

Introduction to MATLAB[®] Programming

With an emphasis on Software Design through Numerical Examples

Jonathan H. Dorfman

Decagon Press, Inc.

Introduction to MATLAB[®] Programming

With an emphasis on Software Design through Numerical Examples

Jonathan H. Dorfman

Decagon Press, Inc.

1442A Walnut Street #312

Berkeley, CA 94709-1405

www.DecagonPress.com

Copyright © 2007 by Jonathan H. Dorfman

Cover Design: ©2007 Decagon Press, Inc.

Notice of Rights

All rights reserved. This work may not be translated, copied, transmitted, or reproduced in any way, in whole or in part, by any means, electronic, mechanical, photocopying, recording, or otherwise, or in any form of information storage or retrieval, without the express written permission of the publisher. For information on obtaining permission for reprints and excerpts, contact permissions@decagonpress.com.

Notice of Liability

The information in this book is distributed on an “As Is” basis, without warranty. Neither the author nor the publisher shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the instructions contained in this book, or by the software described in it.

Trademarks

The use in this work of trade names, trademarks, service marks, and similar terms, even if not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

MATLAB[®] is a trademark of The MathWorks, Inc. and is used with permission. The MathWorks does not warrant the accuracy of the text or exercises in this book. This book’s use or discussion of MATLAB[®] software or related products does not constitute endorsement or sponsorship by The MathWorks of a particular pedagogical approach or particular use of the MATLAB[®] software.

For MATLAB[®] product information, please contact:

The MathWorks, Inc.

3 Apple Hill Drive

Natick, MA, 01760-2098 USA

Tel: 508-647-7000 Fax: 508-647-7001

E-mail: info@mathworks.com

Web: www.mathworks.com

Preface

Why another book on MATLAB

The present text evolved from courses taught by the author at U. C. Berkeley in 2005-2006. The class was designed as an elective for students taking the upper-division Numerical Analysis sequence which covered introductory topics such as root-finding and interpolation, followed by Numerical Linear Algebra, and concluding with numerical methods for initial-value problems for systems of Ordinary Differential Equations. Since all course assignments required some component of MATLAB programming, the need existed for an efficient path enabling novice students to write functional, albeit rudimentary, programs within a few weeks. This circumstance motivated one of the guiding premises on which this text is based - the need to quickly develop sufficient competency in MATLAB programming to code a numerical homework assignment.

As there are many excellent introductions to MATLAB, not the least of which are provided by the MathWorks online tutorials, the above consideration does not, by itself, justify writing this text. With some exceptions, however, these texts cover *elements* of the language, and do not undertake a single serious theme of evolving complexity. In my courses, on the other hand, because students were focused on completing their first assignment (on applying Newton's root-finding algorithm to compute the square root of five), we had the benefit of a concrete context and goal for writing our first program. This circumstance motivated the second guiding premise of the present text - to write functional programs, albeit awkward and limited, from the start. Over time and after many iterations, our basic program evolves to an intermediate version capable of generating formatted tables and plots (e.g. for analyzing rates and orders of convergence). Though still limited to computing the square root of five, thanks to function handles and MATLAB's impressive vectorization capability, this implementation is shown to be one line away from a *general purpose root-finder for solving a system of nonlinear equations in n unknowns*.

For many, this course was their first introduction to writing software. During office hours in the Computer Lab, I had the opportunity to observe many students who were of the "MATLAB? - how hard can it be..." school of programming. Armed with a for-loop and display-statement, some students approached their assignments by generating scores of pages of numbers, perhaps unaware that no one would ever take the time to read a single digit. Others would attempt

to display a graph, but were at a loss when the plot resulted in a horizontal line with the y -axis labelled 10^{137} . With a little effort, these students could generate meaningful and self-documenting tables, quickly identify bugs in their code, and create professional looking plots with color and legends; they just needed someone to *show* them. And this brings us to the third guiding principle (and title) of this text, which I believe justifies its publication - beginning students should be taught the principles of *software design*. An experienced programmer learning MATLAB is likely to ask only two questions:

“Where’s the Help Documentation, and how do I access the Debugger?”,

but many students *never* learn to use the Debugger. These observations are not surprising to anyone who recalls the early days when Fortran ruled and programs were unstructured heaps of spaghetti code, with print statements sprinkled throughout for debugging.

In this spirit, the following principles of program design are developed throughout the course:

- Effectively using Help and Debugging tools
- Modularizing code into functional units, especially input, output, and algorithms
- Designing input functions which employ graphical user interfaces (GUI’s)
- Designing output functions which generate professional quality figures
- Designing algorithms whose performance can be quantified
- Designing algorithms whose applicability can be generalized using function handles
- Making programs portable so they are executable as stand-alone applications by third parties, and usable as functional components by other programmers
- Writing readable code by adding comments and by judicious use of intermediate variables and auxiliary functions, whose names are meaningful and self-documenting

In view of the above, I believe this book could be profitably used for a Computer Science course entitled “An Introduction to Programming”.

What is not covered

This is *not* a text for a course in Numerical Analysis. Indeed, many of the subtle issues that would make such a course scientifically challenging are intentionally ignored, for example the behavior of Newton’s algorithm near a critical point. The fact that we work exclusively with the *absolute error*, rather than with the numerically more meaningful *relative error*, should clarify the point of view taken.

Though some exercises introduce multidimensional ($p \times q \times r$)-arrays and a few advanced matrix operations (e.g. `lu`, `qr`, `eig`), substantive treatments of multilinear algebra (e.g.

`kron`, `reshape`, `repmat`), and canonical forms (e.g. `spdiags`) have been omitted. We also have intentionally ignored MATLAB's symbolic toolbox, which we consider to be of secondary interest (for reasons of performance) to numerical applications.

Future directions

In the bibliography we recommend three books for future study. These are suggested for further reading on each of the three fundamental directions developed in the course: Linear Algebra, Differential Equations, and User Interface Design. Each of these texts will be accessible to the student completing the present text.

Acknowledgements

Finally, I wish to thank Barbara Peavy - the very existence of the MATLAB course offered at Berkeley owes much to her commitment to the quality of the Mathematics Department's curriculum, not to mention her administrative wizardry. I also acknowledge with gratitude helpful feedback from Max Trokhimtchouk, who taught the MATLAB course over the Summer of 2007, and whose critical reading of early drafts is greatly appreciated. Additionally, I thank Robert Lyons and Professors Craig Evans and Beresford Parlett for their encouragement while I undertook this project.