Foundations of MATLAB Programming for Behavioral Sciences

This textbook introduces the fundamentals of MATLAB for behavioral sciences in a concise and accessible way. Written for those with or without computer programming experience, it works progressively from fundamentals to applied topics, culminating in in-depth projects. Part I covers programming basics, ensuring a firm foundation of knowledge moving forward. Difficult topics, such as data structures and program flow, are then explained with examples from the behavioral sciences. Part II introduces projects for students to apply their learning directly to real-world problems in computational modeling, data analysis, and experiment design, with an exploration of Psychtoolbox. Accompanied by online code and datasets, extension materials, and additional projects, with test banks, lecture slides, and a manual for instructors, this textbook represents a complete toolbox for both students and instructors.

Maxwell Mansolf is Assistant Professor of Team Science in the Department of Medical Social Sciences at the Feinberg School of Medicine, Northwestern University. As a team scientist, Dr. Mansolf is a co-investigator and collaborator on a broad array of research projects, including on child health and education, cognitive aging, healthcare worker well-being, personality, missing data, machine learning, and psychometrics, with over 40 peer-reviewed publications to date. For over six years, he has taught an undergraduate intro to MATLAB course at UCLA.

Foundations of MATLAB Programming for Behavioral Sciences

With Applications

Maxwell Mansolf
Northwestern University, Illinois

CAMBRIDGE UNIVERSITY PRESS

Shaftesbury Road, Cambridge CB2 8EA, United Kingdom

One Liberty Plaza, 20th Floor, New York, NY 10006, USA

477 Williamstown Road, Port Melbourne, VIC 3207, Australia

314–321, 3rd Floor, Plot 3, Splendor Forum, Jasola District Centre,
New Delhi – 110025, India

103 Penang Road, #05–06/07, Visioncrest Commercial, Singapore 238467

Cambridge University Press is part of Cambridge University Press & Assessment,
a department of the University of Cambridge.

We share the University's mission to contribute to society through the pursuit of
education, learning and research at the highest international levels of excellence.

www.cambridge.org
Information on this title: www.cambridge.org/highereducation/isbn/9781009350273

DOI: 10.1017/9781009350303

When citing this work, please include a reference to the DOI 10.1017/9781009350303

First published 2025

Cover image created by Maxwell Mansolf in MATLAB

A catalogue record for this publication is available from the British Library

A Cataloging-in-Publication data record for this book is available from the Library of Congress

ISBN 978-1-009-35027-3 Hardback
ISBN 978-1-009-35029-7 Paperback

Additional resources for this publication at www.cambridge.org/mansolf

Dedicated to:
My parents, to whom I owe everything.

Contents

Preface *page* xi
To the Student xvi

Part I MATLAB Programming

1 Basic MATLAB 3

1.1 The Design Background of MATLAB 3
1.2 Basics of Programming 6
1.3 Scripts 10
1.4 Script Style 12
1.5 Errors and Warnings 15
1.6 Getting Help 17
1.7 Conclusions 17
Exercises 18

2 Data Structures and Indexing 20

2.1 Vectors, Matrices, Scalars, Arrays, and Dimensions 20
2.2 Algebra on Arrays 26
2.3 Indexing 30
2.4 Conclusions 32
Exercises 33

3 Functions 36

3.1 Function Basics 36
3.2 Function Overloading 38
3.3 Some MATLAB-Specific Function Types and Options 43
3.4 Conclusions 45
Exercises 46

4 Logical Values 48

4.1 Logical Values 48
4.2 Logical Operators 49
4.3 Logical Indexing 53
4.4 Conclusions 55
Exercises 56

5 Conditionals and Loops 58

5.1 Background and Motivation 58
5.2 `if`, `elseif`, `else`, and `end` 58
5.3 The Other Conditional: `switch` 60
5.4 `while` Loops 61
5.5 `for` Loops 63
5.6 Nested Program Flow 65
5.7 `break` and `continue` 68
5.8 Conclusions 68
Exercises 69

6 Problem Solving with Conditionals and Loops 71

6.1 Means–End Analysis 71
6.2 Analogy 79
6.3 Conclusions 82
Exercises 82

7 Text 85

7.1 Text in MATLAB 85
7.2 Dialog: Prompts and User Responses 89
7.3 Application: Code as Text, Text as Code 91
7.4 Conclusions 94
Exercises 95

8 Other Data Structures 97

8.1 Higher-Dimensional Arrays 97
8.2 Cell Arrays 101
8.3 Names with Meaning: `table` and `struct` 102
8.4 Comma-Separated Lists 106
8.5 Beyond the Fundamentals: Classes, Properties, Methods 108
8.6 Conclusions 108
Exercises 109

9 Loading and Saving Data 111

9.1 How MATLAB Finds Things 111
9.2 MATLAB-Format Data (`.mat`) 114
9.3 Other Formats: Legacy Functions and Newer Alternatives 115
9.4 Conclusions 117
Exercises 118

10 User-Defined Functions 119

10.1 MATLAB Functions Revisited 119
10.2 Function Handles and Anonymous Functions 120
10.3 User-Defined Functions 121
10.4 Aside: Error Handling 124
10.5 Storing and Accessing Multi-Line Functions 125
10.6 Conclusions 126
Exercises 126

11 Graphing in MATLAB 128

11.1 A Graph 128
11.2 Figure Windows 130
11.3 Graphics Customization for `plot()` 131
11.4 Other Types of Graphs 138
11.5 MATLAB Graphics Architecture 142
11.6 Conclusions 145
Exercises 145

Part II Applications of MATLAB Programming in Behavioral Sciences

12 Computational Modeling 149

12.1 Principles and Methods 149
12.2 The Rescorla–Wagner Model 150
12.3 Ecological Simulation 159

13 Data Analysis 163

13.1 Tabular Data 163
13.2 Hierarchical Data 169
13.3 Statistical Modeling in MATLAB 173

14 Computerized Experiments 179

14.1 Experimental Design 179
14.2 Experimental Designs: `meshgrid()` and `combinations()` 180
14.3 Projects: MATLAB Experiments 182
14.4 Psychtoolbox 190

Glossary 207
Index 216

Preface

Purpose and Audience

This book is intended to teach MATLAB to undergraduate behavioral science students from a variety of backgrounds, including those who have never learned computer programming before. This is a unique and diverse demographic: motivated, intelligent, and hard-working, but often discouraged by lack of experience (or poor experiences) with similar material in the past, resulting in an internal sense that they will *never* understand computer programming. Others having taken a step back from technical material, feeling it beyond their reach. This textbook does not assume much existing technical knowledge, and is useful for anyone learning programming, or MATLAB, for the first time, including advanced undergraduate students, postdoctoral researchers, and faculty. Readers will learn how to use MATLAB data types, functions, and tools to analyze data, model behavior, and administer computerized experiments.

Philosophy

Existing texts on this topic are sadly burdened by what cognitive psychologists call the *curse of knowledge*: expertise so extensive that fundamentals are only briefly covered (after all, this is basic stuff, right?), and expertise in those fundamentals is assumed once the "real" course begins. Rushing through the basics leaves many students behind and, in my experience, causes much of the defeatist mindset that many students bring into my course. Of all the positive student evaluations I have received, the most precious follow the same structure: "After my previous experiences, I thought I'd never understand computer programming, but now I do". With this book, I hope I can bring this miracle to your classroom as well.

To this end, this textbook differs from similar texts in disproportionately emphasizing programming fundamentals. In particular, this textbook contains:

- several detailed introductory chapters to acquaint students with the basics of computer science, computer programming, and MATLAB, forming a foundation for the rest of the textbook;
- multiple chapters on the operation and application of loops and conditionals: crucial tools for any programmer, but often the most difficult concepts for new programmers to grasp;
- dedicated instruction on the theory of problem-solving as applied to computer programming; behavioral scientists may know some of this material already, but it becomes highly valuable as programming tasks become complex;
- exercises designed to provide near-immediate feedback by design, enhancing learning while reducing back-and-forth between student and instructor;

- *once students are ready*, numerous applications of MATLAB programming to behavioral science, focusing on problems they can solve right away with the fundamentals they've learned;
- a (gentle) introduction to Psychtoolbox, permitting students to design detailed computerized experiments using MATLAB.

Structure

In Part I of this book, you will learn fundamentals of computer programming in MATLAB, starting from an (almost) blank slate. Most of the problems will be simple and abstract, relying only on a basic knowledge of arithmetic and algebra, although problems will be tied to applications in the behavioral sciences when possible. The goal of Part I is to turn non-programmers into programmers, and programmers into better programmers. Two chapters in Part I are special: Chapter 1 introduces readers to computer sciences fundamentals and how to use the MATLAB interface, and Chapter 6 teaches problem-solving strategies critical for solving complex programming problems. Students and instructors should approach Part I in order, starting from the first chapter, and only deviating when absolutely necessary.

Once students have developed these programming skills, Part II explores several applications (**Projects**) of the tools learned in Part I to the behavioral sciences, including computational modeling, data analysis, design and implementation of experiments, and Psychtoolbox. Chapters in Part II can be approached in any order, and instructors should emphasize particular sections of relevance to themselves and/or their students if depth is preferred over breadth. Supplemental **Projects**, extensions, and background material are available in the course's webpage, located at www.cambridge.org/mansolf.

Textbook Features

Exercises

Learning computer programming requires practice. At the same time, "just trying to program" without knowing how the pieces work is a fool's errand, so some didactic preparation for each programming task is essential. Thus, while it is important to read and study each chapter's content, it is more important to engage seriously in the **Exercises** at the end of each chapter because solving these problems is where most learning occurs. The foundations of computer programming for MATLAB can be read in an afternoon, and probably retained fairly well, but mastering these tools will require applying them to a variety of problems. There is simply no substitute for this, just as there is no substitute for real-world experience flying a plane or riding a bicycle.

Real-world problems won't be framed in terms of programming concepts; your future employer will not ask you to write a `for` loop with three distinct conditional statements, they will instead want the new features added by Thursday. The exercises in this book will teach you how to translate real-world problems into tasks MATLAB can solve, and then solve them.

Common sense and research are clear that immediate feedback is beneficial for learning, so **Exercises** are structured such that once an answer is obtained, you as the student *know*, or at least

have high confidence, whether your code is correct. Take the exercises in this book seriously and you will be rewarded with mastery of MATLAB and of the foundations of programming in countless other languages.

Skills Acquired

MATLAB and computer programming require some concrete knowledge, but fundamentally they are *skills* to apply to get things done. Each chapter will begin with a list of skills you will learn therein. A demonstration of this list is provided for the skills you will learn in this textbook.

Use this textbook effectively	(Preface)
Use MATLAB effectively	(Part I)
Model behavior	(Chapter 12)
Analyze data	(Chapter 13)
Conduct experiments	(Chapter 14)

Within Chapters

To support the course content and enhance the effectiveness of **Exercises**, this text includes several content features interspersed within each chapter.

First, because computer programming can be disorienting, this textbook calls out important aspects directly; **IMPORTANT!** boxes signal information that is especially critical for optimal use of MATLAB, including ways to avoid potential pitfalls and maximize the use of MATLAB features. Here is an example:

> **IMPORTANT!** Like a math class, programming is cumulative. If you have trouble, reviewing previous material can help tremendously.

Larger features will be used for more detailed instruction, and their frequency will change throughout the text to adapt to content. For stand-alone programming topics, **EXPLAINER**s (e.g., Box 0.1) provide detailed overviews of such content, and will appear throughout the text. **HOW TO** boxes (e.g., Box 0.2) provide details on use of the MATLAB interface and programming in general; in Chapter 1, these boxes will be the most common, while for the rest of the textbook they will appear as needed when new features become introduced or especially relevant.

Lastly, **EXAMPLE**s will be used to demonstrate how to solve programming problems, but more importantly, these **EXAMPLE**s will demonstrate how to *approach* solving programming problems, detailing the steps needed to reach solutions. As Chapter 1 includes relatively little programming, **EXAMPLE** boxes will be sparse in this chapter, but will appear regularly throughout the remaining text, especially in Part I when **EXAMPLE**s will be crucial starting points for some of the more difficult exercises.

For all larger features, cross-references within the text will be used to indicate where this material fits in to the main; for example, Box 0.1 is primarily used to illustrate how these larger features look, and can be reviewed for more information on how this textbook came to be.

Box 0.1 EXPLAINER: Origin of This Textbook

Teaching technical topics to behavioral science students (in my case, psychology undergraduates) is tricky. For some students, programming is a natural extension of their interests, while for others it fulfills a graduation requirement or helps achieve a desirable research placement. I began programming before starting my undergraduate studies, and when it came time to teach it to others, I quickly learned that students like myself are the exception rather than the rule. The variety of students' backgrounds in computer programming made the difficulty and pace of the material perhaps the biggest challenge: how do I start from the basics, get students to a real working comprehension of the material, and not take the years it took me to get there on my own?

The psychological dissonance of non-technical students being compelled to learn technical material complicates the problem further. At best, students are highly motivated but are navigating a novel and bizarre landscape of seemingly endless terminology and complex, opaque systems. At worst, they have already been through this ordeal a few times and it went poorly, leaving them jaded, bitter, and with a deep internalized feeling that the material is simply beyond their ability. As someone who grew up in a rural area and never "got" people (this probably piqued my interest in psychology), I understood their suffering and wanted to give them a fresh start.

This course is the culmination of more than a decade satisfying these competing needs in teaching technical material, here computer programming, to behavioral science students. Such material needs to start from the very basics without boring the advanced students, and needs to teach people how to think in new ways, quickly, without leaving them behind or burning them out or reopening old scars, all while staying true to the subject matter. Past undergraduate students, graduate students, postdoctoral researchers, and faculty that have used my materials can attest that the approach implemented in this textbook is successful in these goals.

Companion Materials

Outside of the print/e-book, accompanying this textbook is a variety of supplemental materials to facilitate administration and student learning. Some examples are given here.

- The Open Science Foundation (OSF) repository, located at https://osf.io/mzq2f/. Here, students can find data files and sample scripts for complex examples and **Projects**. *For the most part, students will not interact with OSF directly*; instead, when these files are needed in code, code which directly uses these files will be included in text, such that students can run this code and it will "just work", loading the requisite data into MATLAB.
  - OSF is designed to store data forever, but if something goes wrong and code doesn't "just work", all data and code are also available at the Cambridge University Press companion webpage for this textbook.
- The companion webpage for this textbook (www.cambridge.org/mansolf) also includes:
  - for instructors, an instructor manual, lecture slides and a test bank
  - for each **Project** in Part II

    - additional background material for interested students to learn more
    - one or more **Extensions** for further study
- for each chapter in Part II, one or more additional **Projects** from behavioral sciences.

The companion webpage allows instructors to tailor their instruction, focusing on a particular MATLAB application or topic area and/or providing extra credit opportunities. Students can also engage with these materials to expand their MATLAB skills beyond the scope of a structured course.

To the Student

To the student reading this, I want to start by addressing the obvious question: *"Why am I learning computer programming?"* Motivation is important for learning, and there are good reasons to be highly motivated to learn computer programming. If you aim to do research, either in academia or industry, computers are increasingly becoming the main tool, and in some cases the only viable tool, to do so. Behavioral scientists write programs to design experiments, model psychological processes, and analyze and present data. Even if novel programs are not written to perform these tasks every time, they often involve using programs that others have written, which means programming is still involved. Proper use of programs requires knowing how they work, and this understanding can help everyone become a better user of technology. On the more inspirational end, we live in the age of data and code sharing, where billions of lines of code are at your fingertips on websites like GitHub, StackExchange, and MATLAB code sharing resources. If you can use, modify, and understand programs others wrote, you will essentially be able to do anything they can do. Needless to say, this grants you immense power.

What if you don't want to sit in front of a computer for your entire working life? Technical skills like computer programming are *still* worth learning. Every new practical skill you learn opens new career opportunities because it "stacks" on your existing skill set, yielding combinations and interactions that may make you uniquely skilled for a niche (which often means "lucrative") career. Good at working with kids *and* know how to program? Computer programming opens doors to game design, hybrid and interactive instruction, and virtual reality and simulation. Only interested in behavioral research? Increasingly, the tools to analyze and visualize data require computer programming, and having access to those tools will give you an edge over your peers in applications to jobs, grants, and manuscript submissions to journals. In short, adding computer programming to your skill set gives you the ability to augment *any* job with a technical savvy that will impress your peers, supervisors, and clients, and can open doors you did not know existed. This is especially important for students in the behavioral sciences, where in most career paths there are more qualified applicants than available positions.

Technical skills are also a great way for students from marginalized backgrounds to overcome discrimination. Sociologists and psychologists have found that the presence of certain characteristics on job applications, for example being a mother, can lead to lower likelihoods of being selected for a position, all else considered. However, this is not the whole story: a critical, but often overlooked, finding of this research is that the *more* information is presented on the application, and the more other factors such as skills tests are weighted in applicant selection, the less these immutable characteristics influence hiring and promotion decisions. Obtaining technical skills is a great way to set yourself

apart from your competitors, as hiring managers, future supervisors, and coworkers can overcome their biases when *objective* metrics, including mastery of skills like computer programming, are included as criteria for hiring. In short, for all students, but especially those who may bear the burden of discrimination, skills training can give you a competitive edge and help you enter and advance in your chosen career.

Code versus Text and Other Notation

When learning programming, you will encounter a lot of code. Therefore, when learning programming from a textbook, it is helpful to have a visual cue to indicate whether you are reading code or prose (regular text). In this book, computer code is presented in a *fixed-width* font, where each typeset character (letter, number, space, punctuation mark, etc.) has the same width. `This sentence, for example, is written in fixed-width font; notice that thin letters like ``i'' and ``t'' have a lot of extra space, unlike regular text.` This helps visually distinguish computer code from regular text, which is useful as a didactic tool and in general; you will see this convention used within MATLAB and nearly anywhere else you see code. Functionally, the MATLAB programming system uses `fixed-width` font to indent sections of code in a way that makes it easier to understand complex structures like functions (Chapter 10) and conditionals and loops (Chapter 5). In this book, whenever text is properly interpreted as MATLAB code or output, it will be presented in `fixed-width` font.

A few other notational conventions are used throughout this book.

- MATLAB uses many punctuation symbols, such as the comma (`,`), semicolon (`;`), and period (`.`), and mathematical operators like plus (`+`) and minus (`-`). As was done here, these characters will be presented in parentheses and `fixed-width` font to help distinguish them from regular punctuation. Characters used as part of operations in paragraphs (e.g., `x+1`) will generally not be in parentheses, except when helpful to distinguish them from regular text.
- Keyboard keys will be denoted in standard text. A (+) in a key combination, such as Ctrl+C to copy text, means to press both keys (here, Ctrl and C), not to press (+). Letters are presented as capitals but you do not need Shift unless otherwise instructed.
- In many places, especially in Chapter 1 and Chapter 10, I will indicate specific aspects of the MATLAB interface, including tab titles, buttons, and so on. These will be presented in **bold** font.
- Text in **bold font** is also used to denote key terms and book features, and the **Glossary** at the end of this text will provide stand-alone definitions of these terms.
- MATLAB has several keywords which, in the MATLAB text editor with default formatting, appear in `fixed-width` blue font. Most of these keywords will be introduced in Chapter 5 and Chapter 10. Here, these keywords will be presented in **`bold, fixed-width`** throughout this text to draw attention to these critical programming tools.

- Lastly, once each **Exercise** is solved, your result will be stored in MATLAB variables (subsection 1.2.2). Use the variable name(s) provided in parentheses at the end of the exercise. For example, if you compute 1+1 and had `(my_Variable)` listed at the end of the exercise (as shown here), store the result of this operation in a variable named `my_Variable`. `(my_Variable)`
  - If multiple variables are produced, they'll be separated as in a sentence. `(v1)`, `(v2)`, and `(v3)`.
  - When subsequent problems use an existing solution, for example if we want to modify a copy of `(v1)`, this variable will also be presented with parentheses.
  - If there is other "evidence" that code worked, like a graph (Chapter 10), you'll see (N/A) instead.

How to use Companion Materials

This course comes with two online resources: an Open Science Foundation (OSF) repository, located at https://osf.io/mzq2f/; and the Cambridge University Press companion webpage, located at www.cambridge.org/mansolf. For the OSF, ideally you won't need to interact with that link directly. Instead, when data, images, or scripts are needed, direct links to those resources will be provided, such that you can copy the code into MATLAB and it will "just work". OSF is designed to store data forever, but if something goes wrong and code doesn't "just work", all data and code are also available at the companion webpage. For Part II when we apply MATLAB to real-world problems in behavioral sciences, the companion page also contains additional background on each **Project**, optional **Extensions** to each **Project**, and one or more additional **Projects** for each application area (computational modeling, data analysis, experimental design). Your instructor may assign these materials as part of the course, or feel free to explore them to further expand your MATLAB skills.

Self-Care When Programming

Be patient with yourself. Computer programming requires thinking like a computer, which is disorienting for most people because, as any behaviorist knows, humans are not computers. This is not easy stuff. I have spent months debugging a single tiny piece of code, and students have told me some problems took them several hours across multiple days. This can be frustrating, but is completely normal: people are not computers, and in undertaking computer programming you are in some sense reaching beyond the limits of your humanity. If you are successful, great power lies beyond, so try not to become discouraged when it gets difficult. Box 0.2 recommends some habits to make computer programming more enjoyable for humans like us.

Box 0.2 HOW TO: Be kind to yourself

There are some habits that good programmers use to stay sane.

- *Take breaks.* Many programming problems are solved when you are not programming. Moving your attention elsewhere, even for a few minutes, can spark the creativity to solve a difficult problem, or provide the rest to finish a lengthy problem.
- *Return to difficult problems* if you find yourself stuck, and address other, easier problems first. Similarly, return to difficult parts of problems if you get stuck on them, addressing other parts of the same problem first. Often, putting together one piece of code can give you the crucial insight needed to solve a more difficult problem.
- *First, do what you can.* If a problem seems daunting, see if you can solve a piece of it, even a small one. Once you have that piece, it becomes easier to build toward the larger solution. We will practice this strategy, *means–end analysis*, and others in Chapter 6.
- *Don't neglect your physical well-being.* Move around every hour or so, if you can, or at least do something else for a bit. Drink plenty of fluids and don't go hungry. It can be easy to get "tunnel vision" and ignore other important things in your life.

By following these principles, you can avoid associating programming with discomfort and frustration. Programming can be difficult, but that doesn't mean it can't be fun.

Enjoy,

Maxwell Mansolf
Northwestern University, Feinberg School of Medicine