CAMBRIDGE

Cambridge University Press
978-0-521-86783-2 - Modeling Methods for Marine Science
David M. Glover, William J. Jenkins and Scott C. Doney
Excerpt
More information

1

Resources, MATLAB primer, and introduction to linear algebra

"Begin at the beginning," the King said, very gravely, "and go on till you come to the end: then stop."

Lewis Carroll

Welcome to *Modeling Methods for Marine Science*. The main purpose of this book is to give you, as ocean scientists, a basic set of tools to use for interpreting and analyzing data, for modeling, and for scientific visualization. Skills in these areas are becoming increasingly necessary and useful for a variety of reasons, not the least of which are the burgeoning supply of ocean data, the ready availability and increasing power of computers, and sophisticated software tools. In a world such as this, a spreadsheet program is not enough. We don't expect the reader to have any experience in programming, although you should be comfortable with working with computers and web browsers. Also, we do not require any background in sophisticated mathematics; undergraduate calculus will be enough, with some nodding acquaintance with differential equations. However, much of what we will do will not require expertise in either of these areas. Your most valuable tool will be common sense.

1.1 Resources

The activities of modeling, data analysis, and scientific visualization are closely related, both technically and philosophically, so we thought it important to present them as a unified whole. Many of the mathematical techniques and concepts are identical, although often masked by different terminology. You'll be surprised at how frequently the same ideas and tools keep coming up. The purpose of this chapter is threefold: to outline the goals, requirements, and resources of the book, to introduce MATLAB (and give a brief tour on how to use it), and to review some elements of basic linear algebra that we will be needing in subsequent chapters.

As we stated in the preface (you *did* read the preface, didn't you?), our strategy will be to try to be as "correct" as possible without being overly rigorous. That is, we won't be dragging you through any theorem proofs, or painful details not central to the things you need to do. We will rely quite heavily on MATLAB, so you won't need to know

© in this web service Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-521-86783-2 - Modeling Methods for Marine Science
David M. Glover, William J. Jenkins and Scott C. Doney
Excerpt
More information

how to invert a matrix or calculate a determinant. But you should know qualitatively what is involved and why some approaches may be better than others; and, just as important, why some procedures fail. Thus we will try to strike a happy medium between the "sorcerer's apprentice" on the one extreme and having to write your own FORTRAN programs on the other. That is, there will be no programming in this book, although you will be using MATLAB as a kind of programmer's tool. The hope is that the basic mathematical concepts will shine through without getting bogged down in the horrid details.

1.1.1 Book structure

The course of study will have four sections. The first section (Chapters 1–7) is aimed at data analysis and some statistics. We'll talk about least squares regression, principal components, objective analysis, and time-series approaches to analyzing and processing data. The second section (Chapters 8–12) will concentrate on the basic techniques of modeling, which include numerical techniques such as finite differencing. The third (Chapters 13–17) will consist of case studies of ocean models. The largest amount of time will be spent on 1D models (and to a lesser extent 2D and 3D models) since they contain most of the important elements of modeling techniques. The final section (Chapters 18–19) will discuss inverse methods and scientific visualization, which in some respects is an emerging tool for examining model and observational data. Throughout the text will be examples of real ocean and earth science data used to support the data analysis or modeling techniques being discussed.

1.1.2 World Wide Web site

We teach a course at the Woods Hole Oceanographic Institution using this text, and this text is supported by a web page (`http://www.cambridge.org/glover`), where the most up-to-date versions of our codes are available. The MathWorks also maintains a web presence (`http://www.mathworks.com/`) and versions of our work may be available there.

We draw upon a number of other sources, some from textbooks of a more applied mathematics background and others from the primary ocean literature. Davis's (2002) book is a primary example of the former, although it really only covers the first section of this book. Bevington and Robinson (2003) is such a useful book that we strongly recommend you obtain a copy (sadly it is not as inexpensive as it once was). The book by Press *et al.* (2007) is useful for the second section, although the Roache (1998) book is best for finite difference techniques (and with the latest edition it is happily no longer out-of-print). The third and fourth sections will rely on material to be taken from the literature. There is a list of references at the end of the book, and the other texts are listed only as supplemental

© in this web service Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-521-86783-2 - Modeling Methods for Marine Science
David M. Glover, William J. Jenkins and Scott C. Doney
Excerpt
More information

references in case you (individually) have a need to delve more deeply into some aspects of the field.

1.2 Nomenclature

We could put a lot of quote marks around the various commands, program names, variables, etc. to indicate these are the things you should type as input or expect as output. But this would be confusing (is this an input? or an output?) plus it would be a real drag typing all those quote marks. So a word about nomenclature in this text. This book is being typeset with LaTeX, and we will try to use various fonts to designate different things in a consistent manner. From now on, when we mean this is something that is "computer related" (the name of a file, a variable in a MATLAB program, a MATLAB command, etc.) it will be in `simulated typewriter font`. If it is a downloadable link, it will be underlined as in `http://URL` or `filename`, at least on first introduction. If, however, we are referring to the mathematics or science we are discussing (variables from equations, mathematical entities, etc.), we will use *mathtext* fonts. In particular, scalar variables will be in simple italic font (x), vectors will be lowercase bold face ($\mathbf{b}$), and matrices will be upper case bold face ($\mathbf{A}$).

If you have read enough math books, you have learned that there is no universally consistent set of symbols that have unique meanings. We will try to be consistent in our usage within the pages of our book, but there are only 26 letters in our alphabet and an even smaller number of Greek characters, so some recycling is inevitable. As always, the context of the symbol will be your best guide in deciphering whether this is the λ from Chapter 4 (eigenvalues) or Chapter 13 (decay constants). On the first introduction of new jargon or names of things the text will be *italicized*.

1.3 A MATLAB primer

You can read this book, and benefit greatly, without ever touching a computer. But this is a lot like theoretical bicycle riding; there is no better way to learn than with hands on experience, in this case hands on a keyboard. MATLAB is a very powerful tool (there are others), and for showing practical examples it is more useful than pseudocode.

The best way to learn how to use MATLAB is to do a little tutorial. There are at least two places where you can get hold of a tutorial. The best is the *MATLAB Primer* (Davis and Sigmon, 2004), the seventh edition as of this writing. A second place is to use the online help available from your MATLAB command window. If you are using the MATLAB graphical user interface (GUI), pull down the help menu from the toolbar at the top of the command window and select "MATLAB help". This will start the online documentation MATLAB help browser. In the "Contents" tab on the left select the "MATLAB" level and go to "Printable Documentation (PDF)" to find a listing of manuals available to you. The first one, "Getting Started", is a good place to begin. If you are not using the command

© in this web service Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-521-86783-2 - Modeling Methods for Marine Science
David M. Glover, William J. Jenkins and Scott C. Doney
Excerpt
More information

window GUI, type `helpdesk` at the MATLAB prompt, and this online documentation help browser will be launched. If you get an error with the `helpdesk` command we suggest you speak with your system administrator to learn how to configure your web browser or type `help docopt` if you are an experienced computer user. Whichever one you pick, just start at the beginning and work through the tutorial. It basically shows you how to do most things. Don't worry if some of the linear algebra and matrix math things don't make much sense yet.

Read the primer. We won't waste time repeating the words here, except to point out a few obvious features.

- MATLAB is case sensitive; a variable named `A` is not the same as `a`. Nor is `Alpha` the same as `ALPHA` or `alpha` or `AlPhA`. They are all different.
- Use `help` and `lookfor`. If you don't know the name of a command, but, for example, want to know how to make an identity matrix, type `lookfor identity`. You will then be told about `eye`. Use `help eye` to find out about the `eye` command. Note that MATLAB capitalizes things in its help messages for EMPHASIS, which confuses things a little. Commands and functions are *always* in lower case, although they are capitalized in the help messages.
- Remember that matrix multiplication is not the same as scalar or array multiplication; the latter is designated with a "dot" before it. For example `C=A*B` is a matrix multiplication, whereas `C=A.*B` is array multiplication. In the latter, it means that the elements of **C** are the scalar product of the corresponding elements of **A** and **B** (i.e. the operation is done element by element).
- The colon operator (:) is a useful thing to learn about; in addition to being a very compact notation, it frequently executes much, much faster than the equivalent `for ... end` loop. For example, `j:k` is equivalent to `[j, j+1, j+2, ..., k]` or `j:d:k` is equivalent to `[j, j+d, j+2*d, ..., j+m*d]` where `m = fix((K-J)/D)`. There's even more: the colon operator can be used to pick out specific rows, columns, elements of arrays. Check it out with `help colon`.
- If you don't want MATLAB to regurgitate all the numbers that are an answer to the statement you just entered, be sure to finish your command with a semicolon (;).

MATLAB has a "scripting" capability. If you have a sequence of operations that you routinely do, you can enter them into a text file (using your favorite text editor, or better yet, MATLAB's editor) and save it to disk. By default, all MATLAB script files end in a `.m` so that your script (or "m-file") might be called `fred.m`. You can edit this file with the MATLAB command `edit fred`; if the file does not exist yet, MATLAB will prompt you, asking if you wish to create it. Then, you run the script by entering `fred` in your MATLAB window, and it executes as if you had typed in each line individually at the command prompt. You can also record your keystrokes in MATLAB using the `diary` command, but we don't recommend that you use it; better to see A.1 in the appendix of

© in this web service Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-521-86783-2 - Modeling Methods for Marine Science
David M. Glover, William J. Jenkins and Scott C. Doney
Excerpt
More information

this book. You will learn more about these kind of files as you learn to write functions in MATLAB.

You can load data from the hard drive directly into MATLAB. For example, if you had a data file called `xyzzy.dat`, within which you had an array laid out in the following way:

```
1 2 3 4
5 6 7 8
9 0 1 2
```

then you could load it into MATLAB by saying `load xyzzy.dat`. You would then have a new matrix in your workspace with the name `xyzzy`. Note that MATLAB would object (and rightly so, we might add) if you had varying numbers of numbers in each row, since that doesn't make sense in a matrix. Also, if you had a file named `jack.of.all.trades` you would have a variable named `jack` (MATLAB is very informal that way). Note that if you had a file without a "." in its name, MATLAB would assume that it was a "mat-file", which is a special MATLAB binary format file (which you cannot read/modify with an editor). For example, `load fred` would cause MATLAB to look for a file called `fred.mat`. If it doesn't find it, it will complain. But first, make sure MATLAB is looking in the correct file directory, which is an equivalent way of saying make sure the file is in MATLAB's `PATH`.

You can save data to disk as well. If you simply type `save`, MATLAB saves everything in your workspace to a file called `matlab.mat`. Don't try to read it with an editor (remember it's in binary)! You can recover everything in a later MATLAB session by simply typing `load`. You can save a matrix to disk in a "readable" file by typing `save foo.dat xyzzy -ascii`. In this case you have saved the variable `xyzzy` to the file `foo.dat` in ASCII (editable) form. You can specify more than one variable (type `help save` to find out more). Remember the `-ascii`, because nobody but MATLAB can read the file if you forget it.

You can even read and write files that are compatible with (shudder!) Excel. There are a number of ways to do this. For example, to read an Excel file you can use `A=xlsread('filename.xls')`, and the numeric data in `filename.xls` will be found in the MATLAB variable `A`. The `xlsread` function has a number of other capabilities; to learn more simply type `help xlsread`. MATLAB even has a function that will tell you things about what is inside the Excel file, for example `SheetNames`; to learn more type `help xlsfinfo`. Also, the MATLAB functions `csvread` and `csvwrite` facilitate transferring data to and from Excel; do a `help csvread` to have MATLAB explain how to use these functions.

A final word about the MATLAB code presented in this book. As we write (and rewrite) these chapters we are using MATLAB release R2007b (depending whether we upgraded recently). To the best of our knowledge, all of the examples and programs we provide in this book are compatible with R2007b (version 7.5). As time goes on, some of our code will undoubtedly become incompatible with future MATLAB release X. To

© in this web service Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-521-86783-2 - Modeling Methods for Marine Science
David M. Glover, William J. Jenkins and Scott C. Doney
Excerpt
More information

deal with this eventuality we have decided to make our material available on web pages (http://www.cambridge.org/glover) instead of the more static CD-ROM media.

1.4 Basic linear algebra

A scalar is a single number. A vector is a row or column of numbers. You can plot a vector, for example [3, 7, 2], which would be an arrow going from the origin to a point in three-dimensional space indicated by $x = 3$, $y = 7$, and $z = 2$. A matrix may be thought of as a bundle of vectors, either column or row vectors (it really depends on what "physical reality" the matrix represents). If each of the vectors in a matrix is at right angles to all of its mates, then they are said to be *orthogonal*. They are also called *linearly independent*. If the lengths of the vectors, as defined by the square root of the sum of the squares of its components (i.e. $\sqrt{x^2 + y^2 + z^2}$), are also 1, then they are said to be *orthonormal* (ortho – at right angles, normal – of unit length). For example, a vector $[1/\sqrt{2}, 0, 1/\sqrt{2}]$ has a length of 1, as does $[1/\sqrt{3}, 1/\sqrt{3}, 1/\sqrt{3}]$ and [0, 0, 1].

Before we start, there are some simple rules for matrix manipulation. First, you can only add or subtract matrices of the same size (same number of rows and columns). The one exception in MATLAB to this is when you add or subtract scalars to/from a matrix. In that case the scalar is added/subtracted from each element of the matrix individually. Second, when you multiply matrices, they must be *conformable*, which means that the left matrix must have the same number of columns as the right matrix has rows:

$$\begin{bmatrix} 1 & 2 & 4 \\ 5 & 8 & 7 \end{bmatrix} \times \begin{bmatrix} 3 & 9 \\ 2 & 4 \\ 1 & 8 \end{bmatrix} = \begin{bmatrix} 11 & 49 \\ 38 & 133 \end{bmatrix} \tag{1.1}$$

Matrix multiplication is non-commutative. That is, in general, $\mathbf{A} \times \mathbf{B}$ is not the same as $\mathbf{B} \times \mathbf{A}$ (in fact, the actual multiplication may not be defined in general). The algorithm for matrix multiplication is straightforward, but tedious. Have a look at a standard matrix or linear algebra text to see how matrix multiplication works. Even though you won't actually be doing matrix multiplication by hand, a lot of this stuff is going to make more sense if you understand what is going on behind the scene in MATLAB; Strang (2005) is a good place to start.

1.4.1 Simultaneous linear equations

The whole idea of linear algebra is to solve sets of simultaneous linear equations. For example, consider the following system of equations:

$$\begin{aligned} x + 2y + z &= 8 \\ 2x + 3y - 2z &= 2 \\ x - 2y + 3z &= 6 \end{aligned} \tag{1.2}$$

© in this web service Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-521-86783-2 - Modeling Methods for Marine Science
David M. Glover, William J. Jenkins and Scott C. Doney
Excerpt
More information

which can be represented as:

$$\mathbf{Ax} = \mathbf{b} \tag{1.3}$$

where **A**, the matrix of the coefficients, looks like:

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 3 & -2 \\ 1 & -2 & 3 \end{pmatrix} \tag{1.4}$$

The column value vector (**b**) representing the *right-hand side* (RHS) of the equation system is:

$$\mathbf{b} = \begin{pmatrix} 8 \\ 2 \\ 6 \end{pmatrix} \tag{1.5}$$

Note that the matrix **A** contains the coefficients of the equations, **b** is a column vector that contains the knowns on the RHS, and **x** is a column vector containing the unknowns or "target variables".

You enter **A** into MATLAB the following way:

```
A = [ 1 2 1; 2 3 -2; 1 -2 3];
```

(the array starts and ends with the *square bracket*, and the rows are separated by semicolons; also we've terminated the statement with a semicolon so as not to have MATLAB regurgitate the numbers you've just typed in.) You can enter **b** with:

```
b=[8; 2; 6]
```

Finally, the column "unknown" vector is:

$$\mathbf{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{1.6}$$

(Don't try to enter this into MATLAB, it's the answer to the question!)

Now how do you go about solving this? If we gave you the following scalar equation:

$$3x = 6 \tag{1.7}$$

then you'd solve it by dividing both sides by 3, to get $x = 2$. Similarly, for a more general scalar equation:

$$ax = c \tag{1.8}$$

you'd do the same thing, getting $x = c/a$. Or more appropriately $x = (1/a) \times c$, or put slightly differently, $x = inv(a) \times c$. Here, we've said that $inv(a)$ is just the *inverse* of the number.

Well, you can do a similar thing with matrices, except the terminology is a little different. If you enter these data into MATLAB, then you can solve for the values of the three variables (x, y, and z) with the simple statement `x=A\b`. This is really equivalent to the

© in this web service Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-521-86783-2 - Modeling Methods for Marine Science
David M. Glover, William J. Jenkins and Scott C. Doney
Excerpt
More information

statement `x=inv(A)*b`. Now check your answer; just multiply it back out to see if you get `b` by typing `A*x`. The answers just pop out! This is simple. You could do this just as easily for a set of 25 simultaneous equations with as many unknowns (or 100 or 10^4 if you have a big computer).

But what did you just do? Well, it's really simple. Just as simple scalar numbers have inverses (e.g., the number 3 has an inverse; it's 1/3), so do matrices. With a scalar, we demand the following:

```
scalar * inv(scalar) = 1

i.e.:  3 * 1/3 = 1
```

So with a matrix, we demand:

```
matrix * inv(matrix) = I
```

Here we have the matrix equivalent of "1", namely `I`, the *identity matrix*. It is simply a square matrix of the same size as the original two matrices, with zeros everywhere, except on the diagonal, which is all ones. Examples of identity matrices are:

```
                                1 0 0 0
          1 0 0                 0 1 0 0
1 0       0 1 0                 0 0 1 0
0 1       0 0 1                 0 0 0 1
```

Well, you get the idea. Oh, by the way, a matrix must be square (at the very least) to have an inverse, and the inverse must be the same size as the original. Note that like its scalar little brother, "1", the identity matrix times any matrix of the same size gives the same matrix back. For example, `A*I=A` or `I*A=A`.

OK, now try this with the matrix you keyed into MATLAB. Type `A*inv(A)` (you are multiplying the matrix by its inverse). What do you get? You get the identity matrix. Why are some "ones" represented as "1" and some by "1.000"? Also, you sometimes get 0.0000 and −0.0000 (yeah, we know, there's no such thing as "negative zero"). The reason is that the computation of the matrix inverse is not an exact process, and there is some very small roundoff error (see Chapter 2, Section 2.1.5). Which means that "1" is not exactly the same as "1.000000000000000000000000", but is pretty darn close. This is a result of both the approximate techniques used to compute the inverse, and the finite precision of computer number representation. It mostly doesn't matter, but can in some special cases. Also try multiplying the matrix `A` by the identity matrix, `A*eye(3)` (the identity matrix of rank 3). What is *rank*? Keep reading!

Finally, let's do one more thing. We can calculate the *determinant* of a matrix with:

```
d=det(A)
```

The determinant of a matrix is a scalar number (valid only for square matrices) and gives insight into the fundamental nature of the matrix. If it is zero, there may be trouble ahead. We will run into the determinant in the future. We won't tell you how to calculate it, since

© in this web service Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-521-86783-2 - Modeling Methods for Marine Science
David M. Glover, William J. Jenkins and Scott C. Doney
Excerpt
More information

MATLAB does such a fine job of doing it anyway. If you're interested, go to a matrix math text. Anyway, now calculate the determinant of the inverse of `A` with:

```
dd=det(inv(A))
```

(See how we have done two steps in one; MATLAB first evaluates the `inv(A)` then feeds the result into `det()`.) Guess what? `dd = 1/d`. Before we do anything else, however, let's save this matrix to a new variable `AA` with:

```
AA=A;
```

Now that you see how solving linear equations works, try another set of equations:

$$\begin{aligned} x + 2y + z &= 8 \\ 2x + 3y - 2z &= 2 \\ 3x + 5y - z &= 10 \end{aligned} \tag{1.9}$$

i.e., you enter,

```
A = [1 2 1; 2 3 -2; 3 5 -1]
b = [8; 2; 10]
x = A\b
```

Whoops! You get an error message:

```
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 1.171932e-017
```

(Note that your result for `RCOND` might be a little different, but generally very small.)

What happened? Well, look closely at the original set of equations. The third equation is really not very useful, since it is the sum of the first two equations. It is not linearly independent of the other two. You have in effect two equations in three unknowns, which is therefore not solvable. This is seen in the structure of the matrix. The error message arises when you try to invert the matrix because it is *rank deficient*. If you look at its rank, with `rank(A)`, you get a value of 2, which is less than the full dimensionality (3) of the matrix. If you did that for the first matrix we looked at, by typing `rank(AA)`, it would return a value of 3.

Remember our friend the determinant? Try `det(A)` again. What value do you get? Zero. If the determinant of the inverse of `A` is the inverse of the determinant of `A` (get it?), then guess what happens? A matrix with a determinant of zero is said to be *singular*.

1.4.2 Singular value decomposition (SVD)

Common sense tells you to quit right there. Trying to solve two equations with three unknowns is not useful ... or is it? There are an infinite number of combinations of x, y,

© in this web service Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-521-86783-2 - Modeling Methods for Marine Science
David M. Glover, William J. Jenkins and Scott C. Doney
Excerpt
More information

and z that satisfy the equations, but not every combination will work. Sometimes it is of value to know what the range of values is, or to obtain some solution subject to some other (as yet to be defined) criteria or conditions. We will tell you of a sure-fire technique to do this, *singular value decomposition*. Here's how it works.

You can split a scalar into an infinite number of factors. For example, you can represent the number 12 in the following ways:

- 2×6
- 3×4
- 1×12
- $2 \times 3 \times 2$
- $1 \times 6 \times 2$
- 0.5×24
- 1.5×8
- $1\frac{2}{3} \times 7.20$
- $24 \times 1 \times 2 \times 0.25$

and so on. You can even require (constrain) one or more of these factors to be even, integer, etc. Well, the same is true for matrices. This leads to a host of different techniques called *decomposition*. You'll hear of various ones, including "LU", "QR" and so on. Their main purpose is to get the matrices in a form that is useful for solving equations, or eliminating parts of the matrix. They are all used by MATLAB but the one you should know about is SVD (which is short for singular value decomposition). It is nothing magic, but allows you to break a matrix down into three very useful components. The following can be "proved" or "demonstrated" with several pages of matrix algebra, but we will just make an assertion. We assert that for any matrix ($\mathbf{A}$) of dimension $N \times M$ (N rows, M columns), there exists a triple product of matrices:

$$\mathbf{A} = \mathbf{USV}' \tag{1.10}$$

where:

- $\mathbf{U}$ is column orthonormal (i.e. each column as a vector is orthogonal to the others and of "unit length" – sum of squares of elements is 1) and of size $N \times M$,
- $\mathbf{S}$ is a diagonal matrix of dimension $M \times M$, whose diagonal elements are called *singular values*. These values may be zero if the matrix is *rank deficient* (i.e. its rank is less than the shortest dimension of the matrix),
- $\mathbf{V}$ is an orthonormal square matrix of size $M \times M$. In linear algebra and MATLAB $\mathbf{V}'$ means $\mathbf{V}$ *transpose* where you swap rows and columns, i.e. if

$$\mathbf{G} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \quad \text{then} \quad \mathbf{G}' = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix} \tag{1.11}$$

Note that there are two ways of defining the size of the matrices, you may come across the other in Strang's (2005) book, but the results are the same when you multiply them

© in this web service Cambridge University Press www.cambridge.org