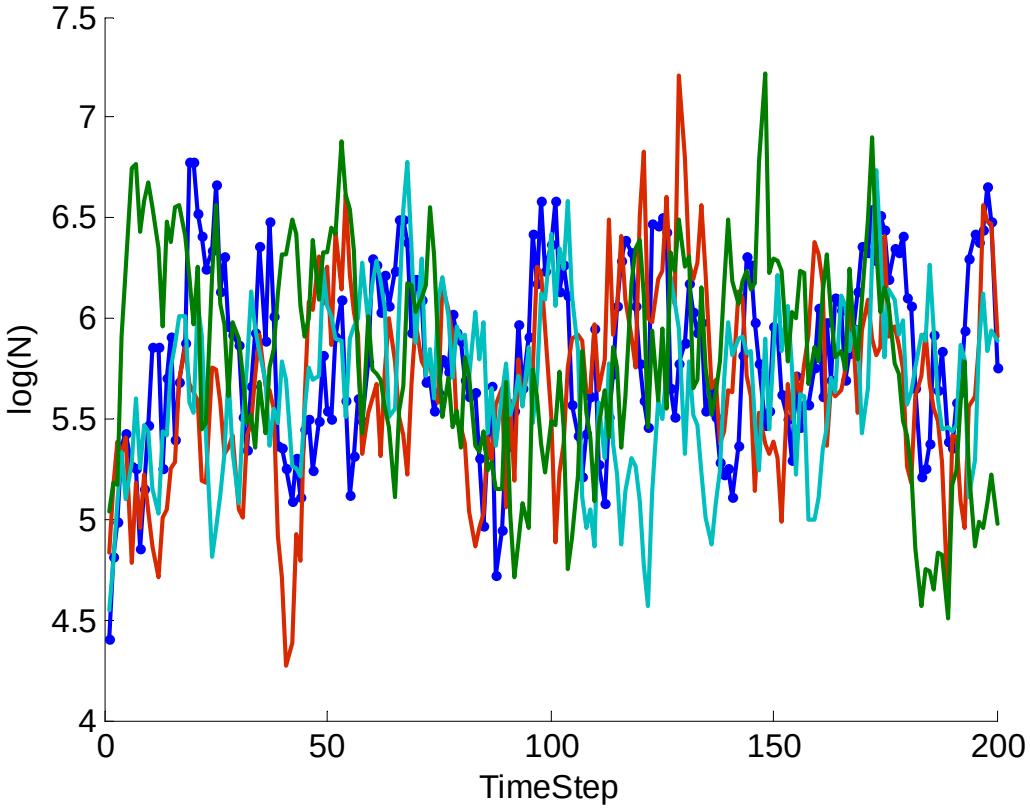
LAMBDA

LONG-TERM ASSEMBLAGE MAR(1)-BASED DATA ANALYSIS



USER GUIDE
BETA VERSION 0.9.2

Steven V. Viscido
NRC Post-doctoral Fellow
Northwest Fisheries Science Center (206)860-2450
2725 Montlake Boulevard East, Seattle, WA 98112 USA
steven.viscido@noaa.gov

CONTENTS

1 INTRODUCTION

1.1 About this document

This document is intended as a reference manual for users of the LAMBDA toolkit. Note that MatLab is required, and some basic familiarity with MatLab (such as how to add paths to your environment, and how to open and execute MatLab files) is assumed. However, once LAMBDA is running on your system (the "LAMBDA" window and logo appear), all functions and steps should be executable using only the graphical user interface (GUI) -- no line commands ought to be necessary.

1.2 System Requirements

To run LAMBDA successfully, the following are minimally required:

- ✓ MatLab version 7.0.1 (R14) w/service Pack 1, or later
- ✓ MatLab's Statistics toolbox
- ✓ At least 256 MB of RAM on your system
- ✓ 5 MB of Hard Drive space for the LAMBDA installation

The memory (RAM) requirements are mostly for MatLab. LAMBDA itself generally takes only 2-3 MB of RAM while running (this includes having a small dataset in memory). However, please note that system requirements may be higher, if you have a very large dataset, or one with lots of variates and covariates. The number of variates dramatically increases how long models will take to run, or bootstrap searches will take to execute, because they are multiplicative. Thus, if a system with two variates takes 4 minutes to be run under LAMBDA, a system of equal length but with 4 variates will take 16 minutes, and one with 8 variates will take 64 minutes. This can be aided by using systems with faster processors and with greater amounts of memory (RAM).

1.3 Terms and Conditions

2 OVERVIEW

2.1 So what is LAMBDA?

LAMBDA stands for **L**ong-term **A**ssemblage **M**AR(1)-**B**ased **D**ata **A**nalysis. It is a MatLab toolkit designed to do MAR-1 based data analysis on long-term datasets (i.e., time series) – see section 2.2 for more details on what a MAR-1 model is. The LAMBDA toolkit is designed so that, once the user starts the program and brings up the user interface, all necessary operations should be able to be performed from the user interface (that is, you should not have to resort to "command line" interactions with MatLab). LAMBDA is designed to work with field data, of the sort one would have from surveys taken in the same area over many months or years, although just about any data in a time series can be used with it. LAMBDA is designed to allow the user to step through the entire modeling process, from importing the data into a format readable by MatLab, to re-scaling the data as necessary, to obtaining descriptive statistics of the dataset, to, finally, performing a MAR-1 regression model and obtaining output parameters. Because LAMBDA was written in the MatLab language, you must have MatLab installed on your computer to use LAMBDA. You will also need to have the Statistics toolbox installed for MatLab, because several of the routines in this toolbox are used by LAMBDA. If you choose certain options, you will also need the Optimization toolbox for MatLab (the options requiring this will be noted below). Finally, throughout this manual I will assume a rudimentary knowledge of MatLab on the part of the user. In other words, I assume you know how to start up MatLab and run a script from the command line or edit window, how to add directories to your MatLab path, and so on. If you do not know how to do these things, I recommend purchasing an introductory MatLab book, and spending an afternoon with it before starting to use LAMBDA.

2.2 So what is a MAR-1 Process?

A MAR-1 process is a Multivariate, Auto-Regressive first (1st) order process. Essentially, it is a means of estimating interactions between multiple variates from time series data, using matrix algebra. Ives et al. (2003) published the mathematics and statistics of how to do this, and the reader is encouraged to read this reference if MAR-1 processes are unfamiliar.

A MAR-1 model is a stochastic, non-mechanistic model that uses time series data to deduce inter-population interactions and the effects of covariates (e.g., physical variables) on populations. Normally, the data exist in a time series for each species or population, and the only things known are the population sizes, such as data obtained by population censuses, or from fishery landing data. If the time series is of sufficient length ("sufficient" being determined from the nature of the data), the strength and direction of interactions between all species and the measured environmental covariates can be estimated in a manner similar to a multivariate regression.

The starting point for a MAR-1 model is the Gompertz equation (Ives et al. 2003), which is an exponential equation describing fluctuations in population size. If we let N_t represent the population density (biomass per square meter, number per hectare, etc), and let *a* represent

the intrinsic rate of increase, and *b* represent density dependence, then the Gompertz equation is

$$N_t = N_{t-1} \exp[a + (b-1)\ln(N_{t-1})] \quad . \tag{1}$$

If we take the natural logarithm of both sides, and let *X* stand for ln(*N*), then the equation simplifies to

$$X_t = a + bX_{t-1} \quad . \tag{2}$$

Notice that eq. (2) is linear, which substantially simplifies the mathematics. This is still a deterministic model. Of course, because we aren't omniscient, things we can't measure always impact natural systems. Therefore, there is always some "error" or unexplained variation in our measurements of *X* (the log of the population size). This is accounted for by means of *E*, the process error, which is assumed be a random normal variate with a mean of zero and variance σ^2, and independent in time, which gives us the stochastic version of eq. (2), namely

$$X_t = a + bX_{t-1} + E_t \quad . \tag{2}$$

This is now a univariate auto-regressive, first order, or AR-1, process. The log of the population size at any time *t* depends on (1) the intrinsic rate of increase *a*, (2) the log of the population size in the previous time step (scaled by the strength of density dependence *b*), and (3) some random process error *E*. One can also add an effect of a covariate by specifying the covariate's measure as *U* (which may or may not be log-transformed, as determined by the nature of the data), and the strength of the interaction between the covariate and the population as *c*. This gives us the equation

$$X_t = a + bX_{t-1} + cU_{t-1} + E_t \quad . \tag{3}$$

For example, *U* might be the log of the mean annual sea surface temperature (SST) for an annual plankton survey, and *c* would then represent how strongly (positively or negatively) SST affects plankton abundance.

The full AR-1 model in eq. (3) is a univariate model. To convert it to a multivariate (MAR-1) model, one uses the same equation, but the variates and covariates are formatted as matrices. If we let **X** stand for the matrix of *X* values for each population, **A** stand for the matrix of *a* values for each population, and so on, we can write the MAR-1 model as follows:

$$\mathbf{X}_t = \mathbf{A} + \mathbf{B}\mathbf{X}_{t-1} + \mathbf{C}\mathbf{U}_{t-1} + \mathbf{E}_t \quad . \tag{4}$$

The key to this model is that it allows one to estimate the interaction strengths reciprocally between all species for which one has population size estimates. This gives us **B**, the interaction matrix. It also allows an estimate of the effects of covariates on the populations

through **C**. Thus, given a time series in which there are two or more species, and any number of covariates, one can estimate how strongly or weakly these components of the system interact with one another. Since it is a regression model, the MAR-1 model will give one coefficients of determination R^2, which will describe how much of the variance is explained by the model. A MAR-1 model can also compute properties of community stability, such as reactivity, return time, and the like. Thus, with just estimates of population sizes over a period of time, and measures of a few covariates, an investigator can deduce how strongly the components of the system interact with each other, and how stable the system is in the face of perturbations.

2.3 Further reading

For further reading, the user is referred to the seminal paper on MAR-1 models:

Ives AR, Dennis B, Cottingham KL, Carpenter SR, 2003. Estimating community stability and ecological interactions from time series data. *Ecological Monographs* **73**:301-330.

2.4 Who would want to use LAMBDA?

The LAMBDA toolbox is primarily aimed at ecologists, fishery biologists, and other scientists who work with time series data. Because LAMBDA is a tool for MAR-1 based analysis (see Section 2.2), and because MAR-1 models are autoregressive models involving time series, the toolkit is not useful for data in other formats (i.e., those not in time series format). I assume the user has basic familiarity with regression, and is sufficiently familiar with MAR-1 models to know what a *variate* and a *covariate* is with regard to his own dataset. My hope is that ecologists and managers will be able to import their data into LAMBDA, and use it to determine the interactions between species in their system, as well as the stability of their system. The goal is to make LAMBDA a useful tool for analyzing long-term data.

2.5 How to use this book

This book is designed as a guide for *users* of LAMBDA (people who use the interface elements to do data analysis on time series). It is not designed as a programmer's guide. Those interested in the nuts-and-bolts of LAMBDA code are referred to the matlab (m) files and their associated comments. Programmers are free to modify the code if they wish, but you do this at your own risk, and the program is not guaranteed to work if you change it.

This book is broken up into sections that are hopefully more-or-less self-sufficient. It is designed to be read through from start to finish, but because of its section-based format, it should also be possible to skip those sections that are of no interest to you, and focus on those that address your immediate needs. There is also a chapter in the back that shows a tutorial of how to use LAMBDA in conjunction with the sample data sets, to perform a simple data analysis, step-by-step. I recommend that the reader peruse the entire book and read a few early chapters, and then move on to the tutorial to see step-by-step how to use the LAMBDA toolbox.

2.6 Acknowledgments

This book, and the development of the LAMBDA toolkit, benefited from the assistance of many people. Eli Holmes served as my primary post-doctoral adviser at the National Oceanic and Atmospheric Administration (NOAA). Funds for my position were provided by a National Research Council (NRC) post-doctoral fellowship. Members of the Conservation Biology (CB) and Fisheries Research and Monitoring (FRAM) divisions provided valuable assistance, advice, and consultation all along the way. In particular, I extend thanks to Stephanie Hampton, Chris Jordan, Mark Sheurell, Steve Katz, Isaac Kaplan, and Chris Harvey. Anthony Ives, who wrote the paper that inspired the development of LAMBDA (Ives et al. 2003) provided his code to me, and helped me understand various parts of it that were confusing to my limited MatLab knowledge.

3 GETTING STARTED

3.1 Obtaining a copy of LAMBDA

LAMBDA is a toolbox for MatLab that is developed and maintained by Steven Viscido, an NRC post-doctoral fellow at the National Marine Fishries Service. It is distributed free of charge under the GNU General Public License (GPL). It can be obtained by e-mailing the author directly (`steven.viscido@noaa.gov`).

The file you receive will be a gun-zip archived file (`tar.gz`) or a zip file (`.zip`), which will need to be uncompressed. On a Windows system, this can be done using WinZip (`http://www.winzip.com`), gunzip (`http://www.gzip.org`) or WinRAR (`http://www.rarlab.com`). For Windows XP/2000 or later, you can also use the default unzipping package included with the Operating System. On a unix system, the gzip/gunzip programs come with most shells these days.

Once you have obtained the program, you are ready to unzip and install it.

3.2 Installing LAMBDA

Once you have obtained the zip archive of the latest version of the LAMBDA toolbox (see Section3.1), simply extract it using whatever unzip program you wish, into whatever directory you wish. The archive will create a `LAMBDA` folder, and all program files and documents will be unpacked into this folder. That's all you need. Once that is done, you might wish to add the `LAMBDA` folder to your MatLab path (see the MatLab documentation for how to do this, or type `help path` in the MatLab command window). After adding the path, you can run LAMBDA whenever you want (see Section 3.3)..

3.3 Running LAMBDA

To run LAMBDA, make sure you are in the `LAMBDA` directory in MatLab. You can check this by typing the command `pwd` at the command line. If you aren't in this directory, you should use the `cd` command to change to it. Once you are in the `LAMBDA` directory, simply type `lambda` at the command line, and the LAMBDA user-interface window will appear (Figure 3.1).

Once this user-interface window appears, you should be able to use its menus and the buttons on subsequent windows to perform all necessary operations within LAMBDA. You should not normally have to execute any further line commands “by hand” at the MatLab command prompt. You are now ready to begin using LAMBDA.

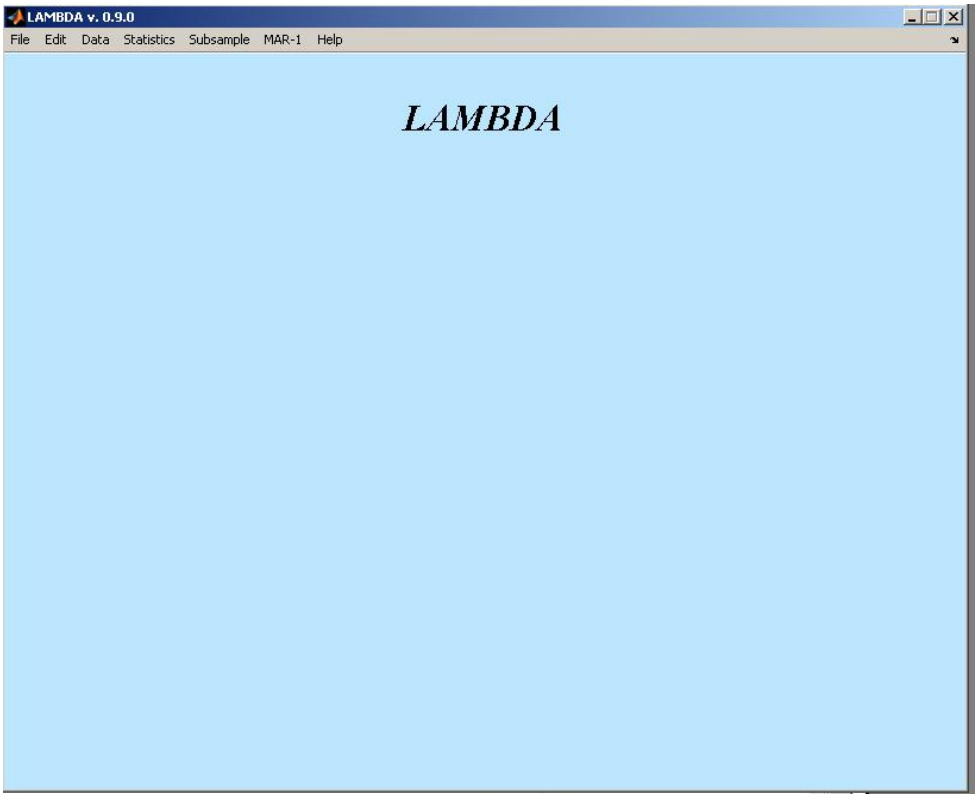


Figure 3.1: The LAMBDA user-interface window on first startup.

4 DATA FILES AND FORMATS

4.1 Data Formats, Variables, and Observations

Perhaps the most important element of any data analysis is the most fundamental: understanding what the variates and covariates are, and the types of data on which you are going to perform your analysis. Therefore, I will begin by defining some terms that will be used regularly throughout this book, so that we are on the same page whenever they come up.

Variables. Each different kind of thing you measure, either in the field, or in a lab, is known as a *variable* (this term should not be confused with *variate*, which I will discuss below). Variables are things such as the air temperature on the day you did your survey, or the number of fish you captured in your trawl net, or the weight of a sample. As the name implies, variables can *vary* in value; they are not always the same. Thus, the air temperature might be warmer one day when you do your sampling, and cooler the next. Each variable normally has a single name associated with it (e.g., temperature, mass, volume, number). The variable is, in other words, *what you are measuring*.

Observation. Each time you make a measurement of a variable, you are making an *observation* of it. So for example, you could go out today and count the number of gophers in your yard, and observe that there are seven gophers. Next week you could go out and again count this same variable (number of gophers), and observe that this time there are nine gophers. Each of these measurements is an *observation* of the variable "number of gophers."

Data sets are ordinarily organized in tabular[1] form. The *variables* will be organized in the *columns* (one column to a variable) and the *observations* of each variable will be listed in the *rows*. This type of organization is called "tabular data storage" or, if you have it on a computer, it is known as a "flatfile." Note: *LAMBDA assumes your data are organized like this in a flatfile. If they are not, you must convert them using some other means, before you can use your data in LAMBDA.*

An example of a "flatfile" dataset is shown below. Note that the variables are: day, air temperature, and number of gophers. Each observation is listed in a given row (so for example, on day 1, there were 4 gophers observed).

```
Day  Temp (C)  No. Gophers
1         17         4
2         15         5
3         16         5
4         16         7
5         18         6
6         17         4
7         20         3
8         14         9
```

[1] Here, "tabular" means "in the form of a table." This is not to be confused with tab-separated data, which LAMBDA will *not* accept. Make sure data are separated by one or more spaces, not tabs. (Any number of spaces is allowable, and the number of spaces may vary from column to column – but avoid using tabs.

To be able to import your data into LAMBDA, they will need to be organized into flatfiles with this type of format. Each *variable* must have its own column, and the *observations* must be listed sequentially row by row. Note that they do not have to be in any particular order; our example data set is sorted by day, but that is not required. LAMBDA will allow you to re-sort the dataset in any way that you want (see Chapter 5).

4.2 Types of Data

LAMBDA is designed to recognize, and expect, several *types* of data. These data types have been divided (arbitrarily) into three categories: *Classification* variables, V*ariates*, and *Covariates*. I will briefly explain what each term means.

Classification variables. Whenever you perform a survey or an experiment, you will have certain variables that you use to categorize what you are doing and how you are taking measurements. For example, you might record the *date* on which a survey was performed (day, week, month, or year, depending on sampling design), the *location* at which the samples were taken, the *sampling apparatus* you used, and other relevant information about your sampling design. This will help you to divide up, subsample, and pool your data (for example, you might decide to look only at samples taken between 1977 and 1981, or only those recorded on Wednesday mornings). Classification variables are used, therefore, to classify, partition, sort, and organize the rest of your dataset. For example, when you record the *date* of your lab experiment, you are using it as an identifier that will help you sort and keep track of your data. LAMBDA assumes that your data file is organized with at least one *classification variables*. That is, one of your columns of data should be a classification variable.

Variates. The variates are usually the counts, biomass estimates, density estimates, or other means of recording the number of individuals in one or more populations. These values will be used by LAMBDA to estimate the **A, B**, and **C** matrices. Because LAMBDA assumes all measures are in the same units, if your variates are not in the same units, *you will need to convert them to a single, standard unit* before you import them into LAMBDA. For example, if you have the biomass of plankton measured in g and the biomass of fish measured in kg, you will need to either divide the former by 1,000 or multiply the latter by 1,000 so they are all scaled the same way. Without doing this, you might have issues when trying to run the MAR-1 model. Your variates may have been recorded either be scaled to the sampling "effort" (e.g., g/m^2), or as raw estimates. If they are raw estimates, then unless your sampling effort (such as the area swept by a net) is exactly identical, you need classification variables (see above) that quantify this "effort" information, and must convert variate data so that each observation comes in the same units. For example, suppose you swept the net twice. The first observation was a 3 m^2 sweep and came up with 100 g of plankton. The second observation was a 2 m^2 sweep that came up with 50 g of plankton. To give these observations the same units, you must divide by area, which converts the first measure to 33 g/m^2 and the second to 25 g/m^2. This need not be done in your input dataset; LAMBDA can do it for you as long as you import your "area swept" information as a classification variable. For more details on scaling your data, see Chapter 6.

Covariates. Covariates are special variables. Their interaction matrix **C** is separate from that of the variates. A *covariate* is one that likely affects the variates, but is not affected by them in turn. In other words, the interaction is a one-way street (if it's a two-way street, then you've got a *variate* instead – see above). An example of this might be sea surface temperature, which would be a covariate in a dataset where plankton densities are the variates (since plankton density is probably affected by sea temperature, but sea temperature is not likely to be affected by plankton density). Covariates will often be on different scales from one another, and this is entirely acceptable, as long as any given covariate is always on the same scale (for example, your temperature estimates should not alternate between Celsius and Kelvin scales). Covariates may, or may not, need to be scaled to the sampling "effort" (area swept, etc), depending on what they are. For example, temperature will usually not depend on sampling effort, and would not need to be scaled, but if you are using the number of phytoplankton per liter of water as a covariate, you will need to scale it by sampling effort. Again, if you provide LAMBDA with classification variables of the sampling effort, it can scale the data for you as needed. *Please note that at this time, LAMBDA can only either scale all covariates, or none, so plan your dataset accordingly.*

The most important thing to remember about your variables is that *you* are the one who must determine which are simple classification variables (that is, variables that help you lump or sort your dataset, but are not active in the system), which are variates (those that mutually act on each other), and which are covariates (those that act on the variates but are not acted on by them). This will depend entirely on context, and on *what you measured*. For example, you might not have measured "both directions" of an interaction between X and Y, which might lead you to consider Y as a covariate, rather than a variate. On the other hand, some things absolutely cannot be variates. For example, the number of gophers in your yard cannot possibly affect sunspot activity; so, although you can count sunspot activity as a covariate acting on the gophers, it would not be reasonable to count it as a variate being equally acted upon by the gophers. In general, you will need to use your biological intuition to determine how to classify variables.

4.3 Importing data

There are three functions for importing data from the file menu. You can (1) import raw (text-file) data, (2) simulate formatted data, or (3) add a new set of data (from a text file) to an existing LAMBDA dataset. This section will cover all three options.

4.3.1 Importing raw (text-file) data

Before you can use LAMBDA to analyze your data, you will need to import your data into LAMBDA so that it is formatted in a way LAMBDA understands. Currently, the only form of raw data that LAMBDA can import is data in *whitespace-separated columns* (one column per variate) stored as text (`txt`) files[2]. LAMBDA searches for files named with a `*.txt` extension, so if your data set is called something else, change the last part of the name using your operating system's file naming tools (or copy it under a new name). LAMBDA will then

[2] It is critical that the columns of data be separated by spaces (white space) and not by tabs. Be careful when doing Excel exports, because Excel clandestinely inserts tabs rather than whitespace in most export formats. It is probably best when using excel to simply do a direct excel import (see section 4.3.2).

be able to import your file. Note that re-naming it does not properly format it; you must make sure your data are in space-separated columns for the import to be successful. An example of what a reasonable LAMBDA dataset would look like is below, with the columns being year, station, plankton biomass, and mean annual temperature.

```
Year  Station  biomass  Temp
1980     1     100.0    20
1980     2     102.3    19
1981     1      99.7    16
1981     2      89.4    17
1982     1     101.3    18
1982     2     101.1    19
1983     1      88.7    16
1983     2      92.3    19
```

Once you are sure you have your data in column format, you are ready to import into LAMBDA. To import a file, use the File menu, and select Import Data and then From text file. This will bring up a window to allow you to select a file. After you choose the file you wish to import, LAMBDA will ask you whether your file contains any "headers". These are words, strings, or other labels at the top of a file that are used to name each of the variables. In our example above, the headers are, in order, "Year," "Station," Biomass," and "Temp." If you have headers and tell LAMBDA so by answering "yes" when it asks, it will then ask if you want to include those or not. If you want to re-name them, answer "no" to the second question. If you have no headers or if you choose to discard existing headers, LAMBDA will automatically name your variables "1", "2", "3," and so on, from left to right.

Note: If you have headers but you tell LAMBDA that you don't, you will get an error, because it will try to treat the first row as numerical values, and they are words. Make sure you check the file before answering this question.

LAMBDA will next ask you to select which variables are *Classification* variables (Figure 4.2). It will bring up a list of all the variables you have imported, and you simply click the ones you wish it to treat as Classification variables, and then choose "OK." If you have several of these, and they are adjacent, you can use shift-click to highlight them all. If they are not adjacent, such as at the top and bottom of the list, hold down the control key, and click (this is called "ctrl-click").

After choosing Classification variables, you will then be asked to select which of the remaining (not yet assigned) data columns are your variates, and then the rest will be assigned as your Covariates. At this point, your data have been fully imported into LAMBDA, and you can then begin the process of manipulating, editing, and otherwise analyzing your dataset.

Please note that importing data does not save it permanently to a file. *Your data will be stored temporarily in a folder within the LAMBDA directory only for the duration of the LAMBDA session.* As soon as you close LAMBDA, the temporary data are erased. If you wish your dataset to be saved in a more permanent fashion, use the "Save Data As…" function from the file menu (see section 4.4).

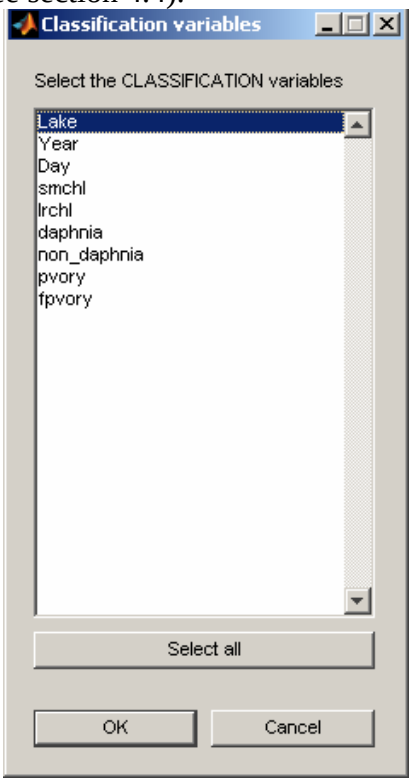


Figure 4.2: Specifying how many columns you have of each data type.

4.3.2 Importing an EXCEL file

LAMBDA can recognize and import Microsoft Excel® files (*.xls). As in the examples above, your data must be stored in columns, with one column per variable. If you have one row of "header" information, that row will be recognized and imported into LAMBDA (similar to the example above with text files). LAMBDA will assign a number to each column at the time of import if you do not have header information in the file. LAMBDA does not accept more than one row of header information, and multiple header rows will produce an error.

4.3.3 Adding a replicate to an existing dataset

If you have a LAMBDA formatted dataset already in memory, and you want to add the data from another text file, you can do this by choosing the Add a rep to an existing dataset option. You can add a text file to the existing dataset only if it has the exact same number of variates, covariates, and time steps as those of the dataset already in memory. For example, you cannot add a 100-time step series to a dataset that only has 20 time steps in it. If the

dimensions of the dataset are the same (number of columns and rows), then the import process will add the data as a new replicate (a new “page” in the MatLab array).

4.4 Loading and saving formatted data

Once your data set has been imported, it is now in “LAMBDA format.” This means it is stored in special data objects, and the information is only accessible through the LAMBDA interface or its functions. The data will only persist as long as LAMBDA is running, so if you wish to save your data for future work in LAMBDA, you should save your data in LAMBDA format. LAMBDA dataset files are saved with a `*.dst` extension. To save your LAMBDA-formatted data to a file, select the File menu, and choose the second option, Save as.... Give the file a name with a `*.dst` ending e.g., `mydatafile.dst`), and choose OK to save it as a file. To load your saved LAMBDA data file, select the File menu and choose Load formatted data. This will let you peruse your folders for any `*.dst` files, and you can then select one. Choose OK to load the file. The data, in LAMBDA format, will now be loaded into memory, and you can perform all normal operations on it.

I recommend that, as soon as you import your data correctly the first time, you immediately save it to a `*.dst` file, so that it isn't necessary to keep importing raw data. Also, any time you perform an operation on a dataset, save it under a separate name, to reduce the need for repeatedly performing the same operation. For example, suppose you import your dataset, and then decide to delete two columns of data. Save the new, smaller dataset under a different name, so that you can use it the next time and do not have to once again import data and delete columns.

4.5 Exporting data to other applications

At some point after working with LAMBDA, you may wish to export the datasets that have been formatted for LAMBDA, back out for use with another package. Since the data are stored on LAMBDA data objects, you will need to export them in a more standard format. This option is available in the File menu. Choose Export Data, and the dataset in memory will be exported to a comma-sparated (`*.csv`) text file. This file will then be readable by any software that can import such files (e.g., Excel, SAS, MatLab).

5 EDITING AND MODIFYING DATA

Once your dataset has been loaded into LAMBDA's memory (which, by definition, causes it to be converted to a LAMBDA-formatted dataset), you will be able to edit and modify it in certain ways. These modifications include loading column labels from a file, editing column labels by hand, and adding or deleting columns from the dataset.

5.1 Loading column labels from a file

You may want to rapidly change all the labels of your dataset at once. For example, suppose we have a file that includes the year and sample site, the biomass of crabs, and the temperature (as a covariate) at the time of sampling, but that these columns did not have headers in the imported file, so it looked like this:

```
1980     1     100.0    20
1980     2     102.3    19
1981     1      99.7    16
1981     2      89.4    17
1982     1     101.3    18
1982     2     101.1    19
1983     1      88.7    16
1983     2      92.3    19
```

Having imported this file, the first column, which contains the year, would be labeled by LAMBDA as "1;" the second column, which contains the sample site number, would be labeled "2," and so on. These labels may be edited by hand (see Section 5.2), but if there are a lot of them, this may be inconvenient, as they would need to be done one at a time. Instead, you can import a single text file that contains the labels. The format of this file is very specific, however: one label must be present on each line, and the labels must be in the same order as the columns were in. Thus, in our example, first "year," then "site," and so on, and finally, there can only be letters, numerals, or underscores (_) in the label names. For example, we could set up a label file for our 4-column dataset that looks like this:

```
Year
Site_Number
Crab_biomass
Temp
```

Label files should be in text format, and should have the extension `*.txt` at the end (LAMBDA searches for `*.txt` files by default). To import a set of labels from such a file, open the Edit menu and select Load column labels. This will import your labels and label your columns as appropriate.

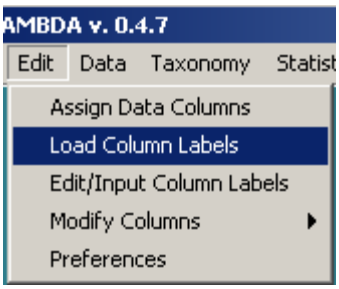


Figure 5.1: Importing labels from a file using the Edit menu.

Note that, if after loading the file you wish to change any of the labels you loaded, this can be done by using the Edit/Input Column Labels feature from the Edit menu (see Section 5.2).

5.2 Editing column labels by hand

The columns you have imported into LAMBDA come with default labels equal to their column numbers (that is, "1" is the label for the first column imported, "2" for the second, and so on, going from left to right in the import file). If you wish to change these labels, or at any point wish to edit the labels of your columns, this can be done by opening the Edit menu and selecting Edit/Input column labels from the list. After choosing this option, a new editing window will appear.

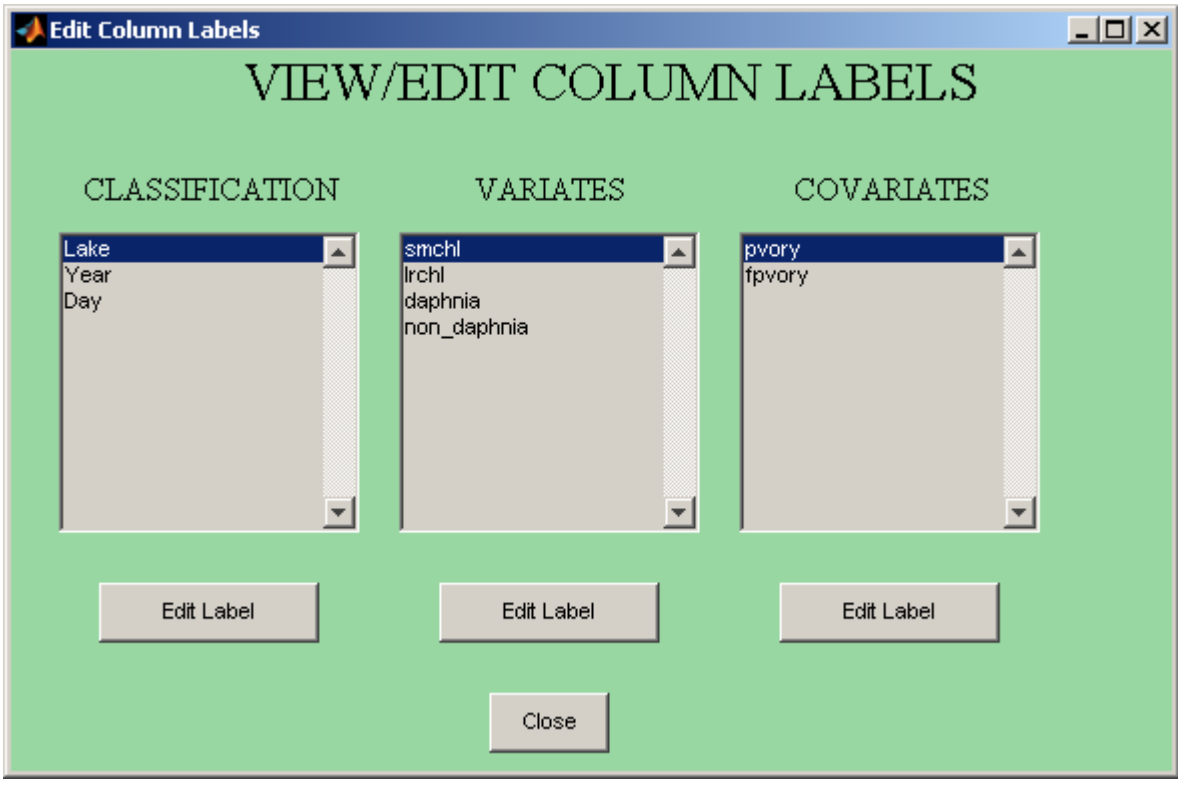


Figure 5.2: Editing labels by hand.

This window presents the dataset in three boxes, one for each type of variable (see Chapter 4). To edit a column label, simply click the label with the mouse, and then select Edit Label at the bottom of the box. Note that each variable type has its Edit Label button, so make sure you press the correct one for the variable label you wish to edit. This will bring up a new box with the old variable name already present, and ask you to type in a new one. If you make a mistake, you can exit this smaller window by choosing Cancel. Please note that once you choose OK from this window, the labels will be changed. If you mistakenly change the labels, you can always edit them again. When you are done editing column labels, you can exit this window by pressing the "close" button.

5.3 Adding and deleting columns

If you wish to add more columns of data (remember: each column represents the data for one variable) to the dataset in memory, or if you wish to delete existing columns, you can do so by means of the Edit menu's Modify Columns option. This will bring up a secondary menu with two choices: Delete, and Add.

Deleting columns. If you wish to delete a column of data, you can do this by choosing the Delete option from the Edit - Modify Columns menus. After choosing Delete, a new window will pop up, listing the column labels for each column in the dataset. You can select one or more of these columns to delete. To select and delete a single column, simply click on that column's name, and select OK. If you wish to delete multiple columns, you can do so by holding down the `ctrl` button and clicking on the name of each variable you want to delete from your data.

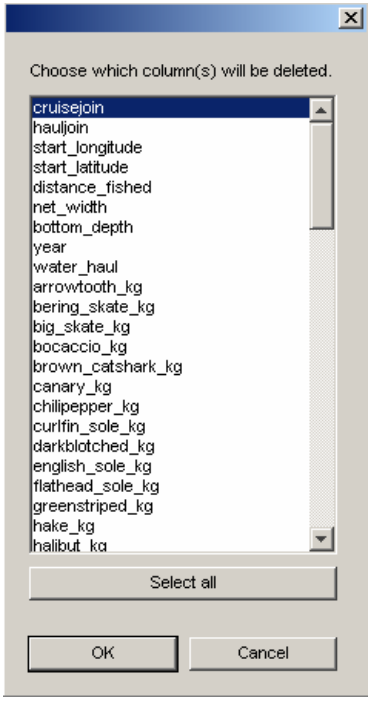


Figure 5.3: Deleting columns of data.

Adding Columns. You may add columns (that is, variables, since each column represents a single variable) to your dataset at any time. The key to remember when you wish to add data is that *only one type of variable* may be added at a time. For example, you may add *Classification* variables, or *Covariates*, but not both at once. If you wish to add more than one type of variables, simply perform the Add operation more than once (each set of variables must be stored in a different file). You may add multiple columns at once, though, as long as they are the same *type* of variate. As with regular file imports, your variables must each be in columns, in a space-separated text file. Unlike regular imports, files with data to add as columns *must* be entirely numeric (no characters or strange typographical symbols allowed). For example, you might wish to add two *covariates* to your dataset: air temperature and wind speed. When you choose Add from the Edit - Modify Columns menus, a window will pop up allowing you to select the file you wish to import. After you choose the file, another window will pop up asking which type of data you are importing (giving you the three variable types to choose from). Select one and press OK to import your file. LAMBDA will tell you how many columns of data have been imported. If you change your mind or make a mistake, simply press Cancel instead. Also, note that new columns come in with numerical labels – you will need to edit them by hand to change these defaults.

6 VIEWING AND SIMULATING DATA

The Data menu allows you to perform two operations; one is to view, plot, or sort any dataset in memory, and the other is to generate simulated data. The simulation can be used to make sure your MAR-1 model correctly estimates known parameters, or to see how different types of parameter sets affect community structure and stability.

6.1Viewing and sorting data

Once your data are imported into LAMBDA, properly formatted, and labeled, it's a good idea to view the dataset, to make sure everything looks correct. To do this, use the Data menu and select View and then choose which variables to view (or "all"). This will provide you with a table of all the columns, and up to 1,000 rows of data. If your dataset is more than 1,000 rows, only the first 1,000 will be shown in the preview. Note, this *does not affect the data in memory* – the observations from 1,001 and up are still present; they simply are not displayed. The reason for this cutoff is that longer tables can cause crashes in MatLab. Once you have verified that the data look as you expected, you can clear this table from the main window by using the Hide data preview option from the Data menu.

If your data are not sorted in the order you want, rows can be sorted by using the Data menu's Sort dataset option. When you select this option a window will appear with three pull-down menus. You may sort by one, two, or three different variables, as desired. Once you have made a selection in the pull-down menu(s), you can sort the dataset. Choose OK to sort the dataset by the variables you have chosen, or Cancel to cancel the operation.

6.2 Merging replicates

If you have multiple replicates, they are stored in "pages" (the 3rd dimension) of the MatLab data array. This is useful for when you are analyzing data, as it keeps the replicates separate. However, when exporting to a text file, or performing certain other operations, you will want your data to all be in a single flat 2-dimensional array (with only rows and columns, but no "pages"). You can do this by selecting the Merge replicates function from the Data menu. LAMBDA will ask if you want to create a new variable to store the replicate number. If you already have a variable that stores this information (such as the "lake number" in the data from Ives et al. 2003), you will not need this functionality. However, if you do not have an identifier for replicate built into your dataset, it is a good idea to use this feature. You can always delete the extra column later if you find you do not need it.

Please note that multi-page data (stored as within-LAMBDA replicates) *cannot be written to a text file,* so if you intend to export, you must *merge replicates* first. When saving files in LAMBDA format, it is perfectly acceptable to have multiple replicates, since LAMBDA can internally keep track of them for you.

6.3 Simulating data

The LAMBDA toolkit can perform simulations to generate MAR-1 type data. To bring up the simulation window, select the Data -> Simulate MAR-1 process option. From here, you will see the SimMar interface (Figure 6.1), which will allow you to simulate your own MAR-

1 process. The window is organized into several frames, and the procedure for conducting simulations is to go from left to right, top to bottom.

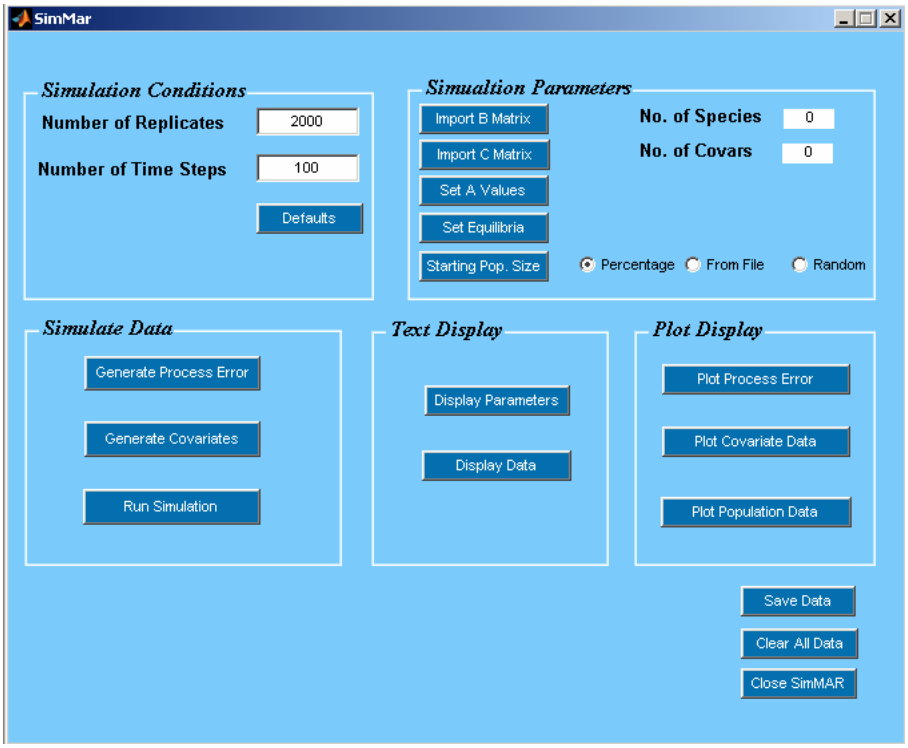


Figure 6.1: The SimMAR interface window

Thus, you begin first by setting the Simulation Conditions (top left frame), then the Simulation Parameters (top right frame). Then you simulate the data, after which you can display it by text or in a graphical plot. Finally, you can save or clear the data. Below, each of the steps will be explained.

6.3.1 Simulation conditions

There are two simulations conditions that must be set before running simulations: the number of replicates you wish to perform, and the number of time steps per replicate. These can be set to any number desired but default to 2,000 replicates and 100 time steps. Simply type the values you wish into the boxes and SimMAR will do the rest. If you want to restore the default values at any time, just click the Defaults button.

6.3.2 Simulation Parameters

Once you have chosen how many replicates and time steps you wish, you must next set up the simulation parameters. These are the **A**, **B**, and **C** matrices, as well as the equilibrium conditions. Each of these must be stored in a text file. Begin with the top button, which lets you import a **B** matrix from a text file, and go downward. All buttons must be pressed except the Set A Values and Set Equilibria button. Because **B** is set, and the equilibrium depends

on **A** and **B**, you can either set the equilibrium, and let SimMAR derive what the **A** must have been (given the **B** matrix you have already provided) to get that equilibrium, or you can set the A, and have SimMAR figure out what the equilibrium will be given that and the **B** matrix. Do not do both… if you do both, whichever you do second will over-ride the previous step. For example, if you set the **A** matrix, and then set the equilibrium conditions, **A** will be changed to match the equilibrium and B matrices, over-writing your imported **A** matrix.

The format of the text files is very specific. The **B** matrix must be a plain text file formatted as a square table with the number of rows and columns equal to how many variates you have. Below is an example of a valid **B** matrix.

```
0.5     -0.3     0.2    0.0
0.6     -0.1    -0.8    0.0
0.0      0.4    -0.3   -0.6
0.0      0.0     0.7   -0.2
```

The **C** matrix must always be included, even if you don't want to include any influential covariates in your model. It must also be a plain text file, with each column representing the influence of a covariate, and each row representing each variate. Thus, the **C** matrix must have the same number of rows as you have variates (and, therefore, as the **B** matrix). An example of a two-covariate system in a four-species community is below:

```
0.4  -0.7
0.6  -0.2
0.1   0.5
0.0   0.0
```

If you don't want any covariates to affect your model, make a text file with a single column of zeros, like so:

```
0
0
0
0.
```

This will cause SimMAR to generate covariate data, but they will have no influence on the system.

The **A** matrix, if you have one, must be a plain text file with the **A** values in a single column (**A** is a column vector). The number of rows in this file must match the number of rows of the **B** matrix, for example:

```
5.4346
7.0440
8.0790
1.1760
```

This step is optional. If you don't choose an **A** matrix, however, you then must choose the other option, and provide SimMAR with the equilibrium densities of each species. Like the **A** matrix, this must be a text file in a single column, with one row per species, for example:

```
10000
1000
500
100
```

These equilibrium densities can be in either log (base e) or linear format. SimMAR will ask you which format you are using and will convert as appropriate.

Again remember, if you set the equilibrium density yourself, SimMAR will derive **A** from those values. If you set **A**, it will derive the equilibrium densities. Therefore, you should only do one of these two things, not both.

The final step to setting up the parameters is to decide what the starting population densities are for the populations. There are three radio buttons here, where you can choose how to set the starting values. The first is "percentage." If you select this option (which is the default), you will be asked for a percentage of the equilibrium value at which to start the population sizes. You can choose this value separately for each species, and you can set a standard deviation for it as well. Your starting population sizes will be the percent you specify in each replicate, with variance based on the standard deviation you set. For example, suppose you want all your populations to start at 33% of equilibrium with 10% standard deviation. You would set these two numbers when the popup menu asks you for these values.

You can also choose to have the starting populations be completely random by selecting this radio button. Completely random populations can start at any non-zero number, including being way above equilibrium.

Finally, you can choose "from file," in which case SimMAR will then ask for a text file containing the starting population sizes. You can set these to anything you wish, in either log (base e) or linear scales. SimMAR will ask you for the scale when you import the file. It will also ask you for a standard deviation, which is measured as a percentage of the equilibrium you set. For example, if Species 1 has an equilibrium population size of 10 individuals/ha and you want a std of 1 individual/ha, you would set the std in that case to 10% (10% of 10 individuals/ha is 1 individual/ha).

If you want all the populations in each replicate to start at exactly the same size, set the standard deviation to zero.

6.3.3 Simulate Data

There are three steps to data simulation. To make the computations proceed quickly, SimMAR generates the entire matrix of process errors for all replicates in one step. This is

because MatLab is faster at generating a large random matrix, that at generating a single random value each time through an iterative loop – therefore, it is easier to pull out a value from a large matrix, than to generate a new random value in each step. Your first step in simulation, therefore, is to press the Generate Process Error button. This will bring up a dialog box asking what standard deviation you wish (the mean process error is always 0). In normal MAR-1 models, this value will usually be set to 1, but you can make it anything you wish. The higher the value, the more random noise will be present in your dataset.

The next step is to generate the covariate values. Again, remember that this must always be done, even if you don't really want a covariate in your dataset. Whether a covariate influences the system depends on the **C** matrix, so if you do not wish to have influential covariates, simply import a **C** matrix of all zero values (see above). When you click the Generate Covariates button, a new window for setting up the covariates will appear (Figure 6.2).

Figure 6.2: The covariate setup window.

This allows you to generate any number of covariates of five different process types – Linear, Cycling, Exponential Decay, Michaelis-Menten, and Logistic. You will need to set up each covariate, and you must have a number of covariates that matches the number of columns imported in the **C** matrix import step (if you have one column there, you can only set up one covariate here, and so on). The choice of covariate process is up to you – you can pick whichever one is appropriate to the system you are simulating.

To choose one or more covariates of a given type, simply type the number of covariates of that type in the "No. of Covars" box. For example, if you wanted two Linear and one Logistic covariate, you would type a "2" in the top text box, and a "1" in the bottom one (see Figure 6.2). Having chosen the number of covariates, you must now press the "setup" button,

and set up the various parameters necessary to generate the covariate. These values will differ with each covariate, and should be self-explanatory. For example, the linear model will need to know the initial value (i.e., the intercept) and the mean change per time step (i.e., the slope). The logistic model needs to know the initial value (N_0), the rate of increase (r), and the carrying capacity (K). You may name each covariate whatever you wish. By default they are labeled by their type and a numeral, such as "Linear1," "Linear2," and so forth. If you wish to see how your parameters will affect the pattern of the covariate, simply press the "preview" button, and you will see a sample of 10 replicates of data for that covariate. This should help you choose a function that looks the way you want for your covariates.

Once you have set up all the covariates (again, remember that the number of covariates set up must equal the number you requested in the original SimMAR window), you can now generate them. To do this, press the Generate Covar Data button. Ordinarily, covariates will automatically be converted to the logarithmic (base e) scale. However, if you have chosen parameters for your covariates that cause negative numbers to be generated, the covariates will not be transformed. You have two choices in this case – you can either pass the un-transformed (linear) values to LAMBDA for analysis (that is, keep them as is), or you can modify your parameter values, and re-generate data until this does not happen. SimMAR will warn you when negative values are present.

Having generated the covariate data matrix, you can plot it with the Plot Covar Data button. If you are unhappy with the covariate data for any reason, you can clear it and start over. You can also save the covariate data to a file for later use (for example, if you want to simulate many different models with the same covariate data). Finally, you can use the Close Window button to end your session. Any covariate data generated will now be passed to SimMAR in preparation for the final model simulation.

Once you have generated process errors and covariates, you are now ready to run the simulation model. This will use the starting population sizes, and the **A**, **B**, and **C** matrices you have set, along with the covariate data you have generated, to produce a simulated time series of population sizes for your community. Once produced, this dataset is fundamentally indistinguishable from any other set of data you might import into LAMBDA. That is, LAMBDA will not "know" in any way that it is simulated, and will treat the dataset normally. This is very useful if you need to conduct simulation tests of whether your field data could be properly analyzed by LAMBDA.

6.3.4 Text and Plot Display

You can use the Display Parameters button at any time to display what the simulation parameters were. You can use the Display Data button to show the data. Note, this will show you all the data for a given replicate of a given type (e.g., variate data), and if you have a lot of data this could take a long time to display. You can also display the data visually, by using the Plotting buttons. The process error, the covariates, and the variates can all be graphed so you can see what the simulation results were in a visual way.

6.3.5 Saving Simulated Data

Once you have simulated the model, you can save the data to a file. Since the data are stored in LAMBDA format, this allows you to call up the data in later sessions of LAMBDA without having to re-do the simulation. When you select the close button, you will be asked if you wish to retain the data for further use in LAMBDA. If you say no, all the data in memory will be lost, so make sure you have saved your data if you want to keep it, or answer "yes" and use the LAMBDA file menu to save the data (or other features to manipulate it). Typically, after simulating the data, one would choose "yes" to have data passed to LAMBDA, and then conduct regular MAR-1 operations on the simulated data.

The main purpose of this simulation feature is to allow the user to do two things: (1) you can see how MAR-1 processes work under different sets of parameters, and (2) you can verify that the CLS or ML estimates of the MAR-1 parameters (**A**, **B**, and **C**) are correct, by comparing the MAR-1 estimates with what you input for your simulation. Most significantly, if your MAR-1 estimates are not very similar to your simulation parameters, *something is going on* and you should not proceed to analyze real data until you have determined what is causing the discrepancy. Note, you should not expect your numbers to agree *exactly*, but they should be reasonably similar; if you input **A** = [0,0] and the MAR-1 estimated values are **A** = [-10,+8], you know there is a problem. On the other hand, if with the same input values for the simulation, the MAR-1 estimate was **A** = [-0.1,0.2], that would probably be within reason (you can use estimates of confidence intervals to see if these are statistically significant differences). How close you can expect the MAR-1 model to get with a simulation depends on the length of the time series, as well: a 100 year time series will provide better estimates than a 5-year series. You should also check the coefficients of determination (R^2), and make sure they are reasonably good. Very low R^2 values indicate something may be wrong with the simulation, or with the analysis.

6.4 Plotting data

You can plot the data in memory using the Plot Data feature on the Data menu. When you execute this command, it will provide you with a list of survey variables. Choose the one you wish to plot your data against. You will then be able to select whether to plot biomass, count, or covariate data against this survey variable. Finally, you can choose whether to view the data on a logarithmic (base e) or linear scale. For example, you might choose to plot year against biomass on a log scale, or to plot site against covariate values on a linear scale. A separate plot will be generated (in its own window) for each variable in the dataset of the type you have chosen. For instance, if your dataset includes the biomass of beavers, mosquitoes, and ferrets, you will see three plot windows (one for each species).

6.5 Keeping a record of your manipulations

It is always important to know, for a given dataset, exactly what you did to the raw data before an analysis. If you get strange results, for example, you need to be able to answer the question, "Did I forget to log-transform the data?" LAMBDA provides this feature for most data manipulations. To see what LAMBDA has recorded in terms of your data manipulations, choose Show Data Manipulations from the Data menu. This information is always stored along with the data in a LAMBA dataset. This is helpful if, months after an analysis, you can't remember what you did. Simply load the LAMBDA file, and view the data

manipulations, and you will immediately know if rows were deleted, columns edited, and so on.

7 SCALING, POOLING, AND DESCRIPTIVE STATISTICS

Often, your raw data will not be in a form where it can be analyzed directly. You may wish to scale it, pool it, or take means or other such statistics before running your MAR-1 model. For instance, suppose you have a variate with the number of brittlestars captured in multiple samples with a trawl net. Each time you sampled, the net would've been dragged for a slightly different distance along the bottom. Assuming you have a column of data describing this differential sampling effort, you would need to scale your sample estimates by the area swept, to get a density estimate. Otherwise, you could be comparing the number of brittlestars captured in a 50 m^2 sweep with the number captured in a 5 m^2 sweep – which is clearly inappropriate. Additionally, you may wish to pool your data in various ways; for instance, you might want to pool all species in a given family into a single Family-level estimate. You may also wish to transform the data, particularly since the MAR-1 process is designed to work with log-transformed data. And finally, you may wish to take averages or other simple statistics across groups of samples, such as the mean biomass per m^2 in a region. The Statistics menu can help you with all of these steps.

7.1 Scaling the data

If your data were collected using potentially different amounts of sampling "effort" (e.g., different tow distances, different core volumes), you will need to scale each variate as appropriate. This will leave you with a "density" estimate, with units depending on your sample units, but usually something like number/m^2 or g/L. To convert raw counts or biomass to density measures, open the Statistics menu and hold the mouse over the Density option. This will bring up a secondary menu with three options. The first will allow you to compute or otherwise tell LAMBDA what the sampling effort or area was, and the other two perform the conversion.

7.1.1 Computing sampling effort.

Before you can convert your data, you will need to tell LAMBDA which *Classification* variables contain the sampling effort information. To do this, open the Statistics->Density menu and choose Compute Sampling Area/Effort. This will bring up a new window with four options for how "effort" (or area) is represented in your dataset. The first option is to simply tell LAMBDA that the data are already in density format. This is equivalent to canceling the operation – it will make no change in the dataset. The second option is for when you have the area (or other representation of sampling effort, such as volume) in a single variable. For example, you might have volume, in L, recorded in one column of data called `Liters`. If so, you would choose this option. The third option is for people with length and width of sampling effort in two different variates. This is very common with trawl-type surveys, where the width of the net is known (and constant) and the length varies from tow to tow.

Once you have selected your option by clicking on the circular radio button, press OK and you will be shown a new window. If your effort or area information is in a single column of data, LAMBDA will ask which column of data contains the information. Choose the column from the pull-down menu, and select OK. *No re-scaling will be done yet.* If your effort data are in two columns, the new window will ask you to specify which two columns contain the

data, and also ask you for the units of each one. The pull-down menu for units will allow you to choose cm, m, or km. If your data are in other units (such as inches) you will need to convert them. Below this is an area where you can choose the output units. Make sure you specify units of your variables *and* the output correctly, or you may end up with numbers that are surprising (for example, if you put in information in m^2 and get the output in ha, and weren't expecting it, your densities might surprise you). Again, *no re-scaling will be done yet*. All you are doing here is telling LAMBDA what columns contain your sampling effort.

7.1.2 Scaling the variates or covariates.

Once you have computed your sampling effort (area, volume, etc.), you can convert your data to be in the units of effort (such as m^2) that you have given LAMBDA in the previous step. Open the Statistics->Density menu and choose Convert variates to Density estimates or Convert covariates to sampling effort as appropriate. Please note that, at this time, LAMBDA either converts *all* or none – you cannot ask it to convert only some variates. In many cases covariates will not need to be scaled; temperature, for example, is not measured based on density. You will need to use your judgment as to when you need to convert data, and when you do not. Once you select one of these options, LAMBDA will warn you that you are about to convert your dataset, and give you the chance to cancel the operation. After you select OK, the dataset in memory will be converted by dividing each observation by the corresponding effort (e.g., dividing number captured by m^2). This cannot be undone, so you should *save your dataset* before performing this operation.

Once you have completed the re-scaling, your observations should all be in the same units, and you will be able to proceed with other operations (such as taking means, or log transforming).

7.2 Pooling the data

There are two ways to pool, or group, your data: by row, or by column. In each case, pooling amounts to *summing across all observations* within the pool. For example, suppose you have two columns, fish biomass and plankton biomass, and you pool them to get total biomass. An example of how the pooling would be done is listed below.

```
fish        plankton      pooled total
4.2            3.0           7.2
2.4            2.2           4.6
5.8            1.1           6.9
1.1            0.3           1.4
```

Because the pooling returns a *simple sum* of the variables pooled, you must make sure your data are scaled and in the appropriate units before hand. Otherwise, you could be adding 4.2 g of something to 100 mg of something and end up with 104.2 (clearly this is incorrect).

7.2.1 Pooling by column

You may wish to add up several of the columns in your dataset. To sum columns and merge them, choose Statistics->Pool the data and select by Columns. A new window will pop

up asking if you want to pool Variate or Covariate data. You must select one; if you wish to pool data in more than one of these category types, simply perform the operation again and choose another. After you click the radio button (circle) for which data type you wish to pool, select OK and a new window will pop up. This will list all the variables in that category (e.g., all the variates). You can select any of them, or all of them, to pool. Choose individual columns to pool by holding down the ctrl key as you click, which allows you to "cherry pick" each column you want. You can also select all of them using the button on the bottom with that name, if you wish.

When you have completed your selection, choose OK, and LAMBDA will sum up the observations in those columns, creating a new variable to store that sum. The new column will be named `mrg_<name>`, where `<name>` is the name of the very first variate you selected (topmost in the list box). Unlike row-based pooling, your old data columns will not be deleted or changed using this operation – it will simply create a new column.

7.2.2 Pooling by rows

The other way to pool a data is by row. Normally a row of data corresponds to things that were all sampled at the same time, in the same place, etc. For example, consider the following dataset, where the year and sampling station are two of the survey variates, and the number of birds and fish counted are variates.

Year	Station	birds	fish
1977	1	10	100
1977	2	11	90
1977	3	9	27
1978	1	30	182
1978	2	22	88
1978	3	11	144
1979	1	12	25
1979	2	44	41
1979	3	33	69

All the data from station 1 in year 1977 would be placed on the same row, so that in 1977 at station 1, you counted 10 birds and 100 fish. You might only care about annual differences, and not which station you sampled at. In this case, you could pool by rows, and all the values for birds would be added up for each year. For instance, in 1977 there were 30 total birds and 217 total fish counted. Pooling by rows will add up the values in each column separately (fish and birds will be kept separate, in this example). Note that this is a *simple summation*; if you want something more complicated such as means, modes, or medians, use the Compute Descriptive Statistics operation.

When you choose the By Rows operation, a new window will pop up. The window includes a drop-down menu where you can select which column of data you wish to pool by. It will also ask what the minimum and maximum values are (values outside of this will be

eliminated from the dataset), and, most importantly, the *increment*. In our example, if we wished to pool by year, we would set the minimum to 1977, the maximum to 1979, and the increment to 1. If we'd set the increment to 3 or higher, then all the years would've been lumped together (since we only have 3 years of data). If we set the increment to < 1, then some rows would be generated with zeros, since there is no "year 1977.5" value in the dataset. Make sure you carefully examine your dataset before merging by rows. It is often better to *subsample* the dataset and take descriptive statistics, than to pool by rows.

7.3 Transforming the data

The standard transformation algorithms in biostatistics are provided for your convenience. Please note that usually, for MAR-1 models, *data should be transformed by the natural logarithm*. Additionally, one may desire to transform the log-transformed variables further into a z-score (Hampton and Schindler 2006). The other transformations are available for your convenience, but are not usually appropriate for MAR-1 models. The options available are taking the logarithm (base 10 or base e), taking the square root, or taking the arcsine-square root, and taking the z-score. You can select any of these from the Statistics menu by choosing the Transform Data option. Again, the primary one you should use for MAR-1 model analysis is the natural, or base e, logarithm. After you make your selection, a new window will appear asking which type of variable you wish to transform. Note that *this step will change the dataset in memory*, so saving your data before transforming it is highly recommended. You should also pool data, and perform any other such operations *before* transforming it. However, descriptive statistics can be computed afterwards, as well as before (depending on your needs).

7.4 Computing descriptive statistics

The descriptive statistics option will allow you to compute simple descriptive statistical moments such as mean, standard deviation, coefficient of variation, and the like, for your data. You can classify the data using up to five classification variables, and you can have the variables incremented however you like. For example, you might want to know the mean number of organisms captured per decade, in which case, you would select "Year" as the first variable, and then set the year increment to "10." This will cause LAMBDA to compute statistics for each 10-year period. Note that if you ask for increments in which there are elements with no data, LAMBDA will substitute zeros for all places where there is no data. This can be most easily seen by cases where *N* (sample size) is listed as zero.

Once statistical moments have been computed, they can be displayed individually, or all at once. This is accomplished with the Display Statistics menu option. Note that once the display window is up, a menu will appear in its upper left corner, also called Display Statistics, which will allow you to change which statistical moments are displayed.

From the File menu, statistical analysis can be saved to a file, loaded from a file, or exported to a comma-separated text file. Note that LAMBDA-formatted statistics files are stored with a `*.zst` filename extension.

7.5 Plotting Statistics

Statistical moments can be plotted as bar, scatter, or line plots, in 2D or 3D. A plot in 2D normally consists of a display of a single statistic, such as the mean, for all the increments of a variable (for example, year or decade). The user may choose to display error bars taken from any column (most commonly, these are taken from the Standard Deviation or Standard Error column). If you have more than one variate (for example, several species in different columns) a separate plot will be made for each one. Also, note that 3D plots only work with exactly (no more, no less) 2 class variables. These are the class variables chosen when the statistics are computed (you can have as many as you want in the raw dataset). For example, suppose you had a dataset with Year, Station, and Altitude as your classification (Survey) variables. You could only have a 3D plot made of your statistics if, when you computed your statistics, you used only two of the three as your classification variables.

Also, note that for cases where there are two classification variables as part of the statistics, and 2D plots are done, LAMBDA will not make plots if the first classification variable has more than 10 categories. For example, suppose you have 100 years of data at 2 stations. LAMBDA plots a single graph for each category of the first variable. If you list Station first (and then year second, i.e., year within stations), you would have two plots for each species, with year on the X-axis and mean, variance or other statistical moments you have chosen on the Y-axis. However, if you had year first, LAMBDA would give an error message, for otherwise, it would be drawing 100 graphs (showing the statistics at the two stations on one graph for each year). Such an operation is likely to crash many computers, and so LAMBDA simply will not allow it. Either pool your data so that there are less than 10 categories, or switch around which classification variable you list first, so that the one with the fewest categories is listed first.

LAMBDA does not support plotting in the case of three or more classification variables at this time. Please note that LAMBDA's statistical plotting elements are rather primitive because plotting descriptive statistics is not its primary function. Statistical data should probably be exported to another package (e.g., SigmaPlot) if one wishes to make high-quality plots of descriptive statistics.

8 SUBSAMPLING THE DATA

Whenever one wishes to look at only a subset or small segment of the data (less than all of it), it is time to take a *subsample* of the data. This is accomplished through the options listed in the Subsample menu at the top of the main LAMBDA window. Note that it is not possible at this time to perform the subsampling operations on datasets that contain multiple replicates, and LAMBDA will provide an error message if you attempt this. However, subsampling can be used to *generate* new replicates from a dataset. See below for further details.

8.1 The Subsample window

The first item in this menu, Subsample Data, will bring up a new window. This window allows the user to set up the parameters for a subsample. Note that *nothing will be done to the dataset yet*. At this point you are simply setting up the parameters for a subsequent step. In the Subsample window, you will be allowed to select ranges of any Classification, Variate, or Covariate variable that you wish. For example, imagine the following dataset:

Year	Station	birds	fish
1977	1	10	100
1977	2	11	90
1977	3	9	27
1978	1	30	182
1978	2	22	88
1978	3	11	144
1979	1	12	25
1979	2	44	41
1979	3	33	69

You might wish to look only at the “Station 1” data. In this case, you would select `Station` for the Field (in the drop-down menu), and then select the desired range. LAMBDA provides a set of options for the minimum and maximum. One option is always “all,” which means that you are not actually subsampling this variable (it will use every entry). The second option is “nonzero,” which is useful for culling missing or zero values out of the dataset. The other options will be the minimum to the maximum value in that variable, with 20 increments in between. In our example dataset, this would be values ranging from 1 to 3, in increments of 0.05. Since in this case `Station` is in whole numbers, and we want cases where the station = 1, selecting a minimum of 1 and a maximum of 1.05 would do the trick. All of the stations over 1.05 (that is to say, the 2s and 3s) would be eliminated with such a subsampling scheme, and only Station 1 would be left. When you have the values you desire for the field's minimum and maximum, click the Subsample button, and this range of values will be saved in memory for later use. You will see a list of the subsampling scheme appear (and be updated as you go along) on the right hand side of the window.

You can subsample as many things as you want. For instance, you might wish to subsample `Year` from 1977—1978, `Station` only for station 1, and only cases where birds where over 30 and fish were below 100 individuals. To do this, select each field separately (the order in which you choose them does not matter), choose the minimum and maximum, and

then click Subsample. If at any point you decide you have made a mistake, the entire subsampling scheme can be cleared (and re-set) using the Clear button. The Cancel button will close the window without doing anything.

When you are happy with your subsampling scheme, click Done, and the window will close, and the subsampling scheme will be saved in memory. *The subsample scheme will not yet be applied to the dataset*. Instead, a second, subsampled dataset will be created in memory – one that fulfills only the scheme you have generated. For example, it will contain only data from Years 1-3, at Station 2, if those are the subsample criteria you had selected.

8.2 Viewing, sorting, and saving the subsampled data

You can view this subsampled dataset using the Display->View Subsampled Dataset menu option. You can also sort this dataset, review your subsample criteria, and plot the subsampled data from the Subsample menu options. Finally, if you are happy with the subsample, you can save it under the File ->Save Subsample As... menu item, and you can load a saved subsample as well, so that you can work across multiple LAMBDA sessions.

8.3 Applying your subsample to the full dataset

Finally, if you decide you want to apply the subsample to the entire dataset, choose the menu option Apply Subsample criteria to full database, and the main data in memory will be altered accordingly. This process cannot be reversed, so make sure to *save your work* before doing this. Because the MAR-1 operations, Descriptive statistics, and other operations can only be performed on the full database (*not* the subsample dataset), you will need to perform this operation before conducting any analysis on the subsampled data.

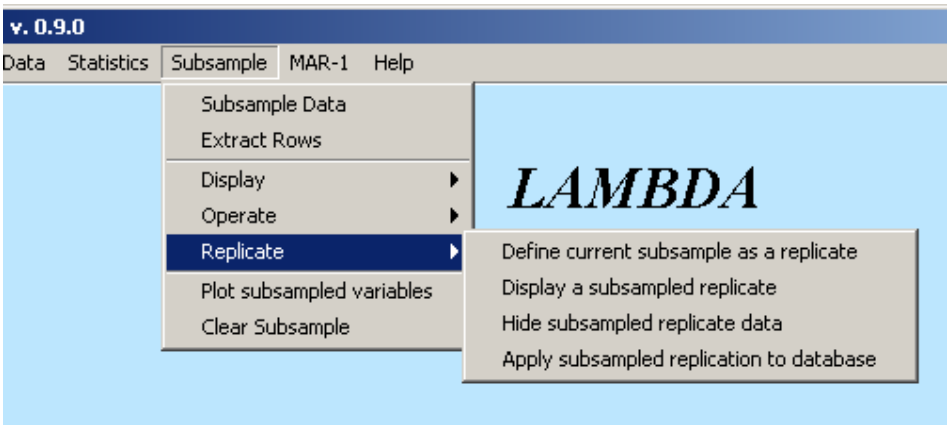
8.4 Subsampled data as replicates

Often, you will have all your data in one large dataset, but will want to consider parts of it (subsets of it) as replicates. For example, you might have sampled your time series at different locations, and you might want to assume each sample site is a replicate of the whole community. Again consider the following dataset:

Year	Station	birds	fish
1977	1	10	100
1977	2	11	90
1977	3	9	27
1978	1	30	182
1978	2	22	88
1978	3	11	144
1979	1	12	25
1979	2	44	41
1979	3	33	69

Here, you might wish to consider the three sample stations as replicates. However, they are in the same dataset. To split them up into different replicates, so that you can run a separate MAR-1 analysis on each, you can use the Subsample features. Invoke the subsample window (section 9.1) and choose the first value range to subsample (for the first replicate). In this example, we would choose the field as Station and the min and max to be 1.0 and 1.05. This

subsample would include only lines of data where the Station was between 1.0 and 1.05 (in other words, Station = 1 in our example dataset). After closing the subsample window, we would then choose options from the Replicate menu:



Notice that the first option is Define current subsample as a replicate. This is what we would use. You have a subsample that includes only "Station 1" data. You can define this subsample to be a single replicate. It would be stored in a temporary dataset (until you apply it when finished) as "Replicate #1." You could then continue to subsample, creating a new subsample for station 2, and then defining that as another replicate, and then station 3, and so forth. When you're done with these definitions, you can choose Apply subsampled replication to database. As with the operation applying normal subsampling to the database, this is an irreversible step, so make sure you save your data first, if necessary. From the Replicate menu you can also display any of the replicates you have generated using subsampling, to check your work.

One important note: Replicate datasets must have the same number of records (lines of data, or steps in the time series). For example, if you have 100 time readings for station 1, and only 25 for station 2, you cannot split them up into replicates within the same dataset. LAMBDA will produce an error if you attempt to do this, and will not allow you to apply the replicates to the full database.

9 MAR-1 MODEL ESTIMATION

Estimating a MAR-1 model is, of course, is the whole reason one would use LAMBDA, rather than some other package: its ability to easily, at the click of a few buttons, perform a CLS or Maximum Likelihood based estimate of the MAR-1 model for your time series. Please note that before using the features on the MAR-1 menu, you will need, at a minimum, to have done the following:

- ✓ Loaded a dataset containing at least one variate column.
- ✓ Made sure the dataset is in LAMBDA format.
- ✓ Transformed the data using the natural logarithm.
- ✓ Scaled the data by the sampling effort[3].
- ✓ Pooled or subsampled the data so it consists of a *single* time series, or a set of several replicate time series[4].

To use the MAR-1 estimation system correctly, all of the above elements are required. Make certain, in particular, that you have a single time series for your data (per replicate), and that you do not have multiple readings for a given species on a given date. Either pool your data by station, depth, etc., or subsample the data (converting to replicates if necessary –see section 9.4), for example doing a MAR-1 model for station 1, then station 2, and so on.

9.1 Choosing the estimation method

Currently there are three estimation methods for MAR-1 models available in LAMBDA. The Conditional Least Squares (CLS) method employs standard conditional least-squares techniques to find the best model fit for your time series data. This is a quick estimation method, and is recommended by Ives et al. (2003) due to computational efficiency. You can choose this method by selecting CLS Parameter Estimation from the MAR-1 menu.The other two methods are similar, in that they are both Maximum Likelihood (ML) methods. The more standard Simplex method (which is also discussed in Ives et al. [2003]) uses MatLab's built in minimization techniques (the `fminsearch` function). It provides slightly poorer estimates (according to my tests), and takes a much longer time and is more computationally intensive than CLS estimations. Therefore I highly recommend against this method. The other ML method is called "Simulated Annealing," and is, in my experience, slightly slower than CLS, and equally accurate. These methods usually end up agreeing with each other if done properly, so which method you choose is largely a matter of preference, but from my extensive tests, I see no reason to use anything other than CLS estimates, since the other methods, at best, give the same answer, in slightly to greatly increased periods of time.

9.2 Determining which interactions to use

Perhaps the most crucial step, once your data are ready to be analyzed, is to determine exactly which interactions should, and should not, be used. Ideally, one should have some *a priori* expectations or biological intuition about the data and the interactions, from having

[3] Convert biomass to kg per square meter, or counts to number per hectare, or the like, so that the data are standardized.

[4] That is, you should not have data from multiple stations still listed together in the dataset – pool, averge, sum, or subsample so there is only one reading for each species on each sample date (see chapter 8).

worked on the system previously or communicated with those who know it well. However, this is often not the case, because frequently the interactions are simply unknown. More commonly, you will probably know a few of the interactions (or have good reason to suspect them) but not the rest. LAMBDA can handle all of these cases, from manually specifying the model, to a fully randomized parameter search for the most likely model.

9.3 Manually specifying the model

If you have some good reasons why you think certain interactions should be used and others should not, you can tell LAMBDA explicitly, by choosing each one by hand.

For either estimation method (see section 9.1), you can select Manually specify which interactions to use. You will then be presented with a list of the variates in a new window. At this stage you can cancel or re-set things, or you can select a single variate, and a new window will be presented. The window will show a list of variates, on the left, and covariates, on the right. There are two columns for the variates, with arrows pointing left and right. Use the arrow buttons to move variates from one column to the other. The "triple arrow" will move all the variates or covariates from one column to the other. Use the arrows to move variates and covariates back and forth until you have selected the ones you want to interact, and not interact, with the species of interest. You will need to do this for each one of your variates. When you click Apply, the list of known variate and covariate interactions will be updated, and you can continue to the next variable in the Specify Interactions window.

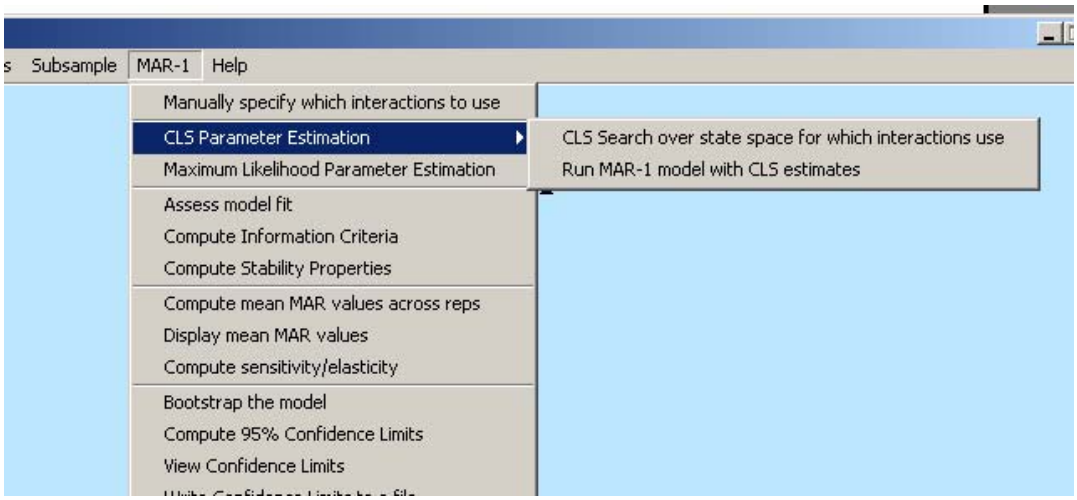
As an example of how this works, imagine a dataset with the biomass variates `Shrimp`, `Crabs`, `Birds`, and the covariates `Temp` and `WindSpd`. The Specify Interactions window will have a list of the three variates (Shrimp, Crabs, and Birds). You would click on the first one, `Shrimp`, and then click the Choose button. The next window would list the three variates on the left, and the two covariates on the right, with two columns available to each. To start with, none of the variates or covariates would be presumed to affect the biomass of Shrimp in your system.

Let us assume that you would expect Shrimp to always have an effect on themselves (density dependence). To require this interaction, you would click on `Shrimp` in the left-most column, and then click the arrow pointing to the right. `Shrimp` would move to the right column, as a required interaction. Let us also assume that you want crabs, but not birds, to interact with shrimp (you are sure the birds do not eat or otherwise interact with the shrimp in your system, for instance). You would then click on `Crabs` and use the arrow to move them to the right as well. Then, let's imagine you think temperature but not wind speed affects shrimp biomass. You would then click on `Temp` and use the arrow button to move it to the right on the covariate side of the window. At this stage, you have set things so that Shrimp are affected by themselves and crabs, but not birds, and by temperature, but not wind speed. If you are happy with that, you would select Apply, and the interaction matrix would be updated. You would then repeat this procedure for birds and crabs, and so on until all variates have their interactions specified. Once this is done, you can move on to the latter parts of the analysis.

9.4 Iterative Search and Model Estimation

Most of the time, unfortunately, you will not be lucky enough to know which interactions occur in your system, so you will have to let LAMBDA search to find them. Note that doing the search and estimating the model are *one step* in the ML technique but are *separate steps* for CLS (they are not built into the procedure).

CLS searches and estimates: To perform a CLS search to find the proper interactions, choose CLS search over state space from the MAR-1->CLS Parameter Estimates menu.



Currently, LAMBDA estimates this by iterating as many times as you wish (the default is 500, and I recommend not using any less iterations than that), starting with a random interaction and covariate matrix, and for each iteration, cycles over the cells in the matrices, changing one at a time. Over these iterations LAMBDA estimates the **A**, **B**, and **C** matrices and determines the Akaike Information Criterion (AIC) or Bayesian Information Criterion (BIC), at your discretion, for each step. The AIC and BIC are based on a combination of (1) how likely the model is given the data, and (2) a penalty for including more and more variates. The routine is thus designed to find the most likely model possible, using the fewest possible interactions. Note that self-interactions (the diagonal of the matrix) are fixed to 1, because density dependence is assumed to at least play a small role in all systems. After performing the search, LAMBDA will tell you which interactions it thinks it should include, and which it should exclude (1 = include, 0 = exclude, the interaction in question).

Note that this search only finds the interactions to be included or excluded. It does not perform the actual analysis. You must choose Run MAR-1 model with CLS estimates to perform the actual analysis. This will then run the model and find the CLS estimates of the parameters.

ML Searches: To perform a Maximum Likelihood analysis, choose Run MAR-1 model with ML estimates from the menu.

Unlike the CLS, the search step and the final estimate occur all at once. Maximum Likelihood searches (either using classical MatLab `fminsearch`, or using Simulated Annealing) keep searching over the possible variables, changing them bit by bit, until they settle on a final result, which (if done right) is the *most likely* model given the data. Once this is complete, LAMBDA will report the final model parameters.

9.5 Combining iterative search and manual specification

You can combine the two approaches, which is useful if you know for a fact certain interactions occur in your system (or at least strongly suspect it) but do not know about the others. However, note that because ML searches and final estimates occur in one step rather than two, the *combined approach can only be done with CLS estimation*. To do this, you should first let LAMBDA do the iterative CLS search. It will come up with a set of interactions that it is basically proposing to you. Then, you can select the Manually specify which interactions to use option, and when the interaction window comes up, you will see the interactions LAMBDA thinks are appropriate. You can move them around as usual, and then save the result. This will provide a hybrid model between the iterative and manually specified methods.

9.6 Saving and clearing estimates

If you wish to re-run the model (for example, to compare the two ML methods or to compare the CLS and ML method), you can save your MAR estimates using the File menu. You can then clear them from memory (wiping the slate clean so you can perform another analysis) using the Clear MAR estimates option from the MAR-1 menu.

9.7 Assessing the model fit

Once you have estimated the MAR-1 model for your system, you can assess how well the model fit. There are two menu options for doing this, which must be done in a particular order. First, you can select Assess CLS Model fit, which will compute and display in the command window the total and conditional R^2, as well as the $\boldsymbol{\Sigma}$ matrix. This first step must be done before you can compute AIC and BIC, because it also computes the likelihood, which is needed for the AIC/BIC computation. The next menu option, Compute information criteria, will compute and display the least squares or maximum likelihood AIC and BIC for your model.

9.8 Computing stability properties

The Ives et al. (2003) paper computes three stability properties – Variance, Return Rate, and Reactivity – each in two ways. LAMBDA will perform the same computations for your model. Choose Compute Stability Properties to have this step performed. The information will be displayed in the command window. For an explanation of what the terms mean, please consult the text of Ives et al. (2003).

9.9 Computing MAR-1 across many replicates

If your dataset consists of more than one replicate, then each procedure (including the iterations) needs to be done for each replicate. This is automatic within LAMBDA: you do not have to do anything special. LAMBDA will scan your data set and, if more than one replicate exists, it will simply cycle over all replicates, finding the CLS or ML estimates of

the parameters, assessing model fits, and computing dynamical properties for each replicate in turn.

The output of a multi-replicate dataset may be slightly difficult to read, as it is stored in the 3rd "dimension" of a MatLab array. Usually when one has many replicates, one is interested in the mean values of the parameter estimates in any case, and LAMBDA can provide these for you. After your estimates are complete and dynamical properties computed, choose Compute mean MAR values across reps, and LAMBDA will take the average of the **A**, **B**, **C**, and other values it has computed, including AIC, BIC, dynamical properties, and the like. You can display any or all of these values to the command window by choosing Display mean MAR values from the MAR-1 menu. When you select this option, LAMBDA will ask if you wish to save these values to a text file. If you choose Yes, you must supply a file name, and then what you see in the command window, will also be saved to a comma-separated text file (`csv`).

Note that, because computations can take a long time, LAMBDA saves its work if you have multiple replicates, storing its estimates as each replicate is concluded. This information is stored in the `LAMBDA/work` directory in a file called `WORK.mat`. If your computer crashes, or you otherwise terminate LAMBDA by any means other than the File-> Close LAMBDA option, this `WORK.mat` file will remain in the `LAMBDA/work` directory. The next time LAMBDA is started, you can re-start your MAR-1 analysis and LAMBDA will pick up where it left off (it will ask if you wish to do this). Note: *At the time of this writing (version 0.9.0) this data storage is only done with CLS estimates, not with ML estimates*.

9.10 Bootstrapping the model

You can bootstrap your model to obtain confidence limits. Basically, this procedure randomizes the **E** (process error) matrix, and re-runs the model. This can be done any number of times, though the default is 2,000. When done, a `BOOT` structure will be produced, containing all the iterations of the bootstrap. This is then used for later steps (there is no need to view it). After running the bootstrap, you can compute the confidence limits, which will be stored in the `CONFLIM` structure. This structure contains the confidence limits. The text in the command window tells you how the information is stored, and it can be displayed from the command line. Alternatively, you can choose the View Confidence Limits option, and a new window will be produced. This will provide you with buttons and you can examine the confidence limits of the bootstrapping this way. You may also choose to Write confidence limits to a file, if you wish to save the bootstrap results for later analysis or for use by another program. Finally, to make it easy to tell whether CIs overlap the zero values, you can display (and save to a file) the **B** and **C** matrices in a "pretty printed" format. Simply choose the "Significant Interactions" button when using the "View Confidence Limits" window.

Please note that *replicated data cannot be bootstrapped*. LAMBDA will only perform a bootstrap analysis on dataset that is contained in a single replicate. If you wish, you can bootstrap individual replicates of your dataset and store them. Remember that the whole reason to bootstrap a dataset, however, is to create simulated replicates to generate error estimates, confidence limits, and the like. With replicated data, you do not need to do a

bootstrap – the replicates can provide you with estimates of error, which you can view using the Show non-zero species interactions option from the MAR-1 menu.

9.11 Normal Probability Plots

In addition to numerical analysis, LAMBDA can also provide normal probability plots of the residuals in each of our variates. The bottom option on the menu will execute this command. Consult a standard statistics text for an explanation of what these plots display.

9.12 Saving MAR-1 Estimates

The MAR-1 estimation process can take a long time, and once it is done, it is best to save it in case you ever need to examine it again. This can be done from the File -> MAR menu. Also from that menu, older MAR-1 estimates can be re-loaded (so, for example, if you wish to re-examine an earlier analysis, you can do using this menu option).

9.13 Tools for evaluating MAR-1 estimates

The MAR->Tools menu provides some simple tools for evaluating MAR-1 estimates. For example, sometimes one has a "known" (or previously computed) **B** matrix, and wants to know the Stability properties of it. These can be computed by using the first option to check the stability of a known matrix. A window will appear asking for an input text file (`*.txt`). This should be a file containing your **B** matrix in text format. Once you select the file, the stability properties will be reported in the command window.

Additionally, you may wish to know if the top *N* interactions (with the highest absolute value) found by the MAR-1 estimation step, compare with those in a known **B** matrix (for example, the one you used for a simulation, or one from another sampling site). You will need to have performed a MAR-1 estimate first. Then choose this option from the menu, and select the text file containing the **B** matrix to which you are comparing the estimates. You will then be asked to tell LAMBDA how many interactions matter. This is up to you – you will need to decide which interactions are strong enough that they "count," and which are not. By default, LAMBDA assumes one interaction matters per species, but you may change this to any amount you wish. LAMBDA will then compare the top *N* interactions in the known **B** matrix, with those in the estimated matrices, and report on the percent correct. For example, if the estimate got 3 of the top 4 interactions correct, this would yield a 75% similarity.

Finally, if you wish to see the distribution (rather than just the mean and standard deviation) of the various stability properties, you can use the Plot Frequency Distribution option from the tools menu to accomplish this. Note that this function only works if there are two or more replicates of the data.

9.14 Estimating Elasticity and Sensitivity of the B matrix

One can calculate the elasticity and sensitivity of any square matrix. Sensitivity is the effect on the eigenvalues (λ) of a change in a given pair-wise interaction. Since in MAR-1 models l

is the stability of the system, then Sensitivity in this context becomes “the sensitivity of the stability” to changes in the B values. Elasticity is the proportional response to a proportional perturbation of the system. Thus, elasticity can be seen as “proportional sensitivity.” Whether these computed values are of any real interest will depend on the system being studied, and is left up to the judgment of the investigator.

10 SETTING PREFERENCES FOR LAMBDA

The Edit menu has an option for setting preferences in LAMBDA. Choosing this option brings up a checklist. You can use the checklist to select which options you wish to use. If the box is checked, it means the option is turned "on." Most of these involve telling LAMDA whether or not to keep giving warnings about potential challenges to the data analysis – if you are sure of what you are doing, you can turn these off. However, two other options bear further explanation because their meaning is not obvious.

10.1 The use of "inlined" code

Because it makes the use and maintenance of code easier, LAMBDA is written almost entirely as a suite of function calls. Thus, when you ask LAMBDA to perform a MAR-1 analysis, it must first generate the **U**, **X** and **Y** matrices from your data, which it does by passing the data into the `GenU` and `GenXY` functions. LAMBDA must then generate CLS fits using the `CLSfit` function, and then assess those fits with the `assessCLSfit` function, and so forth. Each of these functions is used repeatedly in the code, which is why it makes sense to build a single function, and call it multiple times, rather than to write the same code over and over again as a single script.

Unfortunately, such a scheme is not possible when using parallelized code (see below), and sadly, it also slows down computations considerably (by a factor of about two). So it is actually faster to run a script that has the same code pasted into it, over and over again, than it is to call those lines of code repeatedly as functions. The drawback to this increased computational speed, is that the code is less easily maintained. If the CLS fits are done in one function, and I find a bug in the CLS fit algorithm, it is corrected once (in the function) and is thereby fixed throughout the entire program. However, if the code is repeated four or five times in one file (rather than as a function call), the bug will have to be corrected in every case, and it is always possible to miss doing one of them. Thus, it is harder to make corrections in all the necessary places when the code is linear ("inlined" code) rather than written by function calls.

The correct way to code depends on your needs, of course, but my own preference is for maintainable and bug-free code over computational speed. As a result, I coded LAMBDA to be function-based, rather than "inlined." However, because of the needs of Star-P, and the needs some folks will have to speed up their computations, I have included an "inlined" version. Why should you pick one over the other then? Well, if you want speed, you would choose the "inlined" option (or use Star-P, which would be even faster). However, almost all of my work was done using the function-based version on a single desktop. As such, I have beaten that to death and am absolutely certain the functions are bug-free. The inlined version is much newer, and although I do not believe there are any bugs, I cannot be quite as certain about it. It will never be possible to be quite as certain, because any time a change is made in that inlined version, it will have to be made numerous times, and so there is always a greater chance of an error.

I therefore recommend not using the inlined version, unless your time requirements are such that it is the only practical choice. The functional form is in a more advanced state of

development and less likely to contain errors. I will note that so far as I know, the inlined version is free of errors as well but, ultimately, with that version you are "taking your chances." The code is open source of course, so you are welcome to check it yourself and make sure it is correct (if you can understand my coding habits and comment style).

10.2 The Star-P option

Star-P is a version of MatLab created by a different company (with the permission of The Mathworks), designed to run on parallel computers (a single machine with two or more processors). Ordinarily Matlab executes code sequentially. In other words, if you have 80 replicates, and you have one processor, the replicates are done one at a time. With a parallel machine having, say, eight processors, Star-P can "farm out" one replicate to each processor, thereby resulting in a much higher computational speed. Although in theory this should be about an eight-fold increase, our tests thus far have only yielded a 2-3 fold increase over the inlined version. Since that version is about double the speed of the functional form (see above), this amounts to a 4-6 fold increase in computation speed relative to using the functional form of LAMBDA on a single-processor desktop under standard MatLab.

If you have access to a Star-P server, you can use this faster version of LAMBDA with it. You should start Star-P first (see your system administrator for details on how to do this). Once it is up and running, you may execute LAMBDA normally. LAMBDA is able to detect that you have Star-P running, and will ask you if you wish to use the Star-P version of the functions. If you choose "yes," LAMBDA will run CLS estimates (and only those) in parallel. All other operations, including ML estimates, are done in series, not in parallel, so no performance benefit can be derived from using Star-P for those operations. In other words, the only time you should use the Star-P version is for doing CLS estimates of large datasets with many replicates.

At this point, the Star-P compatibility should be considered to be in "pre-Beta" stages. I have used it enough to be fairly sure it works properly, but unless a large number of people request Star-P functionality, I don't plan on adding any further Star-P features to LAMBDA. If you wish further parallelization, you can contact me with the request, and I will take it under advisement. However, please understand that this is provided as a convenience, and I do not consider it to be critical to the basic functioning of LAMBA. (In other words, it is not a top priority.)

APPENDIX 1: QUICKSTART GUIDE

This quick-start guide will give you a very brief script to follow for quickly importing data from text files and doing a basic MAR-1 analysis (with nothing fancy).

1. Start up LAMBDA by changing to the LAMBDA directory and typing "`lambda`" at the MatlLab command prompt.

2. Click on the File menu and select Import -> From a text file.

3. Navigate through the directory structure until you find your text file and select it.

4. Select which variables in your dataset are Classification variables

5. Next, select which ones are variates (i.e., species of interest). Whatever is left over (if anything) will be classified as a co-variate.

6. Make sure your data were imported correctly by clicking on the Data menu and choosing the view -> all option.

7. Click on the statistics menu and choose transform the data -> natural logarithm.

8. Click on the MAR-1 menu, select CLS Parameter Estimation, and choose search over state space…

9. Once the search is complete, again select MAR-1 -> CLS Parameter Estimation but this time choose Run MAR-1 model with CLS estimates.

10. Click on the MAR-1 menu and choose Assess model fit.

11. Click on the MAR-1 menu and choose Compute Information Criteria.

12. Click on the MAR-1 menu and choose Compute Stability Properties.

13. Click on the File menu, and choose MAR -> Save MAR Results.

14. Your MAR-1 model results will be displayed in the command window.

APPENDIX 2: FLOW CHART OF LAMBDA PROCEDURES

Fig. 2