ":" symbol is used. It is for purposes of this manual only, and is not a part of CAPPS. It is used here merely to separate the sample command file from its explanation.

Sub-commands that are available within almost all of the modules include:

AGESEX [ ] - Specify the name for an AGSEX output file. The contents of the entries will vary with the particular module, but all will be in the same sex by age matrix format. See Chapter 9 for format.

CASES [ ] - Specify the name for a CASE output file. CASE files will contain different entries, depending on the particular module, but will always be in the same format. See Chapter 9 for format.

TRACE [ ] - Specify the name for a TRACE output file. TRACE files will contain summary information on the number of individuals affected by that particular module. See chapter 9 for format.

RATES [ ] - Specify the name of an appropriate RATE file. RATE files are required for modules that use them. See Chapter 6 on RATE files.

VAR [ ] - Specify the name for the variable containing the results of the module. This variable can then be displayed in the SUMMARY module, or used as an
independent variable in other modules.

**START [ ]** - Specify the year or replicate simulation to begin using this module. (The DEFAULT is 1.) This can be used to delay the execution of a particular module until a specified year within the simulation, or until a specified run. If omitted, the module will be executed beginning with year 1 and/or replication 1.

**STEP [ ]** - Specify the interval of years or runs for executing the module. (The DEFAULT is 1). This can be used to execute the module only every 2, 3, etc. years or runs. If omitted, the module will be executed during each model year and/or each replication.

**STOP [ ]** - Specify the year and/or replication to stop using this module. (The DEFAULT is the appropriate value of YEAR or RUN.) This can be used to discontinue execution of the module at a particular year or run. If omitted, the module will continue in use until the simulation ends.

**PAUSE [ON : OFF]** - When ON, subcommand PAUSE instructs GAPPS to display detailed information on each individual being processed, and to pause between each individual processed until a key is struck (we recommend the space-bar key). This way, screen information about that individual may be frozen for inspection, and the inner-workings of the process examined. This can be a useful educational tool, although will not generally be used during actual runs.
SWITCH [ON : OFF] - The SWITCH subcommand simply allows the user to quickly disable a command by changing the word "ON" to "OFF". Omitting the SWITCH subcommand is equivalent to having it set to ON. An alternative way of disabling a module is to use semi-colons to turn the entire block into a comment (c.f.). In experimental uses, where you may want to quickly run alternative models with and without a particular module, using SWITCH may be faster.

Note that there are alternative ways to use defaults, or to disable a subcommand. For example, specifying START 1 and STEP 1 is exactly equivalent to omitting them both. The START, STEP and STOP subcommands will likely be used only when using modules in less than all years and/or runs is contemplated. The output subcommands AGESEX, CASES, REPORT, and TRACE are always optional; they need be included only when output of this type is desired.

Some modules have other subcommands. For example, ALLELES has subcommands DIVERSITY and HETERO, and HUNT has the subcommand NUMBER. These are explained in the individual modules section.

Editing the Command file

After making backup copies, the Command file is yours to edit, almost at will. We suggest using a simple text or program editor, although word processors can be used as long as the output is in standard ASCII format. The following editing conventions and possibilities apply:

PARENTHESES

Each command and subcommand must be followed by an open parenthesis "(". This indicates to GAPPS that everything following will be a part of that (sub)command. Before going on
to the next (sub)command, a close parenthesis "\)" is required. This tells GAPPS to expect another (sub)command. Thus, parentheses delimit all information contained within a (sub)command. For a command, this is true whether or not subcommands are included with the command.

Example:

```
AGE

......

---

AGE ()

---

Example:

```
BEGIN

()  : Incorrect.

YEAR 100  : No parentheses enclosing

)  : the specified number of

: years

---

BEGIN

(  : Correct.

YEAR (100)

)

---

As long as the proper parentheses are included, the spacing of the subcommands, etc. is flexible. It does not matter whether you use a separate line for each new piece of information, or put them all on the same line, as long as each word, string, etc. is delimited by a parenthesis. The 3 examples below are all legal, and will all have exactly the same effect. Which way you prefer will be largely a matter of personal taste.
Examples:

BEGIN (                  : Correct.
    RUN               (10)
 )

BEGIN                   : Correct.
    (RUN             (10))

BEGIN (RUN (10) )     : Correct.

However, you must be certain to include only the correct number of parentheses.

Example:

BEGIN (                  : Incorrect.
    (RUN             (10))        : Too many parentheses.

BEGIN (                  : Incorrect.
    RUN             10)           : Too few parentheses.

SUBCOMMANDS

- Subcommands cannot be used in the absence of a Command. For example, simply typing RATES and specifying a Rate file will not have meaning unless preceded by a command.

Example:

RATES (BIRTH.DBF)       : Incorrect. No command.

BIRTHI(
    RATES (BIRTH.DBF) : Correct
 )

- Subcommands may be placed in any order within an
appropriate command.

Example:

```
BIRTH(
    RATES   (BIRTH.DBF) : Correct
    SWITCH   (ON)
    VAR      (BABIES)
    CASES   (BIRTHS.CAS)
)
```

```
e tc.
```

Example:

```
BIRTH(
    SWITCH   (ON)
    CASES   (BIRTHS.CAS) : Correct, and identical to above example in function.
    RATES   (BIRTH.DBF)
    VAR      (BABIES)
)
```

Example:

```
BIRTH (SWITCH(ON) CASES (BIRTHS.CAS) RATES (BIRTH.DBF)
         VAR (BABIES))
```

```
e tc. : Also Correct, and identical to above example in function.
```
COMMENTS

- Comments are allowed anywhere in the command file. Comments must be enclosed by semi-colons (;). Comments may be useful in reminding you what is going on.

Example:

```
BEGIN (RUN (20))
Begin the simulation year : Incorrect, no semi-colons
BEGIN (YEAR (100))
SURVIVE ( etc.
```

```
BEGIN (RUN (20))
;Begin the simulation year  : Incorrect, only 1 semi-
BEGIN (YEAR (100))      : colon. Remainder will
SURVIVE ( etc.         : be interpreted as comment.
```

```
BEGIN (RUN (20))
;Begin the simulation year; : Correct
BEGIN (YEAR (100))
SURVIVE ( etc.
```

COMMAND ORDER

- The modules may be placed in any order. However, note that some orderings will make no sense, or may result in the model not running at all. Some orderings will make sense, but will not necessarily result in the model you want. Take time to think out the consequences of the order you specify.

Example:

```
BEGIN (RUN (5))
END ()
SURVIVE ( : Incorrect: Ended run loop
   RATES (MORT.DBF) : before doing anything
BEGIN (YEAR (50))
END ()
```
BEGIN (RUN (5))
BEGIN (YEAR (50))
SURVIVE (RATES (MORT.DBF))
END ()
END ()

Example:
BEGIN (YEAR (8))
AGE ()
SURVIVE (RATES (MORT.DBF))
END ()

Example:
BEGIN (YEAR (8))
SURVIVE (RATES (MORT.DBF))
ACE ()
END ()

Example:
BEGIN (YEAR (10))
SURVIVE (RATES (MORT1.DBF))
LITTER (RATES (LITTER1.DBF))
AGE ()
BIRTH (RATES (GENDER.DBF))
BREED (RATES (BREED.DBF))
MATE (RATES (FATHER.DBF))
WEAN (RATES (WEAN.DBF))
END ()

: Correct
: (But note, that w/ mortality
: only, pop'n will go extinct!)
: Ends Years
: Ends Runs

: Not incorrect, but animals
: will always be 1 year older
: than expected when subjected
: to mortality

: Correct, animals
: are subjected to mortality,
: then have their ages
: incremented.

: Correct, implies that
: females breeding during year
: x give birth during year x+1
: and that females may die
: between being bred and
: giving birth. Probably
: appropriate for species with
: long gestation, where period
: of high mortality comes
: between breeding and giving
: birth.
Example:

```
BEGIN (YEAR (10))
SURVIVE (  
  RATES (MORT1.DBF))
BREED (  
  RATES (BREED.DBF))
MATE (  
  RATES (FATHER.DBF))
LITTER (  
  RATES (LITTER1.DBF))
AGE ()
BIRTH (  
  RATES (GENDER.DBF))
WEAN (  
  RATES (WEAN.DBF))
END ()
```

: Correct, implies that females breeding during year \( x \) give birth during year \( x \) and that no females die between being bred and giving birth. Probably appropriate for species with short gestation, where no significant mortality occurs between breeding and birth.

: Note: in both examples, AGE precedes birth, otherwise, newborns immediately become 1 year-olds.

**REPETITION**

Commands may be repeated within a loop, and will be executed each time they are encountered. Subcommands cannot be repeated within a single command.

Example:

```
SURVIVE (  
  RATES (WINTER.DBF))
CENSUS (  
  VAR (CNSUS1))
SURVIVE ( RATES  
  (SPRING.DBF))
CENSUS (VAR (CNSUS2))
e tc.
```

: Correct. This section of the command file calls for 2 events with different rates, and 2 censuses, 1 after each mortality event. Survival rates in WINTER.DBF and SPRING.DBF are seasonal survival rates.
Example:

```
SET (  
  (PROP = POP * 0.10)  
)
```

: SET creates a variable using POP in this case

```
HUNT (  
  RATES    (HUNTBOOTH.DBF)  
  NUMBER   (PROP)  
  NUMBER   (25)  
  AGESEX   HUNT.DAT  
  SWITCH   (ON)  
)
```

: Incorrect; each Command may have only 1 of each of the possible subcommands. This is asking the hunt to take both a fixed number and a proportion of the population

```
HUNT (  
  RATES    (HUNTS.DBF)  
  NUMBER   (25)  
  SWITCH   (ON)  
  AGESEX   (HUNTALL.DAT)  
)
```

: Correct; 2 separate hunts, 2 separate Rate files. The first hunts 25 during the spring, the second hunts 10% during the fall. Different age/sex specific rates apply for the 2 hunts. Note that the results of both hunts may be written to the same file; they will appear in separate records however.
CHAPTER 5

THE POPULATION FILE

The population file in GAPPS contains all the information known about each individual. We call each piece of information a "trait".

The use of traits in GAPPS

Each individual has a total of 31 traits. They are listed, together with the field they occupy in an animal's record, and the number of columns occupied, in Table 5.1. One trait that always accompanies each individual is its identification number. ID numbers are used internally in the program, but are otherwise unimportant.

Four traits can be used to differentiate types of animals for purposes of structuring rate files: (1) age, (2) sex, (3) with offspring or not, and (4) with parents or not. The last 3 of these are essentially "yes-no" kinds of data. Each animal is either female or not, with offspring or not, and with offspring or not. The last two can be considered descriptions of the animal's family status. When first born, an animal will be with its parents (usually just the mother). After it weans, it will leave its parents, but may not yet have offspring. If female, it may later have offspring. Taking all possible combinations of sex, with offspring and with parents would result in $2^3 = 8$ categories, each of which has a possible 30 age-classes, for a total of 240 "types" of animals. However, GAPPS does not recognize the possibility of both being with parents and with offspring. Thus, there are a maximum of 6 categories x 30 age-classes = 180 different types of animals for purposes of rate files. For example, one can define a death rate for only 7 year-old females that are accompanied by offspring.

An additional trait, F (the inbreeding coefficient, trait number 9), can be used in 3 of the event modules to modify rates
(BREED, MATE, and SURVIVE). It is used if any of these 3 events are to be influenced by inbreeding depression. (See the sections of these modules).

The remaining traits are used by GAPPS internally, or may be of interest to the user as summary information. They are not capable of delineating other "types" of animals. None of these traits refers to a "yes-no" kind of question, rather they all take values that will be different for each individual. For example, one cannot define a specific death rate for only females that have 5, as opposed to 4 young, or for males who have mated with female number 278.

Each individual in GAPPS also has a status code (trait number 4). Status codes refer to the breeding-birthing sequence. There are 5 status codes. They are listed in Table 5.2. GAPPS needs to know the status code of any individual that is processed during the BREED, MATE, LITTER, BIRTH, or WEAN events. A female that is AVAILABLE is a candidate for breeding. A female that is BRED is a candidate for mating. A female that is MATED can have her litter size determined by LITTER. A female that is LITTERED can give birth when BIRTH is called. A female that is BIRTHED is a candidate to have her offspring weaned when WEAN is called.

GAPPS needs to know the identity of each individual's mother, father, mate, and children in order to perform its functions correctly. For example, in order to calculate each individual's inbreeding coefficient, its mother and father must be known. As another example, suppose that a female with offspring dies. Her offspring are no longer "with parent", but are now alone. GAPPS must know who her offspring are, so that their traits may be changed appropriately.

All individuals in GAPPS also possess a hypothetical diploid locus, at which 2 alleles are always present. Alleles in GAPPS simply take integer values. When mating takes place, the female who will give birth must also store her mate's alleles, so that her offspring may receive their alleles through simple Mendelian inheritance. The mate's alleles must be stored with the female.
because it is possible for him to die before the offspring are born, and if his alleles were stored only in him, they would be lost. Associated with each allele is a population tag, which can be used to trace the origin of animals and alleles; 0 will usually stand for the "base" population, but other tags may be used to indicate immigrants.

Creating and editing population files

GAPPS cannot perform a population projection through time without a population file. Therefore, you must create a population file for GAPPS to "work on". One can think of this as the "initial population". There are 2 ways to create a population file: (1) "from scratch", using dBASEII to create new records for each animal, and (2) having GAPPS make a new file from one of the pre-supplied "generic" population files.

To create a population file from scratch, use the dBASEII structure supplied with your disk, called POPSTR.DBF. It contains no members, but is in the form GAPPS requires. Then use the APPEND command to fill in the traits of each individual you want to create. Many of the entries for traits will be arbitrary. For example, you may want to start assigning ID numbers with 1, or with 101. You may want to start assigning allele numbers with 1 or with 57. It will make little difference. However, for some of the entries, it will be important that you remain consistent. For example, if individual # 1 is a female who has had an offspring with ID # 5, it is important that you give assign individual # 5 a mother's ID of 1. If individuals in your population are to be mated, it is important that you assign the mate fields of both animals correctly. GAPPS will run if you make errors in your assignations, but it will probably not run the way you want it to. Making sure that all traits are consistent throughout your population will be a laborious job. Most often, you will want to create populations this way only for specific applications where...
precise control over traits is necessary, and most often only for very small populations.

An alternative is to create a population file with GAPPS, using a pre-supplied population file. Create a Command file that contains the events and rate files which you believe characterize your population, and specify a pre-supplied population file in the INPUT subcommand within the READ command. Make sure you include the proper reproduction modules (BREED, MATE, LITTER, BIRTH, and WEAN), or no new animals can be made. Include a WRITE command in your command file, and specify the new population's name with a .DBF extension in the OUTPUT subcommand. When the simulation ends, you will have a new population, which will have a structure dictated by the events and rates you specified.

How long should you run this simulation to create your new population file? It depends how long it will take for the structure of the old (pre-supplied) population to disappear completely in favor of your new population structure. Generally, this will be longer for animals that live a long time, and shorter for animals that die and replace themselves quickly. It is usually not sufficient simply to run it long enough to kill off the original animals because some dynamics have lag-times that will outlast your original population. It's best to run at least a few generations.

After creating your new population with WRITE, you may still edit it with dBASEII. You may append some new members, delete some, or change that traits of some members. However, if you do, remember that you are under the same constraints as if you were creating one "from scratch". If mates do not agree, or if parents are not associated with the correct children and vice versa, GAPPS will run, but you may not get the results you desire.

Three population files are supplied with your copy of GAPPS. One is called GJPOP.DBF and is the initial population used in the example in Chapter 2, "Overview: a quick sample session". The others are called POPR.DBF and POPK.DBF, and represent generic
population structures that might represent mammal species with r and K evolutionary strategies, respectively. Any of these 3 may be used as starting points to create population structures that reflect a species you're working with.
<table>
<thead>
<tr>
<th>Trait</th>
<th>Name</th>
<th>Columns</th>
<th>Decimal places</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ID</td>
<td>5</td>
<td>0</td>
<td>Individual's ID number</td>
</tr>
<tr>
<td>2</td>
<td>SEX</td>
<td>1</td>
<td>0</td>
<td>0 = male; 1 = female</td>
</tr>
<tr>
<td>3</td>
<td>AGE</td>
<td>2</td>
<td>0</td>
<td>0-29; 0 = young-of-year</td>
</tr>
<tr>
<td>4</td>
<td>STATUS</td>
<td>1</td>
<td>0</td>
<td>For breeding, etc.</td>
</tr>
<tr>
<td>5</td>
<td>PARENTS</td>
<td>1</td>
<td>0</td>
<td>0 = alone; 1 = w/ parents</td>
</tr>
<tr>
<td>6</td>
<td>OFFSPRING</td>
<td>1</td>
<td>0</td>
<td>Number of offspring</td>
</tr>
<tr>
<td>7</td>
<td>MOTHER</td>
<td>5</td>
<td>0</td>
<td>Mother's ID number</td>
</tr>
<tr>
<td>8</td>
<td>FATHER</td>
<td>5</td>
<td>0</td>
<td>Father's ID number</td>
</tr>
<tr>
<td>9</td>
<td>F</td>
<td>6</td>
<td>4</td>
<td>Inbreeding coefficient</td>
</tr>
<tr>
<td>10</td>
<td>TAG1</td>
<td>3</td>
<td>0</td>
<td>Population tag: allele 1</td>
</tr>
<tr>
<td>11</td>
<td>TAG2</td>
<td>3</td>
<td>0</td>
<td>Population tag: allele 2</td>
</tr>
<tr>
<td>12</td>
<td>ALLELE1</td>
<td>5</td>
<td>0</td>
<td>Allele from father</td>
</tr>
<tr>
<td>13</td>
<td>ALLELE2</td>
<td>5</td>
<td>0</td>
<td>Allele from mother</td>
</tr>
<tr>
<td>14</td>
<td>MATE</td>
<td>5</td>
<td>0</td>
<td>Mate's ID number</td>
</tr>
<tr>
<td>15</td>
<td>MTAG1</td>
<td>3</td>
<td>0</td>
<td>Pop'n tag: mate's allele1</td>
</tr>
<tr>
<td>16</td>
<td>MTAG2</td>
<td>3</td>
<td>0</td>
<td>Pop'n tag: mate's allele2</td>
</tr>
<tr>
<td>17</td>
<td>MALLELE1</td>
<td>5</td>
<td>0</td>
<td>Mate's allele from father</td>
</tr>
<tr>
<td>18</td>
<td>MALLELE2</td>
<td>5</td>
<td>0</td>
<td>Mate's allele from mother</td>
</tr>
<tr>
<td>19</td>
<td>LITTERSIZE</td>
<td>2</td>
<td>0</td>
<td>Litter size</td>
</tr>
<tr>
<td>20</td>
<td>CHILD01</td>
<td>5</td>
<td>0</td>
<td>First child's ID number</td>
</tr>
<tr>
<td>21</td>
<td>CHILD02</td>
<td>5</td>
<td>0</td>
<td>Second child's ID number</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>31</td>
<td>CHILD12</td>
<td>5</td>
<td>0</td>
<td>Twelfth child's ID number</td>
</tr>
</tbody>
</table>

Table 5.1. The population file, showing the 31 traits of each individual. The order, name, columns, and number of decimal places are as in the dBASEII format. The remarks are not part of the population file in dBASEII, but are shown here for clarity.

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>AVAILABLE</td>
<td>Individual is not with parents or offspring, may be bred.</td>
</tr>
<tr>
<td>1</td>
<td>BRED</td>
<td>Individual has bred, may now mate.</td>
</tr>
<tr>
<td>2</td>
<td>MATED</td>
<td>A mate has been selected.</td>
</tr>
<tr>
<td>3</td>
<td>LITTERED</td>
<td>A litter size has been determined.</td>
</tr>
<tr>
<td>4</td>
<td>BIRTHED</td>
<td>Individual has given birth, is with offspring.</td>
</tr>
</tbody>
</table>

Table 5.2. Breeding status codes for females.
RATE FILES

Rate files contain the information necessary to govern how animals die, breed, give birth, etc. A Rate file must be in existence for every called event module which requires a Rate file. Eight modules require Rate files: BIRTH, BREED, HUNT, IMMIGRATE, LITTER, MATE, SURVIVE, and WEAN.

Types

There are 3 types of rate files, depending on which traits can be used to differentiate members of the population:

Type "A": AGE is used to differentiate animals. Since there are 30 age-classes, a total of 30 different rates are possible, and all "A" type rate files expect 30 records.

"A" type modules:
BIRTH (all are assumed to be females)
BREED (all are assumed to be females)
LITTER (all are assumed to be females)
MATE (all are assumed to be males)
WEAN (all are assumed to be females)

Type "SA": SEX and AGE are used to differentiate animals. Since there are 30 age-classes and 2 sexes, a total of 60 different rates are possible, and the "SA" type rate file expects 60 records.

"SA" type module:
IMMIGRATE (all are assumed to be without offspring and without parents).

Type "POSA": With or without PARENTS, with or without OFFSPRING, SEX, and AGE are used to differentiate animals. There are a total of 180 different rates possible ([2 * 2 * 2 * 30] -
for unrecognized category "simultaneously with parents and offspring", and both "POSA" type rate files expect 180 records.

"POSA" type modules:
HUNT
SURVIVE

NOTE: Although separate sections exist for males with and without offspring, there is no internal mechanism in the current implementation of GAPPS to distinguish males with and without offspring. The sections are maintained in the 2 "POSA" rate files for reasons of symmetry; however, all males will be "without offspring". For this reason, entries in the "Males With Offspring" section are irrelevant, and may be left blank.

Format

All rate files have columns allocated to the following information:

FUNCTION (mathematical function defining the rates)
INDVAR (independent variable for function, if applicable)
PARM1 (4 parameters which can be used to govern the rates. See next section).
PARM2
PARM3
PARM4

"A" type rate files will have these columns preceded by a column for the AGE of the animal (Table 6.1). An IMMIGRANT file (type "SA") will have columns for SEX and AGE (Table 6.2). The 2 "POSA" files will have columns devoted to OFFSPRING, PARENTS, SEX and AGE (Table 6.3).
### Table 6.1

An "A" type rate file, (used for BREED, MATE, BIRTH, LITTER, and WEAN). There are 30 records, 1 for each age from 0 to 29, and for each a function must be specified, along with from 1 to 4 parameters, and an independent variable (INDVAR), if applicable.

<table>
<thead>
<tr>
<th>AGE</th>
<th>FUNCTION</th>
<th>INDVAR</th>
<th>PARM1</th>
<th>PARM2</th>
<th>PARM3</th>
<th>PARM4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CON</td>
<td></td>
<td>0.500</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>CON</td>
<td></td>
<td>0.500</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>2</td>
<td>SL</td>
<td>POP</td>
<td>0.100</td>
<td>0.200</td>
<td>0.000</td>
<td>1.000</td>
</tr>
<tr>
<td>29</td>
<td>SL</td>
<td>POP</td>
<td>0.100</td>
<td>0.200</td>
<td>0.000</td>
<td>1.000</td>
</tr>
</tbody>
</table>

### Table 6.2

An "SA" type rate file, (used for IMMIGRATE). There are a total of 60 records, 30 for each sex. From 1 to 4 parameters may be specified, and an independent variable (INDVAR) is necessary for MM or SL functions.

<table>
<thead>
<tr>
<th>SEX</th>
<th>AGE</th>
<th>FUNCTION</th>
<th>INDVAR</th>
<th>PARM1</th>
<th>PARM2</th>
<th>PARM3</th>
<th>PARM4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>CON</td>
<td></td>
<td>0.010</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>CON</td>
<td></td>
<td>0.020</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>CON</td>
<td></td>
<td>0.500</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>CON</td>
<td></td>
<td>0.010</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>CON</td>
<td></td>
<td>0.010</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>MM</td>
<td>POPK</td>
<td>0.000</td>
<td>0.500</td>
<td>0.850</td>
<td>1.950</td>
</tr>
</tbody>
</table>
Table 6.3. A "POSA" type rate file (used for SURVIVE and HUNT). "P" stands for 'with parents or not', with 1 = with parents and 0 = independent from parents. "O" stands for 'with offspring or not', with 1 = accompanied by offspring and 0 = unaccompanied by offspring. There are a total of 180 records, with blocks for animals both with parents and offspring omitted. From 1 to 4 parameters may be specified, with an independent variable (INDVAR) required for an SL or MM function.

<table>
<thead>
<tr>
<th>P</th>
<th>O</th>
<th>SEX</th>
<th>AGE</th>
<th>FUNCTION</th>
<th>INDVAR</th>
<th>PARM1</th>
<th>PARM2</th>
<th>PARM3</th>
<th>PARM4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>CON</td>
<td></td>
<td>0.010</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>CON</td>
<td></td>
<td>0.020</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>CON</td>
<td></td>
<td>0.500</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>CON</td>
<td></td>
<td>0.010</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>CON</td>
<td></td>
<td>0.010</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>MM</td>
<td>POPK</td>
<td>0.000</td>
<td>0.500</td>
<td>0.850</td>
<td>1.950</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>CON</td>
<td></td>
<td>0.005</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>CON</td>
<td></td>
<td>0.010</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>CON</td>
<td></td>
<td>0.005</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>CON</td>
<td></td>
<td>0.010</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>

etc.

Table 6.4. The 3 mathematical functions currently recognized by rate files in GAPPS.

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Name</th>
<th>Number of Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>CON</td>
<td>Constant</td>
<td>1</td>
</tr>
<tr>
<td>SL</td>
<td>Straight line</td>
<td>2 (4)</td>
</tr>
<tr>
<td>MM</td>
<td>Modified Michaelis-Mention</td>
<td>4</td>
</tr>
</tbody>
</table>

---
Defining Rates

The current implementation of CAPPS allows for 3 mathematical functions defining rates. These are summarized in Table 6.4.

One of these 3 abbreviations (CON, SL, or MM) should appear in the FUNCTION column of all rate files.

Function CON (Constant)

The simplest function is the constant. The parameter value in the PARM1 column is taken as the constant rate; the values in PARM2, PARM3 and PARM4 are irrelevant when CON is the specified function. The independent variable (INDVAR) is also irrelevant; it does not matter if this column is filled or is left blank when CON is the specified function.

Example:

<table>
<thead>
<tr>
<th>AG</th>
<th>FUNCTION</th>
<th>INDVAR</th>
<th>PARM1</th>
<th>PARM2</th>
<th>PARM3</th>
<th>PARM4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CON</td>
<td></td>
<td>0.500</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>CON</td>
<td></td>
<td>0.500</td>
<td>2.000</td>
<td>0.340</td>
<td>1.234</td>
</tr>
<tr>
<td>2</td>
<td>CON</td>
<td>TEAINCHINA</td>
<td>0.500</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>

The specified rate in the above example is 0.5 for ages 0, 1 and 2. Because the specified function in a constant (CON), only the first parameter (PARM1) is examined; the other 3 are irrelevant. No independent variable (INDVAR) is necessary to define a constant rate, thus the variable TEAINCHINA is also ignored.

Function SL (Straight line)

The straight line function (SL) uses the formula for a straight line:
\[ y = a + bx \]

to define the desired rate, where \( y \) is the resulting rate, \( x \) is the independent variable, \( a \) is the intercept term, and \( b \) is the slope. The user also specifies a minimum and maximum value the resulting rate may attain. PARM1 and PARM2 are used for these upper and lower bounds. The slope \( b \), is placed in the PARM3 column, and \( a \), the intercept slope term is placed in the PARM4 column. The independent variable must be a legally named variable from within GAPPS. For example, you might use POP, if POP was legally named in a CENSUS event in your Command file.

Example:

<table>
<thead>
<tr>
<th>AG</th>
<th>FUNCTION</th>
<th>INDVAR</th>
<th>PARM1</th>
<th>PARM2</th>
<th>PARM3</th>
<th>PARM4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SL</td>
<td>POP</td>
<td>0.000</td>
<td>0.800</td>
<td>0.000</td>
<td>0.450</td>
</tr>
<tr>
<td>1</td>
<td>SL</td>
<td>POP</td>
<td>0.000</td>
<td>1.000</td>
<td>0.100</td>
<td>0.450</td>
</tr>
<tr>
<td>2</td>
<td>SL</td>
<td>POP</td>
<td>0.200</td>
<td>1.000</td>
<td>0.100</td>
<td>0.650</td>
</tr>
</tbody>
</table>

In the above example, rates for ages 0-2 are linear functions of the variable POP. For age 0, the probability will take the value:

\[ \text{Probability} = 0 + 0.45(\text{POP}) \]

but will never exceed 0.80. For age 1, the probability will take the value:

\[ \text{Probability} = 0.1 + 0.45(\text{POP}) \]

and for age 2, the probability will take the value:

\[ \text{Probability} = 0.1 + 0.65(\text{POP}) \]

but will never be less than 0.2, even if POP = 0.

**Function MM (Modified Michaelis-Menton)**

The MM equation is very useful for simulating non-linear
functions, especially sigmoid-shaped relationships. The equation takes the form:

\[ \text{Min} + (\text{Max} - \text{Min}) \times \left(1 - \frac{V^x}{(0.05P_{95}^x + V^x)}\right) \]

where, Min = the minimum value the function is permitted, Max = the maximum value the function is permitted, V = an independent variable, x = an exponent controlling the shape of the function, and P = the value of V at which the function takes a value 95% of the distance between Min and Max.

The independent variable, V, will generally be some characteristic of the population that rates will depend on. Examples include population size, or population scaled to carrying capacity (N/K). The minimum and maximum values may be constrained logically (e.g., probabilities of bounded by 0.0 and 1.0), or may be further restrained, (e.g., if even under the best possible conditions, survival is no better than 90%, then Max should be set to 0.90). We have found that rather than try to estimate the shape parameter x, an alternative parameter may be defined, namely: \(P_{50}\), which we define as the value of V at which the function takes a value midway between Min and Max. Thus, the 4 parameters to be specified are:

1. Min
2. Max
3. \(P_{50}\)
4. \(P_{95}\). Fig. 6.1 illustrates a typical sigmoid curve produced using the MM function. The independent variable V in this case is the population (N) relative to its carrying capacity (K), or N/K. To create a function of this shape, the user specifies Min and Max, then estimates the values of the N/K at which the function takes the value \(\text{Max} - \text{Min}) \times 0.50 (= P_{50})\), and \(\text{Max} - \text{Min}) \times 0.05 (= P_{95})\).

In Rate files with MM specified as the function, an independent variable must also be specified. It must be a legal variable, previously defined from a VAR or SET command. PARM1 takes the minimum value, PARM2 takes the maximum value. PARM3
takes the P50 value, while PARM4 take the P95 value.

Example:

<table>
<thead>
<tr>
<th>AG</th>
<th>FUNCTION</th>
<th>INDVAR</th>
<th>PARM1</th>
<th>PARM2</th>
<th>PARM3</th>
<th>PARM4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>MM</td>
<td>NK</td>
<td>0.000</td>
<td>0.800</td>
<td>1.000</td>
<td>1.450</td>
</tr>
<tr>
<td>1</td>
<td>MM</td>
<td>NK</td>
<td>0.000</td>
<td>1.000</td>
<td>1.100</td>
<td>2.000</td>
</tr>
<tr>
<td>2</td>
<td>MM</td>
<td>NK</td>
<td>0.200</td>
<td>1.000</td>
<td>0.960</td>
<td>2.200</td>
</tr>
</tbody>
</table>

In the above example, the probability for age 0 will take minimum and maximum values of 0.0 and 0.8, respectively. At NK = 1.00, it take the value 0.4, and NK will be 1.45 when it takes the value 0.04. The probability for age 1 will take minimum and maximum values of 0.0 and 1.0, respectively. At NK = 1.10, it take the value 0.5, and NK will be 2.00 when it takes the value 0.05. The probability for age 2 will take minimum and maximum values of 0.2 and 1.0, respectively. At NK = 0.96, it take the value 0.6, and NK will be 2.20 when it takes the value 0.24.
Fig 6.1. An illustrative example of a function using the MM equation. The independent variable is named $N/K$ in this example (in GAPPS, the "/" is not a legal part of a variable name; it would have to be named NK or N_K). The user specifies the 4 quantities: Min, Max, P50 (point 'a') and P95 (point 'b').
HOW TO RUN GAPPS

Assuming that:

1) You have a Command file satisfactorily edited,
2) A population file exists and the Command file calls for it to be read,
3) All the rate files (all with .DBF extensions) files that are called by the Command file exist on your disk, and,
4) Your computer is on, and you have accessed the directory in which all these files live,

you are ready to run GAPPS. Perform your simulations simply by typing:

GAPPS [Command file name] -p

The "-p" tells GAPPS to pause when examining the Command file when it finds errors. For example, if you've created a Command file named TEST1, run it by typing:

GAPPS TEST1 -p

If this doesn't work, check very carefully that the 4 requirements listed above have been met. If all are met and it still doesn't work, there may be a hardware or software incompatibility. Check the section on hardware and software requirements.

The first thing GAPPS does is check your Command file for matching parentheses, semi-colons and quotation marks. If it finds an odd number of any of these, it will stop and tell you on which line of your command file the error lies. The second thing GAPPS does is check your command file for proper syntax, defined variables, defined commands and subcommands, etc by scanning
through your command file. You will see the command file scroll by while GAPPS checks it. It will also check to see that each file you've called for exists on disk. It will beep and display an error message if there are errors here. Thirdly, GAPPS goes through an optimization routine, where preliminary calculations (such as those that will be repeated often during the program) are completed. Finally, the simulation begins running, and you will see whatever screen display you have requested using the SUMMARY command. Any files you have requested will be made during the run.

If at any time you wish to end the program, use a Ctrl-C (control-C). You may have to push it a few times. The program will stop, but any files made by the program will remain, and will be complete up until the time the Ctrl-C was hit. We recommend avoiding turning your computer off while GAPPS is running.

Included on your disk is a small batch file called DO. It can be used to run GAPPS, by simply typing DO and the name of your Command file, e.g.:

DO TEST1.
MODULES IN GAPPS

Modules can be divided into those that change the status of individuals in the population (Event Modules) and those that perform some structural chore that is essential to the proper running of your model, but do not change the status of individuals within the population (Root Modules). Event Modules include those that require rate files and those that do not. In this chapter, each Module will be described, and the options for its use detailed. It may be useful to refer to the sample Command file, INPUT, while reading this chapter.

The Root modules are:

BEGIN (), DUMP (), END (), RESEED (), SET (), SETUP(), and SUMMARY ().

The event modules not requiring rate files are:

AGE (), ALLELES (), CENSUS (), GENETICS (), INBREED (), PACK (), and PURGE ()

The event modules for which a rate file is required are:

BIRTH (), BREED (), HUNT (), IMMIGRATE (), LITTER (), MATE (), READ (), SURVIVE (), WEAN (), and WRITE ()

In this chapter, portions of command files are given as examples. They are generally not complete command files. Also, as before, the ":" is used to separate the sample command file from the explanation. Do not use a ":" in your command file; this is for purposes of this manual only.

ROOT MODULES

BEGIN ()

The BEGIN command marks the beginning of 1 of the 2 kinds of
loops possible within GAPPS. To begin a replicate simulation, the subcommand RUN must be used, with its specification indicating the number of replicate simulations to be performed. To begin a year within the simulation, the subcommand YEAR must be used, with its specification indicating the number of years to be run within each replicate simulation. A separate BEGIN command must be used for each loop.

Example:

```
BEGIN (
    RUN (10)
    YEAR (100)
)
```

: Incorrect. Both subcommands are used with one command.

Example:

```
BEGIN ( RUN (10) )
BEGIN ( YEAR (100) )
```

: Correct. Program will run 10 replicate simulations, each of 100 years duration.

For each BEGIN command there must be a corresponding END () command (see next section). The BEGIN () and END () syntax is similar to the FOR - NEXT syntax in BASIC and the DO - CONTINUE syntax in FORTRAN.

The only other subcommands available within BEGIN are PAUSE and TRACE.

Example:

```
BEGIN ( RUN (5)
    PAUSE (ON )
    BEGIN (YEAR (100))
```

: Correct. Machine will pause before each of the 5 loops until user strikes a key.

END ()

The END command is the simplest of all. It merely marks the
end of a loop specified by a BEGIN command. There must be a
BEGIN command (using either RUN or YEAR) associated with each use
of END (). END is mandatory for each use of BEGIN.

Example:

BEGIN (RUN 10))  : Incorrect. Only 1 END
BEGIN (YEAR (20)) : command for 2 BEGIN
END ()  : commands.

Example:

BEGIN (RUN 10))  : Correct.
BEGIN (YEAR (20)) :
END ()  :

RESEED ()

At the heart of the operation of GAPPS is a uniform (0,1)
random number generator. In the absence of the RESEED module,
the random number generator is always initiated at the same
place. This way, you can duplicate a run exactly by simply not
changing any commands. The RESEED module sets the random number
generator according to a specified seed number, or according to
the clock that resides in the DOS system of your computer.

The RESEED module may be invoked at any time, but it's most
common use will be at the beginning of a program. If RESEED is
called with an identical seed between loops of a simulation, all
the simulations will be identical.

The user can control the seed used to begin the random
number generation with the SEED subcommand. Any integer up to
65,535 is a legal specification. If SEED is omitted, or if a
zero (0) is entered, RESEED uses the clock of your computer's DOS
system to reseed the random number generator.
Example:

RESEED (  
  SEED (0)  
)  
BEGIN ( RUN (50) )  
.  
.  
END ()

: Correct. Will produce different set of results for each set of runs, because system clock will always be different for each set. Not recommended for debugging.

Example:

RESEED (  
  SEED (38415)  
)  
BEGIN ( RUN (50) )  
END ()

: Correct. Will produce exact same results for entire set of runs, no matter when run, because seed is set by the integer.

Examples:

BEGIN ( RUN (50) )  
RESEED ()  
BEGIN ( YEAR (25) )  
.  
.  
END ()

END ()

: Correct, all 50 runs will be different because system clock will be different. NOTE: This can be dangerous if runs are very short.

BEGIN ( RUN (50) )  
RESEED (9278)  
BEGIN ( YEAR (25) )  
.  
.  
END ()

END ()

: Correct, but wasteful, because all 50 runs will be identical to each other. The random numbers used in all 50 will occur in the same order, thus no new information will be gained.

BEGIN ( RUN (50) )
BEGIN ( YEAR (25) )
END ()

: Correct. The RESEED command is omitted entirely because it is not needed.
SET ()

The SET module is one of the most useful aspects of GAPPs. SET interprets and executes mathematical expressions, and creates variables with the proper values. These variables may be used elsewhere in the program. In SET, there are no subcommands as such, but any legal mathematical function within the parentheses following the SET command is a legal expression.

SET operates according to standard rules of mathematical precedence. For example, multiplication is of higher precedence than addition, so the expression \((3 * 4 + 5)\) is interpreted as 17, not 27. Parentheses are allowed, so that the expression \((3 * (4 + 5))\) is interpreted as 27.

SET can be used to create a brand new variable, to create a new variable using those already defined in event modules, or to modify previously defined variables.

Example:

SET ( A = 100) : Creation of new variable A; a constant.

Example:

LITTER ( 
  RATES (LITTER, DBF) : This portion of a run stream 
  VAR (LITTERS) : creates litters and newborns.
) : SET is used to determine the 
BIRTH ( 
  RATES (BIRTH, DBF) : mean litter size for this birth 
  VAR (NEWBORN) : event, by dividing the number 
) : of newborns by the number of 
SET : litters. SET is further used 
(MEANLIT = : to modify the new variable 
  NEWBORN/LITTERS) : MEANLIT by rounding it to the 
SET : nearest integer. 
(INTIDLIT = : 
  ROUND (MEANLIT)) : 

----------------------------------------
Here are the legal mathematical operators and their precedence level in SET.

<table>
<thead>
<tr>
<th>OPERATOR</th>
<th>PRECEDENCE LEVEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>= (assign)</td>
<td>1</td>
</tr>
<tr>
<td>+ (plus)</td>
<td>2</td>
</tr>
<tr>
<td>- (minus)</td>
<td>2</td>
</tr>
<tr>
<td>* (multiply)</td>
<td>3</td>
</tr>
<tr>
<td>/ (divide)</td>
<td>3</td>
</tr>
<tr>
<td>^ (exponentiation)</td>
<td>4</td>
</tr>
<tr>
<td>- (negate a value)</td>
<td>5</td>
</tr>
</tbody>
</table>

Examples:

\[
\begin{align*}
\text{SET} & (A = 1 + 2 \times 3 ^ 2) : A = 19 \\
\text{SET} & (B = (1 + 2) \times 3 ^ 2) : B = 27 \\
\text{SET} & (C = A + B + 1 ) : C = 47 \\
\text{SET} & (D = A \times B - C) : D = 846 \\
\text{SET} & (E = A - 2) : E = 0.00277 \\
\text{SET} & (F = -2) : F = -2
\end{align*}
\]

SET also recognizes the following mathematical functions.

**ABS(arg1) Function**

The ABS() function accepts a single argument and returns its absolute value.

Example 1: \(\text{SET}( a = \text{ABS}(2) ) \); set a to 2 ;
Example 2: \(\text{SET}( b = -5) \);
Example 3: \(\text{SET}( c = \text{ABS}(y) ) \); set c to 5 ;
Example 4: \(\text{SET}( d = \text{ABS}(x*y) ) \); set d to 10 ;

**CEIL(arg1) Function**

The CEIL() function returns the smallest whole number greater than or equal to its single argument.

Example 1: \(\text{SET}( a = \text{CEIL}(2.7) ) \); set a to 3 ;
Example 2: \(\text{SET}( b = \text{CEIL}(8.1) ) \); set b to 9 ;
Example 3: \(\text{SET}( c = \text{CEIL}(-2.7) ) \); set c to -2 ;
Example 4: \(\text{SET}( d = \text{CEIL}(-8.1) ) \); set d to -8 ;
Example 5: \(\text{SET}( e = \text{CEIL}(d) ) \); set e to -8 ;

**EXP(arg1) Function**

The EXP() function returns the exponential function of its argument.
FLOOR(arg1) Function

The FLOOR() function returns the largest whole number less than or equal to its single argument.

Example 1: SET( a = FLOOR(2.7) ) ; set a to 2 ;
Example 2: SET ( b = FLOOR(8.1) ) ; set b to 8 ;
Example 3: SET ( c = FLOOR(-2.7) ) ; set c to -3 ;
Example 4: SET ( d = FLOOR(-8.1) ) ; set d to -9 ;
Example 5: SET ( c = FLOOR(d) ) ; set c to -9 ;

LN(arg1) Function

The LN() function returns the natural logarithm of its single argument.

LOG(arg1) Function

The LOG() function returns the logarithm to the base10 of its single argument.

MAX(arg1, arg2) Function

The MAX() function returns the maximum of its two arguments.

Example 1: SET( a = MAX(1, 2) ) ; sets a to 2 ;
Example 2: SET( b = MAX(a, 10) ) ; sets b to 10 ;
Example 3: SET( c = MAX(10, a*b) ) ; sets c to 20 ;

MIN(arg1, arg2) Function

The MIN() function returns the minimum of its two arguments.

Example 1: SET( a = MIN(1, 2) ) ; sets a to 1 ;
Example 2: SET( b = MIN(a, 10) ) ; sets b to 1 ;
Example 3: SET( c = MTN(10, a*b) ) ; sets c to 1 ;

MM(arg1, arg2, arg3, arg4, arg5) Function

The MM() functions returns the Michaelis-Menton solution for the five arguments. The first argument is the independent variable. The second and third arguments are the lower and upper limits of the solution. The fourth and fifth arguments are the 50th and 95th percentile saturation parameters.

Example 1: CENSUS( VAR(POPSIZE) )
SET( PROB = MM(POPSIZE, 0, 1, 1.25, 2.5) )

NRAND(arg1, arg2) Function

The NRAND() function returns a normally-distributed random number with a mean expected value equal to the first argument,
and a coefficient of variation equal to the second argument. To get a normally-distributed random number with mean 0 and standard deviation of 1;

Example 1: \( \text{SET( NORMAL\_RANDOM } = 1 - \text{NRAND}(1, 1) \) \)

\[ \boxed{\text{ROUND}(\text{arg1}) \text{ Function}} \]

The \text{ROUND()} function returns the nearest whole number of its argument.

Example 1: \( \text{SET( a = ROUND(1.49999999999) \) \) \}; sets a to 1;
Example 2: \( \text{SET( b = ROUND(1.50) \) \) \}; sets b to 2;
Example 3: \( \text{SET( c = ROUND(-1.4999) \) \) \}; sets c to -1
Example 4: \( \text{SET( d = ROUND(-1.5) \) \) \}; sets d to -2 \)

\[ \boxed{\text{SQRT}(\text{arg1}) \text{ Function}} \]

The \text{SQRT()} function returns the square root of its non negative argument.

\[ \boxed{\text{URAND}(\text{arg1}, \text{arg2}) \text{ Function}} \]

The \text{URAND()} function returns a uniformly-distributed random number within the range \([\text{arg1}, \text{arg2}]\). To generate a random number in the interval \([0,1] \);  

Example 1: \( \text{SET( a = URAND(0, 1) \) \) \}; variable in the range min to max;

All of these functions and operators may be used in any order, as long as the order of precedence and inclusion of parentheses is correct.

Examples:

\[ \text{SET K1 = 100} \]
\[ \text{SET (K = K1) \) \}
\[ \text{BEGIN ( \}
\[ \text{YEAR = 10} \]
\[ \text{BEGIN ( \}
\[ \text{SET (K = NRAND(100,0.10)) \) \}
\[ \text{SET (K1 = K) \) \}
\[ \text{SET (MEANK = (K + K1)/2) \) \}
\[ \text{SET (TINTK = ROUND(MEANK)) \) \}
\[ \text{END ( \}

: Create the variable K as a normally distributed variate each year, with mean 100 and standard deviation 10. Create a variable which averages the K for this year and from the previous year; round this variable to its closest integer value.
SET (NORAN = NRAND(1,1)) : Create a standard normal
SET (STANOR = NORAN - 1) : (0,1) variate

SET (A = NRAND(2.5,0.50)) : Create a normally distributed
SET (B = MAX (A,0)) : variate truncated at 0.

CENSUS ((VAR = N)) : Use a defined variable
SET (K = 500) : from census and a newly
SET (N/NOVERK = N/K) : defined variable to
SET (RATE = MM(NOVERK,0.0,1.0,1.0,1.0,2.0)) : determine a rate using
 : the MM routine.

Many other combinations and permutations are possible. However, keep in mind that variables used on the right-hand side of any equations must be defined, either by a previous VAR subcommand in an event module, or by a previous SET command. Only 1 equation per SET command is legal, but you may have as many as 256 SETs in your Command file. You can also use a variable defined in (or modified by) a SET command as the independent variable in a rate file (see chapter on Rate files).

SETUP ()

The SETUP module contains instructions telling GAPPS whether to actually perform the specified simulation, or merely to run through the instructions inspecting for errors. In addition, a title for the simulation may be specified here.

The entire command file can be "tested" for errors with no running by means of the TEST subcommand. The only legal specifications are ON and OFF. Omitting the TEST subcommand is equivalent to specifying TEST (OFF). In either case, the simulation will be performed (assuming no errors are found in the command file). When syntax errors are found in the command file, with TEST either (ON) or (OFF), the program will stop and display
a message informing you what is wrong. Typical errors will include improper use of parenthesis and semi-colons, missing files, mis-spellings of key words, using variables that have not been previously defined, etc.

If specified, a TITLE will appear in the window at the top of the screen while GPPS is running. The TITLE specification must be enclosed by quotation marks (" "). If no title is specified, the default title of "NO TITLE" will appear at the top of the screen.

Example:

```plaintext
SETUP(
  TEST (ON)
  TITLE ( "THIS IS A TEST")
)
```

: The command file will be checked for errors, but not actually run, even if no errors are found.

```plaintext
SETUP(
  TEST (OFF)
  TITLE ( "THIS IS NOT A TEST")
)
```

: This version will run if no errors are found; equivalent to omitting subcommand TEST

No other subcommands are available in SETUP.

SUMMARY()

The SUMMARY module controls the output of data to the screen during each year of the simulation, and, optionally, to a file. To create a file containing the SUMMARY information, use the REPORT subcommand and specify the name of the file which will contain the information.

The SUMMARY module does not know in advance what data to present. You must tell it via the SYMBOLS subcommand. Up to 13 variables can be requested. SUMMARY will list these variables in the order you ask for them. Variables that can be presented in
SUMMARY are 1) those that have been defined with a VAR command in other modules, 2) those that have been defined by a SET command, 3) the 2 reserved variables, RUN and YEAR. Each module has a VAR subcommand with which the user may specify the name of a variable that will hold a specific piece of information. These are documented in the descriptions of the appropriate commands, and are summarized here:

<table>
<thead>
<tr>
<th>MODULE</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGE</td>
<td>Number of animals aged</td>
</tr>
<tr>
<td>ALLELES</td>
<td>Number of alleles in population</td>
</tr>
<tr>
<td>BEGIN</td>
<td>RUN and YEAR</td>
</tr>
<tr>
<td>BIRTH</td>
<td>Number of births</td>
</tr>
<tr>
<td>BREED</td>
<td>Number of females bred</td>
</tr>
<tr>
<td>CENSUS</td>
<td>Census result</td>
</tr>
<tr>
<td>GENETICS</td>
<td>Mean heterozygosity of population</td>
</tr>
<tr>
<td>HUNT</td>
<td>Number of animals killed in hunt</td>
</tr>
<tr>
<td>IMMIGRATE</td>
<td>Number of immigrants</td>
</tr>
<tr>
<td>INBREED</td>
<td>Mean F statistic over entire population</td>
</tr>
<tr>
<td>LITTER</td>
<td>Number of litters</td>
</tr>
<tr>
<td>MATE</td>
<td>Number of mated males</td>
</tr>
<tr>
<td>PACK</td>
<td>Population size</td>
</tr>
<tr>
<td>PURGE</td>
<td>Number of animals removed</td>
</tr>
<tr>
<td>READ</td>
<td>Number of animals read from specified file</td>
</tr>
<tr>
<td>SURVIVE</td>
<td>Number of natural deaths</td>
</tr>
<tr>
<td>WEAN</td>
<td>Number of weaned offspring</td>
</tr>
<tr>
<td>WRITE</td>
<td>Number of animals written to specified file</td>
</tr>
</tbody>
</table>

Note that these summaries will apply to the time period during which the VAR command was on. This can be particularly important to keep in mind with VARs that report on population size. Population size may change due to various mortality or birth events. The population size reported will be that existing when that particular VAR was activated, which may not necessarily be the population size when SUMMARY is activated.

In addition to these variables, any specified by the user during a SET command may be reported during SUMMARY. The 2 variables RUN and YEAR may always be reported.
Example:

\[ \text{Correct. This run will give data on population size before and after various mortality events. POP1 is pre-mortality pop, POP2 is post-mortality. The number hunted will be a random, normally distributed variable with mean 25\% of the population remaining after natural mortality and standard deviation 10\% of the mean. Summary will provide data on population size at all times during this reduction, and retain it all in a file called MASSACRE.RPT. It will also appear on the screen.} \]

Note also that each SYMBOLS in SUMMARY refers only to that particular module. If, as in the example, there are numerous SURVIVE modules, no single VAR will give the total mortality for the year.

EVENT MODULES NOT REQUIRING RATE FILES

AGE ()

The AGE module performs the simple function of incrementing the age of each population member. It can be placed anywhere within a BEGIN ((YEARS)) - END loop, and may be used more than once within the loop. If AGE is omitted, population members will always "stay the same age", despite the cycling of births,
deaths, etc. Note also that if AGE is placed directly after BIRTH, offspring will immediately become 1 year-old, and 0-year olds will not exist in the population.

Output of the entire population is available via the AGESEX and CASES subcommands. The START, STEP, and STOP subcommands are also available.

Example:

AGE () : Correct. No subcommands needed.

AGE (CASES POP,LST) : Correct. Members' ages will be
(STEP 2) : incremented every other cycle;
) : at that time, a complete
 : population list will be created.

AGE (AGESEX ) : Incorrect. No file name has been
(STOP 100) : specified for the AGESEX sub-
) : command.

ALLELES ()

Each individual in the population possesses 2 alleles at a hypothetical locus. Alleles in GAPPs are simply arbitrary numbers, most often beginning with 1. Newborn individuals randomly receive 1 allele from their mother and 1 from their father. At this hypothetical locus, each individual is either heterozygous (the 2 alleles are different) or homozygous (the 2 alleles are identical).

There may be experimental contexts in which it is desirable to alter the distribution of alleles within the population prior to, or during a simulation run. For example, beginning with a completely "heterozygous" population may yield information on the loss of heterozygosity through the process of mating.

Alleles of individual animals may be altered by editing the
population file directly via dBASEII. This can be a long and
tedious process, especially if the population is at all large.
As an alternative, the ALLELES module may be used to
automatically change the alleles of all members in the population
in 4 specific ways:

1) All members may be given 2 copies of exactly the same
allele. Total number of alleles in population = 1.

2) All members may be given a copy of the same 2 alleles.
Total number of alleles in population = 2.

3) Each member may be given 2 copies of a unique allele.
Total number of alleles in population = N.

4) Each member may be given 2 unique alleles. Total number
of alleles in population = 2N.

Under alternative #1, all members are homozygous and there
is no genetic diversity within the population. Under alternative
#2, all members are heterozygous, but all are genetically
identical. Under alternative #3, all members are homozygous, but
none are genetically identical. Under alternative #4, all
members are heterozygous, and none are identical. The 4 options
are executed under the ALLELES module as follows:

1) ALLELES
   DIVERSITY (MIN)
   HETERO (MTN)
   ; DIVERSITY controls the total number of alleles;
   HETERO controls whether each animal gets 2
different alleles or not (stands for
heterozygosity);
)

2) ALLELES
   DIVERSITY (MIN)
   HETERO (MAX)

3) ALLELES
   DIVERSITY (MAX)
   HETERO (MIN)
Two unique and useful subcommands within CENSUS are AGES and SEXES. These may be used to specify a sub-set of the population for censusing. If they are omitted, CENSUS will automatically tally the entire population. However, using AGES and SEXES allows a CENSUS module to be used for only 1 sex, or for only certain ages of 1 or both sexes. The only proper specifications following the SEXES subcommand are FEMALES and MALES. To specify ages for censusing, legal entries are a series of integer values separated by commas, or ages separated by a dash (e.g. 0,1,2,3-6,8-11,13). SEXES and AGES may be used together.

Examples:

CENSUS ( ) : Incorrect. Repeated
SEXES (FEMALES) : subcommand within
SEXES (MALES) : command. Doesn't make
VAR (BOTH) : sense.
)

CENSUS ( ) : Correct. Variable "POP"
VAR (POP) : will hold the total number
: in the population.
)

CENSUS ( ) : Correct. Two separate
SEXES (FEMALES) : census modules are called;
VAR (FEMS) : each one will tally up
) : a different sex. Two
CFNSUS ( ) : separate variables are
SEXES (MALES) : created, for output or
VAR (MALS) : for use later on in
) : simulation.

CENSUS ( ) : Correct.
AGES (0-2) :
VAR (SUBADULTS):
)
from procedures developed by Henderson (1976) and Quass (1976). Interested users are encouraged to consult these original sources. Initial members of a population are considered to be "founders", and thus all their immediate descendants have $F = 0$.

In a typical run, **INBREED** must occur twice. The first time **INBREED** is called, an array for all sires and dams is set up. The second time it is called, inbreeding coefficients are calculated. Most often, a single, initial call to **INBREED** prior to the main loop will do the trick. If the initial call to **INBREED** is omitted, all future calls will not accurately reflect the level of inbreeding in the population.

The **VAR** subcommand may be used to store $F_{mix}$, the average $F$ value among the population (sum of the individual $F$ values divided by the number in the population). $F_{mix}$ varies between 0 and 1, inclusive.

Other subcommands available in **INBREED** are **AGESEX**, **CASES**, **REPORT**, **PAUSE**, **START**, **STEP**, **STOP**, **SWITCH**, and **TRACE**. **AGESEX** entries consist of the number in each age/sex class, thus it represents another alternative to census the population. **CASES** also includes the entire population. **REPORT** includes the $F_{mix}$, the average inbreeding coefficient among all members, and an analogous statistic among only animals of age 0.

**Example:**

```
[INBREED ()
  RUN (10)]
BEGIN (
  BIRTH (RUN (10))
  BIRTHrates (BIRTH.DBF)
  VAR (BIRTHS)
)
INBREED (
  SWITCH (ON)
  VAR (F)
)
SET (HETERO = 1 - F)
GENETICS (

: Sets up array.

: This section of a command
: file calculates the $F$
: value of each individual,
: the number of alleles in
: the population, and the
: average heterozygosity
: at the hypothetical locus,
: and reports it all to a file
: called GEN.DAT. These data
: will vary each year as
```
REPORT (GEN.DAT) : the genetic make-up of the
       ) : population changes.

Unlike all other modules, the time it takes for INBREED to execute is not simply proportional to the number of animals in the population. Longer pedigrees take much longer to compute than do shorter pedigrees. With a total pedigree exceeding 1,000 animals (alive + dead animals), INBREED will dominate the run time of your program.

PACK (

All individuals within GAPPS possess an identification number. Typically, one begins by numbering animals with #1. As the population progresses through time, animals with low ID's will die and be replaced by animals with higher ID's. Eventually, all animals have large numbers, and many ID's will not be consecutive.

There are advantages to having each member assigned as low a number as possible, and having the numbers be close to each other. The PACK module accomplishes this. It re-numbers individuals without changing any of their other traits. It does not alter relationships among family members. Fig 8.3 shows a typical population after having been run for a number of years; Fig 8.4 shows the same population after having been 'packed'. All traits of members remain unaltered; only their identification numbers have been changed.

The START, STEP, and STOP subcommands may be used to control when PACK is invoked, although its most common use will probably be prior to any loops. As with CENSUS, the ACESEK and CASES subcommands may be used to create output files of the entire population. Additionally, the VAR subcommand may be used to identify an additional variable holding the total population size. In these respects, PACK functions identically to CENSUS.
<table>
<thead>
<tr>
<th>ID</th>
<th>SEX</th>
<th>AGE</th>
<th>MOTHER</th>
<th>FATHER</th>
<th>MATE</th>
<th>CHILD01</th>
<th>CHILD02</th>
<th>CHILD03</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>0</td>
<td>25</td>
<td>0</td>
<td>0</td>
<td>93</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>93</td>
<td>1</td>
<td>17</td>
<td>3</td>
<td>62</td>
<td>50</td>
<td>129</td>
<td>130</td>
<td>131</td>
</tr>
<tr>
<td>105</td>
<td>0</td>
<td>12</td>
<td>17</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Fig. 8.1 A portion of a small population file after having been run for a number of years. The lowest ID number in the population is 50, and the animals between ID numbers 50 and 93 are no longer alive. Animal #50's parents were founders, thus had ID number zero.

<table>
<thead>
<tr>
<th>ID</th>
<th>SEX</th>
<th>AGE</th>
<th>MOTHER</th>
<th>FATHER</th>
<th>MATE</th>
<th>CHILD01</th>
<th>CHILD02</th>
<th>CHILD03</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>25</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>17</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Fig. 8.2. A portion of the same population as shown in Fig. 8.1, after having been renumbered by the PACK module. Note that family relationships among living animals are preserved. The father of animal #2 and the mother of animal #3 are given as zero because they are no longer alive.
PURGE ()

The PURGE command removes a population from current memory. It's primary use will be at the end of a run, before replicating the simulation. When replicate simulations are performed, using the PURGE command gets rid of the population existing in the last year of the first replicate, making room for the initial population to start up again. If PURGE is omitted, the next population read into memory will be appended to the already existing population, rather than replacing it.

Example:

READ ( INPUT (POPIN.DBF) ) : Correct, but will
BEGIN ( RUN (10) ) : append POPIN to the
BEGIN ( YEAR (50) ) : population remaining
. : in memory after year
. : 50 of the preceding
END () : loop.
END ()

Example:

READ ( INPUT (POPIN.DBF) ) : Correct. Will
BEGIN ( RUN (10) ) : run using POPIN
BEGIN ( YEAR (50) ) : each time.
. : The population remaining
. : in year 50 is
END () : irrelevant to the
PURGE () : subsequent loop.
END ()

The PURGE command can also be used to remove only a specified population, using the TAG subcommand. Among attributes of individuals in GAPPS is a population tag (see Chapter 5). If the member is an immigrant, it will have a different tag than a member of the "base" population. The TAG subcommand may be used to purge only members of a specified population. If TAG is
omitted, all individuals are removed.

Example:

```
READ ( INPUT (POPIN.DBF) ) : Correct, but will remove
BEGIN ( RUN (10) ) : all individuals who are
  BEGIN ( YEAR (50) ) : members of population #1,
    : leaving all other members
    : of any other populations.
    :
END () :
PURGE ( TAG (1) ) :
END () :
```

Other subcommands available in PURGE are AGESEX, CASES, PAUSE, START, STEP, STOP, SWITCH, TRACE, and VAR. AGESEX and CASES will report individuals purged. The variable specified by VAR will hold the number of individuals purged.

**EVENT MODULES REQUIRING RATE FILES**

All of the following event modules have at least one mandatory subcommand: RATES. There must be a complete .DBF file in existence on the disk on which GAPPSS resides for each of these modules to run. The rate files provide the information for the probability statements implicit in these modules.

**BIRTH ()**

The BIRTH module creates new population members and determines their gender (male or female). BIRTH applies only to pregnant females, and a litter size for each must be known. Thus, in order for GAPPSS to create new members via BIRTH, the following event modules must also exist in the model:

- `BREED ()` - to impregnate females
- `MATE ()` - to determine a mate for each pregnant female
- `LITTER ()` - to determine litter sizes for mated females
- `WFIN ()` - to allow newborns to become breeders later on
A file with a .DBF extension is necessary to operate BIRTH. It specifies the probabilities of offspring being male. Thus, a constant rate of 0.5 produces an equal sex ratio at birth.

A unique subcommand available in BIRTH is SEED. A separate random number generator exists in GAPPs solely for the purpose of assigning alleles at the hypothetical locus to newborns. It is possible for this random number generator to be completely independent of the larger, global random number generator which is used for all other processes. Thus, for example, one can run a simulation such that all demographic events occur identically each time, with only the assignment of alleles randomized each time. This can be useful in observing the behavior of the hypothetical locus under various demographic assumptions. The SEED subcommand within BIRTH does not control the random number generator for allocation of gender to offspring. The PAUSE, SWITCH, TRACE, START, STEP, and STOP subcommands are available.

The VAR subcommand may be used to hold the total number of animals born during the birth event. The AGESEX, TRACE and CASES subcommands may be used to produce file output (although an AGESEX file will consist mostly of zeros, because all newborns are defined as being 0 years old, thus all other age-classes will be zero).

**Examples:**

BIRTH (  
    CASES (BIRTH.CAS)  
    VAR (BIRTHS)  
  )

: Incorrect. No .DBF file specified for allocation of males and females

BIRTH (  
    RATES (GENDER,DBF)  
    VAR (BIRTHS)  
    START (5)  
  )

: Correct. However, birth events will begin occurring at year 5 and every year thereafter; regardless of other commands.
The BREED module changes the status of females from barren to pregnant. It does not determine litter size or the identity of male mates, nor does it actually produce offspring. It merely tags females as being "mate-able", according to probabilities specified in a required .DBF file. Females are not actually "impregnated" until MATE is called (c.f.). If MATE, LITTER, or BIRTH is omitted, no births will take place. Probabilities of breeding may be specified from 0.0 to 1.0.

The VAR subcommand may be used to hold the number of females bred during the event. The AGESEX, TRACE and CASES subcommands may be used for output. The DEPRESS, PAUSE, SWITCH, START, STEP, and STOP subcommands are available.

DEPRESS is a subcommand that may be used to alter the appropriate rate by a function that takes the individual's inbreeding coefficient (F) into account. It is termed DEPRESS because of the well-known phenomenon known as inbreeding depression. It is available in BREED, MATE, and SURVIVE. A variable must be specified after the DEPRESS subcommand. This variable must be given a value in a previous SET command. The value is the slope of the reduction of the probability (i.e. breeding, mating, or survival probability) as a function of the individual animal's inbreeding coefficient. The equation takes the form:

\[
\text{Rate after inbreeding depression equation} = p \cdot e^{-bf}
\]

where \( b \) is the variable specified by the DEPRESS subcommand, \( F \) is the individual animal's inbreeding coefficient, \( p \) is the probability in the absence of any inbreeding depression, and \( e \) is the base of natural logarithms. This formula is based on genetic theory; for reference, see Morton et al. (1956) or
Calvalli-Sforza and Bodmer (1971).

Examples:

```
BREED
  AGESEX  (PREG.DAT)
  STOP   (100)
) : Incorrect. No .DBF file
: name specified for breeding
: probabilities.
```

```
BREED
  RATES  (BREED.DBF)
  VAR    (BRED)
) : Correct.
: Variable BREED will hold
: number of females bred
: this year.
```

```
INBREED ( )
SET    ( B = 2.2)
BREED
  RATES  (BREED.DBF)
  CASES (BRED.CAS)
  DEPRESS (B)
) : Breeding rates will be
: determined by BREED.DBF,
: but any female with P>0.0
: will have her breeding
: probability lowered by
: being multiplied by
: e^(-2.2F)
```

```
HUNT ( )
```

The HUNT module removes individuals from the population as if hunted. The number of individuals hunted each year must be specified by the NUMBER subcommand, or alternately, by a variable which is specified in the NUMBER subcommand. If a variable is specified (rather than a constant number), it must have been previously defined in a SET command, or by another module. Given that a certain number will be killed, the user must specify probabilities of being killed by age, sex, and family status via a .DBF file specified by the RATES subcommand. If all animals have an equal chance of being hunted, a .DBF file is still required, but all the entries in the .DBF file will be identical in this case.
Examples:

```
HUNT (                                                 : Incorrect. No NUMBER
  RATES (HUNT.DBF) : subcommand.
  )
```

```
HUNT (                                                 : Incorrect. No RATES
  NUMBER (20) : subcommand.
  )
```

```
HUNT (                                                 : Correct. Each year 11
  NUMBER (11) : animals will be killed
  RATES (HUNT.DBF) : according to the relative
  VAR (KILLD) : probabilities specified
  ) : in HUNT.DBF
```

```
CENSUS (                                                 : Each year 10% of the population
  VAR (POP) : is killed by hunting; first, a
  ) : CENSUS is necessary to determine

SET (PRPKIL = POP * 0.1) : population size; SET creates the
  : variable holding 10% of POP;

HUNT (                                                 : HUNT uses HUNT.DBF to allocate
  NUMBER (PRPKIL) : the 10% among the various types
  RATES (HUNT.DBF) : of animals. The age/sex makeup
  AGESEX (HUNT.DAT) : of killed animals is reported
  SWITCH (ON) : in HUNT.DAT
  )
```

By creative use of the SET command, a wide variety of functions may be used to govern the number hunted in any given year. The HUNT module automatically scans the entire population as it decides which animals to hunt. However, it is possible to restrict hunting to 1 sex, or to 1 set of age-classes, by giving some types of animals a 0.00 probability of being hunted. HUNT may also be repeated within the loop, so be judicious use of HUNT, SET, CENSUS and numerous rate files, one can be very particular about which animals get killed.
Example:

```
CENSUS (SEX (MALES))

SET (MHUNT = MALES * 0.2)
HUNT (NUMBER (MIUNIT), RATES (MHUNT,DBF), VAR (MKILL))

HUNT (NUMBER (10), RATES (FHUNT,DBF), VAR (FKILL))
```

MHUNT,DBF gives zero probability of being hunted to all females; FHUNT,DBF does the same for males.

By having 2 separate HUNT modules, the first can can hunt only males, the second only females. The first hunt will take 20% of number of males in the population; the second will take 10 females each year. Various combinations are possible in this way.

Other subcommands available in HUNT include AGESEX, CASES and TRACE for output; as well as PAUSE, SWITCH, START, STEP and STOP.

**IMMIGRATE ()**

The IMMIGRATE module creates new individuals and adds them to the "base" population. As with HUNT, there are 2 mandatory subcommands: RATES and NUMBER. The NUMBER subcommand tells IMMIGRATE how many individuals to create and add. As with HUNT, the specification following the NUMBER subcommand may be a constant, or a variable. If the latter, the variable must be previously defined, either by another module, or by a SET command. The RATES subcommand refers to a mandatory .DBF file, and tells IMMIGRATE the probability of each new individual belonging to each age and sex category.

An additional subcommand in IMMIGRATE is TAG. This refers to a population "tag" which is part of each individual's set of traits. Members of the base population have a tag of 0, unless specifically changed. You may want to keep track of those individuals who were created via IMMIGRATE, so you may specify a separate TAG for them, by specifying an integer value other than 0.
A subcommand unique to IMMIGRATE is BASE. BASE is used to specify the lowest number for alleles given to new immigrants. It may be advantageous to give immigrants different alleles from members of the base populations. This can be accomplished using BASE, by using allele numbers known to be different (probably higher) than any base population member. As with the ALLELES module, you may also specify the type of allelic diversity among immigrants with HETERO (MIN:MAX) and DIVERSITY (MIN:MAX). See the description of ALLELES.

Example:

IMMIGRATE (  
NUMBER (100) : Create an initial population  
RATES (IMMIG.DBF) : with IMMIGRATE. This example  
HETERO (MAX) : will create 100 individuals  
DIVERSITY (MAX) : with age and sex frequency  
) : probability set by IMMIG.DBF.  
WRITE ( : Alleles will total 200.  
OUTPUT (NEWPOP.DBF) : The new population will be called  
NEWP00.DBF

Example:

READ (  
INPUT (BASE.DBF)) : A base population is read  
ALLELES ( : in, and its genetic diversity  
HETERO (MAX) : is maximized prior to the  
DIVERSITY (MAX) : simulations. Each year,  
) : 1 immigrant will join the  
BEGIN ( : population. It will have  
YEAR (10)) : a population "tag", to  
SURVIVE ( : differentiate it from the  
RATES (SURV.DBF)) : base population members.  
IMMIGRATE ( : The first immigrant will  
NUMBER (1) : have alleles 500 and 501;  
RATES (IM.DBF) : the second will have alleles  
BASE (500) : 502 and 503, etc. The  
HETERO (MAX) : addition of these immigrants  
DIVERSITY (MAX) : will thereby increase the total  
TAG (1)) : genetic diversity of the  
GENETICS ( : population. If BASE had been  
REPORT (GEN.RPT)) : specified as 0, no new genetic  
END () : material would have been  
added.

Other subcommands available in IMMIGRATE include the PAUSE,
SWITCH, START, STEP, and STOP. If requested, VAR will contain the number of new immigrants. Output is available through AGESERX, CASES, TRACE and REPORT.

LITTER ()

The LITTER module determines the litter size for each female that has previously been bred and mated (see BREED, MATE). It will not compute litter sizes for any animals if BREED and MATE have not been previously invoked. Probabilities for a litter containing from 1 to 12 offspring may be specified in up to 12 .DBF rate files. LITTER then makes a cumulative distribution out of the probabilities encountered in the .DBF files. It is not necessary for the probabilities in the .DBF files to sum to 1.0; LITTER will do this for you. The .DBF rate files are age-specific, so it is possible to have different distributions of litter sizes for different ages of mothers.

The only mandatory subcommand in LITTER is RATES, and at least 1 .DBF litter rate file must be specified (maximum = 12). The first .DBF file listed in the RATES subcommand is assumed to hold the probability of litter size = 1; the second file is for the probability of litter size = 2; and so on. Other available subcommands in LITTER are AGESERX, CASES, and TRACE for output (these will report on mothers, not offspring); as well as SWITCH, PAUSE, START, STEP and STOP. If invoked, the VAR subcommand will contain the number of litters processed by the module.

Examples:

LITTER ( : Incorrect. No rate file
AGESERX (MOMS.DAT) : specified.
PAUSE (ON)
VAR (LITTERS)
)

LITTER ( : Correct. Litters of
AGESERX (MOMS.DAT) : sizes 1, 2 and 3 will
PAUSE (ON)
VAR (LITTERS) : be calculated.)
The MATE module selects males to be the mates of females that have already been determined to be eligible for mating by BREED. It is necessary to invoke MATE in order for females to successfully breed and have litters. If no eligible males are found in the entire population, no females will breed. Similarly, MATE will find no "matable" females if BREED has not previously been invoked.

The only mandatory subcommand in MATE is RATES, which controls the probability of males becoming mates by their ages. If all males are equally likely to mate with a given female, all ages may be given an identical value (they need not sum to 1.0). If however, males below a specified age are not reproductively active, these ages should be assigned 0 probability, and only reproductively active ages given positive probabilities.

Only males that are no longer under the care of their mother are eligible to MATE. Thus, the WEAN command should also be included in your program. If males of a young age are given positive probabilities of mating, they will mate only if they have been WEANED.

Optional output variables will record the number of males participating in mating. If used, the variable specified by VAR will hold the total number of unique male mates. Other output
can be created by AGSEX, CASES, and TRACE. Other available subcommands are DEPRESS (which operates just as it does in BREED and SURVIVE), PAUSE, START, STEP, STOP, and SWITCH.

Examples:

```
MATE ()
  PAUSE (OFF)
)
```

: Incorrect. No rate file specified.

```
MATE ()
  RATLES (MATE, DBF)
  PAUSE (OFF)
  VAR (MATES)
)
```

: Correct. MATE, DBF gives age-specific probabilities of mating, given that a female has been determined to be "mateable". MATES will hold the # of mates.

READ ()

The READ module accesses and reads a specified population file. A population file must be specified using the INPUT subcommand.

Example:

```
READ ()
```

: Incorrect, no file specified

```
INPUT (POPIN, DBF)
```

: Incorrect, no Command

```
READ ()
  INPUT (POPIN, DBF)
)
```

: Correct. The simulation will run using the initial population in POPIN, DBF

The READ module may be invoked at any time, although its most common use will be very early in the command file, probably before the years loop begins. Most other modules will not run correctly without the population having been read in first.

If READ is used during a run, the newly read-in population will be appended to the population already in memory.
As with PACK, READ can be used as a substitute CENSUS module. The PAUSE, SWITCH, TRACE, VAR, START, STEP, STOP, TAG, AGESEX, and CASES subcommands are available. If used, the TAG subcommand may be used to change the TAG of each individual in the population. If omitted, the default TAG will be 0. When invoked, the variable specified by the VAR subcommand will hold the number of animals processed by the READ command (i.e., the number in the base population). The CASES and AGESEX subcommands will produce files holding the members added.

SURVIVE ()

The SURVIVE module removes animals from the population due to natural mortality. (We use the term "natural mortality" here to refer to all mortality other than from hunting, which can be handled in the HUNT command). SURVIVE requires the subcommand RATES, in which you must specify a .DBF file containing probabilities of surviving natural mortality. Unlike in HUNT, where the user may specify a number of animals to be killed, there is no deterministic component to SURVIVE. One can only specify the probability that an animal of a certain age/sex/family status category will survive each time the module is called.

As with all other modules, SURVIVE may be called numerous times during a yearly cycle. Numerous calls to SURVIVE may be used to model different types of survival (as with HUNT), or they may be used to emulate survival over shorter periods than they entire year. For example, one might want to call SURVIVE 12 times each year, and think of survival as a monthly, rather than yearly, event. In this case, the survival rates in the relevant .DBF file must be appropriately higher.

SURVIVE has a built-in limitation: it will not allow survival rates of greater than 1.0 or less than 0.0. Even if your rate file calls for such a rate, SURVIVE will not recognize it.
As with MATE and BREED, the DEPRESS subcommand is available, to emulate inbreeding depression. If a SURVIVAL module is called with the primary intent of emulating inbreeding depression in a subset of age-classes, the remaining age-classes should be given high enough rates in the .DBF file that, even when reduced by DEPRESS, they will be greater than 1.0. (Remember that when DEPRESS is used, all rates will be multiplied by the quantity: e^{-bf}, and if b = 0 and F = 0, the resulting rate will be less than 1.0 even if the original rate was 1.0).

Although the module is called SURVIVE, and the rates in the appropriate .DBF file are survival, rather than mortality rates, the optional output from SURVIVE is concerned with the animals that die, not the survivors. Thus, when a VAR is specified, it will hold the number of animals that died, not those that survived. Similarly, the AGSEX, CASES and TRACE output subcommands will hold data on the animals dying, not those surviving. (One can always call a CENSUS module directly after the SURVIVAL module to retain information on the survivors). Other available subcommands include PAUSE, START, STEP, STOP, and SWITCH.

Examples:

```
SURVIVE ( AGESEX (DEAD.DAT) ) : Incorrect. No Rate file specified.
```

```
SURVIVE ( RATES (SURV.DBF) AGESEX (DEAD.DAT) ) : Correct. DEAD.DAT will hold the age/sex matrix of those animals dying.
```

```
INBREED () : Correct. Survival rates for all age-classes will be reduced by e^{-1.5F}, where F is each individual's inbreeding coefficient. This module will only operate in odd years, however, and will
```

```
SET (FATAL = 1.5) : for all age-classes will
```

```
SURVIVE ( RATES (SURV.DBF) DEPRESS (FATAL) CASES (DEAD.CAS) STEP (2) )
```

As with MATE and BREED, the DEPRESS subcommand is available, to emulate inbreeding depression. If a SURVIVAL module is called with the primary intent of emulating inbreeding depression in a subset of age-classes, the remaining age-classes should be given high enough rates in the .DBF file that, even when reduced by DEPRESS, they will be greater than 1.0. (Remember that when DEPRESS is used, all rates will be multiplied by the quantity: e^{-DF}, and if b = 0 and F = 0, the resulting rate will be less than 1.0 even if the original rate was 1.0).

Although the module is called SURVIVE, and the rates in the appropriate .DBF file are survival, rather than mortality rates, the optional output from SURVIVE is concerned with the animals that die, not the survivors. Thus, when a VAR is specified, it will hold the number of animals that died, not those that survived. Similarly, the AGESEX, CASES and TRACE output subcommands will hold data on the animals dying, not those surviving. (One can always call a CENSUS module directly after the SURVIVAL module to retain information on the survivors). Other available subcommands include PAUSE, START, STEP, STOP, and SWITCH.

Examples:

```
SURVIVE (                       : Incorrect. No Rate file
        AGESEX (DEAD.DAT)      : specified.
        )
```

```
SURVIVE ( RATES (SURV.DBF)       : Correct. DEAD.DAT will
        AGESEX (DEAD.DAT)      : hold the age/sex matrix
        )                       : of those animals dying.
```

```
TNRBREED ()                       : Correct. Survival rates
SET  (FATAL = 1.5)                : for all age-classes will
SURVIVE (                         : bc reduced by e^{-1.5F}, where
        RATES (SURV.DBF)      : F is each individual's in-
        DEPRESS (FATAL)       : breeding coefficient. This
        CASES (DEAD.CAS)      : module will only operate in
        STEP (2)             : odd years, however, and will
```
VAR (DIED) : be omitted entirely in even years.

WEAN ()

GAPPS assumes that an animal stays under the care of its mother after being born until the user explicitly tells it that the young animal has become independent. WEAN changes the category of a young animal from "with mother" to alone. GAPPS will not allow the mother of offspring to participate in breeding again until they have been weaned. Similarly, youngsters who have not yet been weaned are not permitted to BREED or MATE. GAPPS does not allow an animal to start its own family while still under the care of its mother. In species in which adult females breed every year (or more often), WEAN should be included shortly after BIRTH, to allow the mother to participate in the next BREED event. However, WEAN may be usefully employed with species in which females care for their young over a period of years, to govern the length of the non-breeding interval for adult females.

There may be a tendency to forget to include WEAN. We suggest that it be routinely included as part of the life-history structure.

The mandatory RATES subcommand specifies a .DBF file, which contains probabilities of weaning by age of offspring. Thus, if adult females breed every year, the probability of 0-year olds weaning should be set to 1.0. WEAN operates from the perspective of the mother, so that if more than 1 offspring belong to the same litter, all or none will wean. The biological assumption is that the mother is the active participant; she "pushes" the offspring away, they don't just "leave" by themselves.

The optional outputs in WEAN refer to the weaned animals, however, not the mothers. Thus, VAR will hold the number of weaned young, and AGSEX, CASES and TRACE all hold data on weaned offspring. Other subcommands available are PAUSE, START, STEP,
STOP, and SWITCH.

Examples:

```plaintext
WEAN (     : Incorrect. No RATE file
   PAUSE (ON) : specified.
)
```

```plaintext
WEAN (RATES (WEAN.DBF) : Correct. Each animal still
   PAUSE (ON) : under its mother's care will
                   : be displayed, and weaned or
                   : not on the screen.
)
```

```plaintext
WRITE ()
```

The WRITE module operates exactly like the READ module in reverse. When invoked, it writes a population file in dBASEII format. A population file created by the WRITE module is automatically saved on the device running GAPPs (e.g. hard disk, floppy disk). This new population file may in turn be used to run a new simulation, by calling for it in a future READ command. Whereas READ is essential to running a simulation (the program cannot run if it has no population in memory on which to act), WRITE is optional. It is not necessary to retain a population for future runs. However, it can be a very useful feature.

Like READ, WRITE cannot be invoked without specifying the name of the file to which the population will be written via the OUTPUT subcommand.
Example:

WRITE () : Incorrect, no file specified

-------------------------------
OUTPUT (POPOUT.DBF) : Incorrect, no Command

-------------------------------
WRITE ( OUTPUT (POPOUT.DBF) ) : Correct, will save pop'n
 : in memory to file
 : POPOUT.DBF

An additional subcommand available within WRITE is MODE. The 2 legal specifications are APPEND and CREATE. When APPEND is specified, WRITE will not erase any current files with the same file name, and members will be appended to any in the existing file. When CREATE is invoked, any old files by that name will first be erased, and only the newest version will be saved. When MODE is omitted, the default is CREATE. Other subcommands available within WRITE include AGSEX, CASES, PAUSE, SWITCH, TAG, TRACE, VAR, START, STEP, and STOP subcommands are available. The TAG subcommand may be used to change the population tag of animals as they are written to the file. When invoked, the variable specified by VAR will hold the number of animals written to the file.
# COMMAND-SUBCOMMAND CROSS REFERENCE TABLE

## SUBCOMMAND

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>SUBCOMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGE</td>
<td>0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>ALLELES</td>
<td>0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>BEGIN</td>
<td>0 R R</td>
</tr>
<tr>
<td>BIRTH</td>
<td>0 R O 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>BREED</td>
<td>0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>CENSUS</td>
<td>0 O 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>DUMP</td>
<td>0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>END</td>
<td>**</td>
</tr>
<tr>
<td>GENETICS</td>
<td>0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>HUNT</td>
<td>0 0 R O R O 0 0 0 0</td>
</tr>
<tr>
<td>IMMIGRATE</td>
<td>0 0 0 R O R O 0 0 0 0</td>
</tr>
<tr>
<td>INBREED</td>
<td>0 0 0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

0 = Optional Subcommand  
R = Required Subcommand  
blank = Invalid Subcommand  

* Either RUN or YEAR subcommand, but not both, are required with the BEGIN command.  
** The END command has no subcommands.
### Command-Subcommand Cross Reference Table

#### Subcommand

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>SUBCOMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>LITTER</td>
<td>* 0 0</td>
</tr>
<tr>
<td>MATE</td>
<td>0 0 0</td>
</tr>
<tr>
<td>PACK</td>
<td>0 0</td>
</tr>
<tr>
<td>PURGE</td>
<td>0 0</td>
</tr>
<tr>
<td>READ</td>
<td>0 0 R</td>
</tr>
<tr>
<td>RESEED</td>
<td>0 R 0 0 0</td>
</tr>
<tr>
<td>SET **</td>
<td></td>
</tr>
<tr>
<td>SETUP</td>
<td>0 0</td>
</tr>
<tr>
<td>SUMMARY</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>SURVIVE</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>WEAN</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>WRITE</td>
<td>0 0 0 R 0 0</td>
</tr>
</tbody>
</table>

* = Optional Subcommand  
R = Required Subcommand  
blank = Invalid Subcommand

---

* The *RATES* subcommand for LITTER must contain a file name argument for each possible litter size.

** The SET command has no subcommands; an algebraic expression must appear within its command parenthesis.
OUTPUT FILES

GAPPS is designed to produce output in the form of files, which can be accessed by other programs for purposes of sorting, cataloging, summarizing, analyzing, etc. Most all event modules can produce output files in 4 formats: AGESEX, CASES, REPORT, and TRACE. In addition, modules GENETICS, INBREED, and SUMMARY have specially tailored REPORT files.

All output files in GAPPS begin with 1 blank record.

AGESEX FILES

AGESEX files are ASCII character files written by GAPPS3 in response to an AGESEX subcommand. Each AGESEX subcommand results in two records being written to the AGESEX file. The first record contains male age class counts, and the second contains female age class counts. The format is:

<table>
<thead>
<tr>
<th>Beg</th>
<th>End</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>8</td>
<td>A8</td>
<td>&quot;AGESEX&quot; in columns 1-6, columns 7-8 are blank.</td>
</tr>
<tr>
<td>9</td>
<td>21</td>
<td>12</td>
<td>A12</td>
<td>Name of command event generating the record.</td>
</tr>
<tr>
<td>21</td>
<td>25</td>
<td>5</td>
<td>I5</td>
<td>RUN number.</td>
</tr>
<tr>
<td>26</td>
<td>30</td>
<td>5</td>
<td>I5</td>
<td>YEAR number.</td>
</tr>
<tr>
<td>31</td>
<td>32</td>
<td>2</td>
<td>I2</td>
<td>Sex code; 0=male, 1=female.</td>
</tr>
<tr>
<td>33</td>
<td>37</td>
<td>5</td>
<td>I5</td>
<td>Number of animals in age class 0 for this sex.</td>
</tr>
<tr>
<td>38</td>
<td>42</td>
<td>5</td>
<td>I5</td>
<td>Number of animals in age class 1 for this sex.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>179</td>
<td>182</td>
<td>5</td>
<td>I5</td>
<td>Number of animals in age class 29 for this sex.</td>
</tr>
<tr>
<td>183</td>
<td>187</td>
<td>5</td>
<td>I5</td>
<td>Total animals in all age classes for this sex.</td>
</tr>
</tbody>
</table>
CASES files are ASCII character files written by GAPPs3 in response to a CASES subcommand. Each CASES subcommand results in one record being written for each animal influenced by the command. The AGE, ALLELES, GENETICS, INBREED, PACK, READ, and WRITE commands write a CASES record for every animal in the population. The BIRTH, BREED, CENSUS, HUNT, IMMIGRATE, LITTER, MATE, PURGE, SURVIVE, and WEAN commands write a CASES record only for each animal affected by the command event.

The format is:

<table>
<thead>
<tr>
<th>Beg End</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-8</td>
<td>8</td>
<td>A8</td>
<td>&quot;CASES&quot; in columns 1 5, columns 6 8 are blank.</td>
</tr>
<tr>
<td>9-20</td>
<td>12</td>
<td>A12</td>
<td>Name of command event generating the record.</td>
</tr>
<tr>
<td>21-25</td>
<td>5</td>
<td>I5</td>
<td>RUN number (0 65535).</td>
</tr>
<tr>
<td>26-30</td>
<td>5</td>
<td>I5</td>
<td>YEAR number (0 65535).</td>
</tr>
<tr>
<td>31-35</td>
<td>5</td>
<td>I5</td>
<td>Animal id (1 16378).</td>
</tr>
<tr>
<td>36-36</td>
<td>1</td>
<td>I1</td>
<td>Sex code; 0=male, 1=female.</td>
</tr>
<tr>
<td>37-38</td>
<td>2</td>
<td>I2</td>
<td>Age (0 29).</td>
</tr>
<tr>
<td>39-39</td>
<td>1</td>
<td>I1</td>
<td>Status; 0=available, 1=bred, 2=mated, 3=littered, 4=birthed.</td>
</tr>
<tr>
<td>40-40</td>
<td>1</td>
<td>I1</td>
<td>Parent code; 0=without parents, 1=with parents.</td>
</tr>
<tr>
<td>41-42</td>
<td>2</td>
<td>I2</td>
<td>Offspring (0-12).</td>
</tr>
<tr>
<td>43-47</td>
<td>5</td>
<td>I5</td>
<td>Mother's id (1-16378).</td>
</tr>
<tr>
<td>48-52</td>
<td>5</td>
<td>I5</td>
<td>Father's id (1-16378).</td>
</tr>
<tr>
<td>53-58</td>
<td>6</td>
<td>F6.4</td>
<td>Inbreeding coefficient &quot;f&quot;.</td>
</tr>
<tr>
<td>59-61</td>
<td>3</td>
<td>I3</td>
<td>Animal's allele 1 population tag (0-255).</td>
</tr>
<tr>
<td>62-64</td>
<td>3</td>
<td>T3</td>
<td>Animal's allele 2 population tag (0-255).</td>
</tr>
<tr>
<td>65-69</td>
<td>5</td>
<td>I5</td>
<td>Animal's allele 1 (0-65535).</td>
</tr>
<tr>
<td>70-74</td>
<td>5</td>
<td>I5</td>
<td>Animal's allele 2 (0-65535).</td>
</tr>
<tr>
<td>75-79</td>
<td>5</td>
<td>I5</td>
<td>Mate's id (1-16378).</td>
</tr>
<tr>
<td>80-82</td>
<td>3</td>
<td>I3</td>
<td>Mate's allele 1 population tag (0-255).</td>
</tr>
<tr>
<td>83-85</td>
<td>3</td>
<td>I3</td>
<td>Mate's allele 2 population tag (0-255).</td>
</tr>
<tr>
<td>86-90</td>
<td>5</td>
<td>I5</td>
<td>Mate's allele 1 (0-65535).</td>
</tr>
<tr>
<td>91-95</td>
<td>5</td>
<td>T5</td>
<td>Mate's allele 2 (0-65535).</td>
</tr>
<tr>
<td>96-97</td>
<td>2</td>
<td>I2</td>
<td>Most recent litter size (0-12).</td>
</tr>
<tr>
<td>98-102</td>
<td>5</td>
<td>I5</td>
<td>Offspring 1 id (1-16378).</td>
</tr>
<tr>
<td>103-107</td>
<td>5</td>
<td>I5</td>
<td>Offspring 2 id (1-16378).</td>
</tr>
<tr>
<td>108-112</td>
<td>5</td>
<td>I5</td>
<td>Offspring 3 id (1-16378).</td>
</tr>
<tr>
<td>113-117</td>
<td>5</td>
<td>I5</td>
<td>Offspring 4 id (1-16378).</td>
</tr>
<tr>
<td>118-122</td>
<td>5</td>
<td>I5</td>
<td>Offspring 5 id (1-16378).</td>
</tr>
<tr>
<td>123-127</td>
<td>5</td>
<td>I5</td>
<td>Offspring 6 id (1-16378).</td>
</tr>
<tr>
<td>128-132</td>
<td>5</td>
<td>I5</td>
<td>Offspring 7 id (1-16378).</td>
</tr>
<tr>
<td>133-137</td>
<td>5</td>
<td>I5</td>
<td>Offspring 8 id (1-16378).</td>
</tr>
</tbody>
</table>
The REPORT subcommand writes command specific records to the specified ASCII file. Commands requiring a RATES subcommand can write its current rate table to a file via the REPORT subcommand. This includes the BIRTH, BREED, HUNT, IMMIGRATE, LITTER, MATE, SURVIVE, and WEAN commands. The REPORT records written by the GENETICS, INBREED, and SUMMARY commands, on the other hand, are unique to those commands and have their own format.

Rates REPORT Files for BIRTH, BREED, HUNT, IMMIGRATE, LITTER, MATE, SURVIVE, and WEAN Commands:

Each REPORT subcommand for these events writes one record for each unique rate in its rate table in memory.

<table>
<thead>
<tr>
<th>Beg End</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-8</td>
<td>8</td>
<td>A8</td>
<td>&quot;RATE&quot; in columns 1-4, columns 5-8 are blank.</td>
</tr>
<tr>
<td>9-20</td>
<td>12</td>
<td>A12</td>
<td>Name of command event generating the record.</td>
</tr>
<tr>
<td>21-25</td>
<td>5</td>
<td>I5</td>
<td>RUN number.</td>
</tr>
<tr>
<td>26-30</td>
<td>5</td>
<td>I5</td>
<td>YEAR number.</td>
</tr>
<tr>
<td>31-94</td>
<td>64</td>
<td>A64</td>
<td>Source rate file name.</td>
</tr>
<tr>
<td>95-104</td>
<td>8</td>
<td>A8</td>
<td>DBF rate file type; &quot;AGE&quot;, &quot;SEXAGE&quot;, or &quot;POSA&quot;</td>
</tr>
<tr>
<td>105-109</td>
<td>5</td>
<td>I5</td>
<td>Unique rate sequence number.</td>
</tr>
<tr>
<td>110-111</td>
<td>5</td>
<td>I5</td>
<td>Rate key.</td>
</tr>
<tr>
<td>115-122</td>
<td>8</td>
<td>A8</td>
<td>Rate type; &quot;CON&quot;, &quot;SL&quot;, or &quot;MM&quot;.</td>
</tr>
<tr>
<td>123-138</td>
<td>16</td>
<td>A16</td>
<td>Independent variable name.</td>
</tr>
<tr>
<td>139-146</td>
<td>8</td>
<td>F8</td>
<td>Parameter 1 value.</td>
</tr>
<tr>
<td>147-154</td>
<td>8</td>
<td>F8</td>
<td>Parameter 2 value.</td>
</tr>
<tr>
<td>155-162</td>
<td>8</td>
<td>F8</td>
<td>Parameter 3 value.</td>
</tr>
<tr>
<td>163-170</td>
<td>8</td>
<td>F8</td>
<td>Parameter 4 value.</td>
</tr>
<tr>
<td>171-183</td>
<td>13</td>
<td>F13,6</td>
<td>Current rate value.</td>
</tr>
</tbody>
</table>
GENETICS REPORT Files

Each GENETICS ( REPORT(filename.ext) ) command writes one byte record containing the population's current genetic diversity and heterozygosity measures.

<table>
<thead>
<tr>
<th>Beg End</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1- 8</td>
<td>8</td>
<td>A8</td>
<td>&quot;REPORT&quot; in columns 1-6, columns 7-8 are blank.</td>
</tr>
<tr>
<td>9- 20</td>
<td>12</td>
<td>A12</td>
<td>&quot;GENETICS&quot; in columns 9-16, columns 17-20 blank.</td>
</tr>
<tr>
<td>21- 25</td>
<td>5</td>
<td>I5</td>
<td>RUN number.</td>
</tr>
<tr>
<td>26- 30</td>
<td>5</td>
<td>I5</td>
<td>YEAR number.</td>
</tr>
<tr>
<td>31- 35</td>
<td>5</td>
<td>I5</td>
<td>Diversity (1-65535).</td>
</tr>
<tr>
<td>37- 44</td>
<td>8</td>
<td>F8.6</td>
<td>Heterozygosity (0,1)</td>
</tr>
</tbody>
</table>

INBREED REPORT Files

Each INBREED ( REPORT(filename.ext) ) command writes one record containing the mean population F value, and the corresponding value for animals of age 0.

<table>
<thead>
<tr>
<th>Beg End</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1- 8</td>
<td>8</td>
<td>A</td>
<td>&quot;REPORT&quot; in columns 1-6, columns 7-8 are blank.</td>
</tr>
<tr>
<td>9- 20</td>
<td>12</td>
<td>A12</td>
<td>&quot;INBREED&quot; in columns 9-15, columns 16-20 blank.</td>
</tr>
<tr>
<td>21- 25</td>
<td>5</td>
<td>I5</td>
<td>RUN number.</td>
</tr>
<tr>
<td>26- 30</td>
<td>5</td>
<td>I5</td>
<td>YEAR number.</td>
</tr>
<tr>
<td>31- 42</td>
<td>12</td>
<td>F12.8</td>
<td>Mean F among all animals (0,1)</td>
</tr>
<tr>
<td>43- 54</td>
<td>12</td>
<td>F12.8</td>
<td>Mean F among animals of age 0 (0,1)</td>
</tr>
</tbody>
</table>

SUMMARY REPORT Files

The SUMMARY ( REPORT (filename.ext) ) command writes one record for each year, containing all the SYMBOLS (var, var) list to the specified file.

<table>
<thead>
<tr>
<th>Beg End</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1- 8</td>
<td>8</td>
<td>A8</td>
<td>&quot;REPORT&quot; in columns 1-6, columns 7-8 are blank.</td>
</tr>
</tbody>
</table>
21-25  5  I5  RUN number.
26-30  5  I5  YEAR number.
31-31  1  1X  Blank.
32-40  9  F9.3  Variable value.
41-49  9  F9.3  Variable value.
50-58  9  F9.3  Variable value.
59-67  9  F9.3  Variable value.
68-76  9  F9.3  Variable value.
77-85  9  F9.3  Variable value.
86-94  9  F9.3  Variable value.
95-103 9  F9.3  Variable value.
104-112 9  F9.3  Variable value.
113-121 9  F9.3  Variable value.
122-130 9  F9.3  Variable value.
131-139 9  F9.3  Variable value.
140-148 9  F9.3  Variable value.

TRACe FILES

TRACe files contain one record for each command using a TRACe(filename.ext) subcommand. Its primary purpose is to record the successful completion of the command for debugging purposes.

<table>
<thead>
<tr>
<th>Beg</th>
<th>End</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>8</td>
<td>A8</td>
<td>&quot;TRACe&quot; in columns 1-5, columns 6-8 are blank.</td>
</tr>
<tr>
<td>9</td>
<td>20</td>
<td>12</td>
<td>A12</td>
<td>Name of command event generating the record.</td>
</tr>
<tr>
<td>21</td>
<td>25</td>
<td>5</td>
<td>I5</td>
<td>RUN number.</td>
</tr>
<tr>
<td>26</td>
<td>30</td>
<td>5</td>
<td>I5</td>
<td>YEAR number.</td>
</tr>
<tr>
<td>31-EOR</td>
<td>Variable</td>
<td>%s</td>
<td>Command specific TRACe message.</td>
<td></td>
</tr>
</tbody>
</table>
EXAMPLES

Often, the best way to learn something is by example. Here, we briefly present some examples. Each one shows the objective in simulation modeling (a particular problem), a Command file used to solve the problem, and some commentary on how GAPPS can be used.

Example 1: Comparing age-structures from hunts and those from the standing population.

An application that might be useful in a wildlife management setting is comparing age-structures from the population and that from a hunt, given certain assumptions about the harvest pressure and relative vulnerabilities of different types of animals. Rate file HUNT.DBF contains the modeler's assumptions about the relative vulnerability of ages and sexes. Twenty animals will be killed each year. Here is the Command file:

```bash
SETUP (TITITLE ("Age-structures: population vs. hunt"))
READ (INPUT(POPIN,DBF))
BEGIN (YEAR(50))
    SURVIVE (RATES(SURV,DBF))
    ; Take census, retain age structure of population;
    CENSUS (VAR(POLP))
        AGESX(POLP,DAT))
    ; Hunt 20 each year, retain age structure of kill;
    HUNT (NUMBER(20))
        AGESX(HUNT,DAT)
        RATES (HUNT,DBF)
    ; Reproduce and age;
    BREED (RATES(BREED,DBF))
    MATE (RATES(MATE,DBF))
    LITTER (RATES(LIT1,DBF,LIT2,DBF,LIT3,DBF,LIT4,DBF))
    AGE ()
    BIRTH (RATES(FEM50,DBF))
    WEAN (RATES(WEAN,DBF))
    SUMMARY (SYMBOLS(YEAR,POP))
```
Example 2: A population with separate density-dependent functions for males and females.

One of the best uses of GAPS is in exploring the consequences of different types of density-dependent functions of population behavior. A typical model might use the total number of animals as the independent variable, but GAPS makes it possible to use various subsets of the population as the independent variable, or even such quantities as the number of young, number of matings, etc. Here, males and females both have density-dependent survival, but the dependencies are sex-specific, as might occur if the sexes occupied separate ranges during the most stressful season. Here is the Command file:

```

SETUP {
  TITLE ("Separate sex density response in survival")
  BEGIN (RUN(100))
    READ (INPUT (POPIN.DBF))
    BEGIN (YEAR(50))
      ;Do separate sex censuses;
      CENSUS (             
        SEXES (FEMALES)   
        VAR (DOES))
      CENSUS (             
        SEXES (MALES)    
        VAR (BUCKS))
      ;Do separate sex survival
        Survival rates for males are 1.0 in FEMSURV
        Female survival dependent on number of DOES;
      SURVIVAL (           
        RATES (FEMSURV.DBF) 
        VAR (FKILL))
      ;Survival rates for females are 1.0 in MALSURV
        Male survival dependent on number of BUCKS;
      SURVIVAL (           
        RATES (MALSURV.DBF) 
        VAR (MKILL))
      ;Density independent natality rates;
      BREED (RATES(BREED.DBF))
      MATE (RATES(MATE.DBF))
      LITTER (RATES(L1.DBF, L2.DBF, L3.DBF, L4.DBF))
```
Example 3: An inversely density-dependent function for breeding.

It may be desirable to model the situation in which not all females breed if there are not enough males. In this next example, the probability of a female breeding is proportional to the ratio of adult males to adult females. If the ratio becomes too small, there are not enough males to fertilize all the eligible females, and some will likely not breed. Rate file MFBRD uses a straight line function (SL) relating the independent variable RATIO to the probability of adult females breeding. Here is the Command file:

```plaintext
SETUP (
  TITLE ("Breeding less likely when M:F' ratio too low")
  READ (INPUT (POFIN.dbf))
  REGTN (YEAR (50))
    ; Census adults of both sexes;
  CENSUS (
    SEXES (FEMALES)
    AGES (2-29)
    VAR (ADFEM)
  CENSUS (
    SEXES (MALES)
    AGES (2-29)
    VAR (ADMAL)

  ; Calculate ratio of males to adult females;
  SET (RATIO = ADMAL/ADFEM)

  ; Rate file MFBRD sets breeding probabilities for females
    as a linear function of RATIO. When RATIO is too low,
    breeding can cease altogether. When RATIO is high,
    breeding rates can become 1.0;
  BREED (
    RATES (MFBRD.dbf)
    VAR (BRED)
)`
Example 4: Bringing in an average of 0.5 immigrants per year.

This example demonstrates the use of IMMIGRATE, START and STEP. Using these 3, we can emulate a population in which, on average, 0.5 immigrants enter each year, after a specified year. Here is the Command file:

```
SETUP()
  TITLE("Immigrants enter the population in year 10")
BEGIN (RUN (100))
  READ (INPUT (BASEPOP.DBF))
BEGIN (YEAR (100))
  SET (IMM = 0)
  SURVIVE (RATES (SURV.DBF))
  BREED (RATES (BREED.DBF))
  MATE (RATES (MATE.DBF))
  AGE ()
  LITTER (RATES (LIT1.DBF,LIT2.DBF))
  BIRTH (RATES (GENDER.DBF))
  CENSUS (VAR (BASE))
 ;Starting at year 10, and every other year thereafter, an immigrant enters the population. The probability of an immigrant coming from each age/sex class is specified by file IMMIG.DBF;
IMMIGRATE(
  START (10)
  STEP (2)
  NUMBER (1)
  RATES (IMMIG.DBF)
  VAR (IMM))
  WEAN (RATES (WEAN.DBF))
  CENSUS (VAR (ALL))
  SUMMARY (REPORT (IMMIG.RPT)
               SYMBOLS (BASE,IMM,ALL))
END ()
```
Example 5: Examining the loss of genetic diversity at the hypothetical locus.

Conservation biologists are interested in the effects of genetic drift and inbreeding on small populations. One way to examine this in modeled populations is with our hypothetical locus. We can begin with the maximum possible allelic diversity and heterozygosity, and observe the rate of loss of these 2 genetic indicators at the locus with the GENETICS report command. Here is the Command file:

```
SETUP (
    TITLE ("The loss of genetic diversity at the hypothetical locus")
)
BEGIN (RUN (25))
    READ (INPUT (POPIN.DBF))
    PACK ()
    ;Begin by maximizing the number of alleles in the population;
    ALLELES (
        HETERO (MAX)
        DIVERSITY (MAX))
    BEGIN (YEAR (50))
        SURVIVE (RATES (SURV1.DBF) VAR (DEAD))
        BREAT (RATES (BREAT.DBF) VAR (BRED))
        MATE (RATES (MATE.DBF) VAR (MATED))
        LITTER (RATES (L1.DBF,L2.DBF,L3.DBF) VAR (LITS))
        AGE ()
        BIRTH (RATES (GENDER.DBF) VAR (BABIES))
        ;Retain data on number of alleles and average heterozygosity at the locus;
        GENETICS (REPORT (GEN.DAT)
            VAR (H))
        WEAN (RATES (WEAN.DBF))
        CENSUS (VAR (POP))
        SUMMARY (SYMBOLS (RUN,YEAR
            DEAD,BRED,MATED,LITS,BABIES,H))
    END ()
    PURGE ()
END ()
```

Example 6: Examining the variation in the loss of genetic diversity at the hypothetical locus.

...
diversity given the same set of demographic events.

This example is similar to Example 5, except that we allow the transmission of alleles to vary randomly in each run, while keeping the demographic events exactly the same each time. This is possible because of the separate random number generator in BREED which controls the transmission of alleles from parents to offspring. The SEED number for all other demographic events is reset each time, thus producing the identical patterns; the SEED for BREED is reset according to the computer's clock (which will be different at each run), thus will vary independently. Here is the Command file:

```
SETUP (
  TITLE ("The loss of genetic diversity at the hypothetical locus")
); NOTE: This is similar to the previous example, but in each run
  the demographics will remain exactly the same. Only the
  transmission
  of alleles from parents to offspring will vary randomly from
  run to
  run;
BEGIN (RUN (25))
  RESEED (
    SEED (13028))
; The number 13028 is arbitrary;
READ (INPUT (POPin.DBF))
PACK ()
; Begin by maximizing the number of alleles in the
population;
ALLELES (
  HETERO (MAX)
  DIVERSITY (MAX))
BEGIN (YEAR (50))
  SURVIVE (RATES (SURV1.DBF) VAR (DEAD))
  BREED (RATES (BREED, DBF))
    VAR (BRED))
  MATE (RATES (MATE, DBF) VAR (MATED))
  LITTER (RATES (L1, DBF, L2, DBF, L3, DBF) VAR (L.TTS))
AGE ()
; SEED of 0 within BIRTH allows it to vary while other
demographic events always begin with seed 13028;
BIRTH (RATES (GENDER, DBF) SEED (0) VAR (BABIES))
; Retain data on number of alleles and average
heterozygosity at the locus;
GENETICS (REPORT (GEN.DAT)
  VAR (H))
```
Example 7: Examining the loss of genetic diversity by using the increase in average population inbreeding level.

A more rigorous manner in which to examine the expected loss of genetic diversity due to small population size is to examine each individual's inbreeding coefficient, because pedigree analysis provides a better estimate of the status of an animal's genome than does a single locus. Thus, we can perform a similar analysis as in Example 5, but using the inverse of the mean F of the population (which may easily be obtained from the mean F value). The rate of loss of "noninbreeding" will be similar to the rate of loss of heterozygosity, but the estimate in this case will be more precise than in Example 5. It will also take longer to run, because the pedigree analysis is the most involved of all the event modules. Note as well, that INBREED must be called once at the beginning of the run stream to initialize the necessary arrays, and to calculate F values for the initial population. After this initial call, it may be called as often as desired. Here is a Command file:

```
SETUP (
 TITLE ("The increase in average inbreeding")
 BEGIN (RUN (25))
 RESEED (
   SEED (0))
 READ (INPUT (POPIN.DBF))
   ;Allocate arrays for sires and dams of each individual, 
   and calculate F values for initial population;
 INBREED ()
 BEGIN (YEAR (50))
   SURVIVE (RATES (SURV1.DBF) VAR (DEAD))
   BREED (RATES (BREED.DBF) 
         VAR (BRED) )
   MATE (RATES (MATE.DBF) VAR (MATED))
   LETTER (RATES (L1.DBF,L2.DBF,L3.DBF) VAR (LITTS))
```
AGE ()
BIRTH (RATES (GENDER, DBF) VAR (BABIES))
; Calculate inbreeding coefficients for new population:
INBREED (
    REPORT (F.RPT)
    VAR (F))
WEAN (RATES (WEAN, DBF))
CENSUS (VAR (POP))
SUMMARY (SYMBOLS (RUN, YEAR
    DEAD, BRED, MATED, LITS, BABIES, F))
END ()
PURGE ()
END ()

Example 8: Simulate inbreeding depression among inbred newborns.

Inbreeding depression is usually defined as a lowering of fitness (often taking the form of decreased neonatal survival) due to the expression of deleterious alleles through inbreeding. GAPPS can simulate this phenomenon by using INBREED and DEPRESS. INBREED is used to calculate each individual's inbreeding coefficient (as in Example 7). DEPRESS is used to modify a rate (in this case, a survival rate) by the function \( e^{-BF} \), where \( B \) is a chosen slope constant, and \( F \) is the individual's inbreeding coefficient. Thus, inbred individuals can be made to have lower survival rates than non-inbred individuals. In this example, we wish to effect a change only in newborns. Rate file DPKS0.DBF has nonsense values for all non-zero age-classes, (i.e. 8.00), such that, even if inbred, their survival rate will be greater than 1.0, meaning that all will survive. Newborns are given a survival rate of 1.0. This way, all newborns with \( F = 0 \) will survive; those with positive \( F \) values will have some probability of surviving which will decrease as \( F \) increases. Here is the Command file:

```
SETUP (
    TITLE ("Inbreeding depression in survival of young")
    BEGIN (RUN (25))
    READ (INPUT (POPIN, DBF))
    ; Allocate arrays for sires and dams of each individual,
```
and calculate F values for initial population;
INBREED()
BEGIN (YEAR (50))
SURVIVE (RATES (SURV1.DBF) VAR (DEAD))
BREED (RATES (BREED.DBF)
        VAR (BRED))
MATE (RATES (MATE.DBF) VAR (MATED))
LITTER (RATES (L1.DBF,L2.DBF,L3.DBF) VAR (LITS))
AGE()
BIRTH (RATES (GENDER.DBF) VAR (BABIES))
;Calculate inbreeding coefficients for new population;
INBREED(
        REPORT (F.RPT)
        VAR (F))
;A special survive routine: all animals other than young
of-the-year always survive. Young-of-the-year survive if F = 0; otherwise, they survive at exp(-F(B)), where
F is the individual's F value and B = 2.0 in this
case.;
SURVIVE(
        RATES (DPRS0.DBF)
        DEPRESS (2.0)
        VAR (INBRDS))
WEAN (RATES (WEAN.DBF))
CENSUS (VAR (POP))
SUMMARY (SYMBOLS (RUN,YEAR
            DEAD,BRED,MATED,LITS,BABIES,INBRDS,F))
END()
PURGE()
END()

Example 9: An animal that lives more than 30 years: Elephants.
GAPPS allows the user to specify different rates for up
to 30 age-classes. However, it is possible to model animals that
live for longer than 30 years. If an animal may live to, say, 35
years, one can simply do nothing but let the survival rates let
this happen. GAPPS does not automatically kill animals that live
to be 31. They will simply stay in the population, counted as
"30-year olds" indefinitely until they die. Another alternative
is to have animals AGE less than every cycle. In this scenario,
each age-class in Rate files refers to more than one actual age.
Here is an example of how it might work:
Example 10: Some animals breed more than once yearly.

In this case, simply repeat the necessary natality commands within each year. Remember, it is not critical that each YEAR be considered a year...it can be any convenient cycle. The user must simply be aware of what the Rates in use will do to the population. For example, if more than one SURVIVE module is used each "year", the Rates used for SURVIVE must not be based on annual survival, because each individual will be exposed to them twice a year. If daily survival rates are known and considered convenient, one can use 365 SURVIVE modules each year, although this is not recommended for people with something in life to do besides watch computer screens. Here is one example of a Command file for an animal with 2 breeding periods each year:

```plaintext
SETUP (  
   TITLE ("Elephants, who live more than 30 years"))  
READ (INPUT(PACHY.DBF))  
BEGIN (YEAR(250))  
   SURVIVE (RATES(SURV.DBF))  
   ;Reproduce and age;  
   BREED (RATES(BREED.DBF))  
   MATE (RATES(MATE.DBF))  
   LITTER (RATES(LIT1.DBF,LIT2.DBF,LIT3.DBF,LIT4.DBF))  
   ;Animals only age every 2 cycles, but go through birth  
   and death process each cycle...this way each "age" is  
   really 2 years, e.g. 1 = 1, 2 = 3, 4, ...30 = 59, 60;  
   AGE (       
      STEP (2)  
   )  
   BIRTH (RATES(FEM50.DBF))  
   WEAN (RATES (WEAN.DBF))  
   CENSUS (VAR (POP))  
   SUMMARY (SYMBOLS(YEAR,POP))  
END ()
```
TITLE ("An animal that breeds twice yearly")
READ (INPUT(POPIN.DBF))
BEGIN (YEAR(20))
SURVIVE (RATES(SURV1.DBF))
; Reproduce and age;
BREED (RATES(BREED.DBF))
MATE (RATES(MATE.DBF))
LITTER (RATES(LIT1.DBF,LIT2.DBF,LIT3.DBF,LIT4.DBF))
BIRTH (RATES(FEM50.DBF))
WEAN (RATES (WEAN.DBF))
SURVIVE (RATES(SURV2.DBF))
; Note: These survival rates must be semi-annual;
; Reproduce and age;
BREED (RATES(BREED.DBF))
MATE (RATES(MATE.DBF))
LITTER (RATES(LIT1.DBF,LIT2.DBF,LIT3.DBF,LIT4.DBF))
AGE ()
BIRTH (RATES(FEM50.DBF))
WEAN (RATES (WEAN.DBF))
SUMMARY (SYMBOLS(YEAR, POP))
CENSUS (VAR (POP))
END ()
COMMON ERRORS

I. Command files

1. Parentheses don't match.
2. Semi-colons don't properly enclose comments.
3. Forgetting to include AGE ().
4. Forgetting to include WEAN ().
5. Rate file missing or improperly specified.
6. Order of events not correct.
7. Forget to include initial INBREED command.
8. (add your own here...)
9. 

II. Rate files

1. Use of INIVAR that is not defined in SET or VAR.
2. Mixup of PARM1 through PARM4 (see Rate files).
3. Mixup of types of animals.
   (add your own here...)
4. 
5. 
6. 

III. Population files

1. Relationships among individuals not logically correct.
   (add your own here)
2. 
3. 
TECHNICAL SPECIFICATIONS, REQUIREMENTS, AND LIMITS

Hardware & Software Requirements

CAPP3 is written in the "C" Programming Language for the IBM PC, XT, AT, and close compatibles. Close compatibles are those using the Intel 8086, 8088 2, 80386, or 80286 cpu chips. The Color Graphics Adapter (CGA) screen memory must begin at memory address B800H, and monochrome display memory at address B000H. The computer must be equipped with a matching Intel 8087/80287 numeric co-processor. A minimum of 512K of installed memory is required, and 640K is highly recommended.

Program Limits

Total Number of Animal Simulated (Live and Dead)
16,378

Total Number of Commands per Command File
256

Total Number of Unique AGESEX, CASES, REPORT, and TRACE Files
256

Total Number of Unique RATES Files
256

Total Number of Variables Declared by SET() or VAR()
254

Animal Age Range
0 - 29

Maximum Variable Name Length
Maximum File Name Length
63

Maximum Animal ID Number
16,377

Maximum Population Tag Number
255

Total Number of RUNs per Session
No Limit

Total Number of YEARs per RUN
No Limit
LITERATURE CITED


