Foundations of Computational Finance with MATLAB®

ED McCARTHY

WILEY

Foundations of Computational Finance with MATLAB®

Foundations of Computational Finance with MATLAB®

Ed McCarthy

WILEY

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

Limit of Liability/Disclaimer of Warranty

Library of Congress Cataloging-in-Publication Data

Names: McCarthy, Ed (Edward), 1955– author.
Title: Foundations of computational finance with MATLAB / by Ed McCarthy.
Description: Hoboken, New Jersey : John Wiley & Sons, Inc., [2018] | Includes index. |
Identifiers: LCCN 2018014808 (print) | LCCN 2018016054 (ebook) | ISBN 9781119433873 (epub) | ISBN 9781119433910 (pdf) | ISBN 9781119433859 (cloth)
Subjects: LCSH: Finance—Mathematical models. | Finance—Data processing.
Classification: LCC HG106 (ebook) | LCC HG106 .M396 2018 (print) | DDC 332.0285/53—dc23
LC record available at https://lccn.loc.gov/2018014808

Cover Design: Wiley
Cover Image: © monsitj/iStockphoto

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

To my wife, Diane

Contents

Introduction xiii

Why You Should Read This Book xiii
The Intended Reader xiv
Why MATLAB®? xiv
How to Use This Book xvi
Font Conventions xvi
About the Author xvii
MathWorks Information xviii
References xviii

Part I MATLAB Conventions and Basic Skills 1

Chapter 1 **Working with MATLAB® Data** 3

1.1 Introduction 3
1.2 Arrays 3
1.2.1 Numerical Arrays 4
1.2.2 Math Calculations with Scalars, Vectors, and Matrices 10
1.2.3 Statistical Calculations on Vectors and Matrices 16
1.2.4 Extracting Values from Numerical Vectors and Matrices 19
1.2.5 Counting Elements 26
1.2.6 Sorting Vectors and Matrices 28
1.2.7 Relational Expressions and Logical Arrays 31
1.2.8 Dealing with NaNs (Not a Number) 35
1.2.9 Dealing with Missing Data 39
1.3 Character Arrays 40
1.3.1 String Arrays 44

1.4 Flexible Data Structures 46
1.4.1 Cell Arrays 47
1.4.2 Structure ("struct") Arrays 49
1.4.3 Tables 51
References 60
Further Reading 60

Chapter 2 **Working with Dates and Times** 61
2.1 Introduction 61
2.2 Finance Background: Why Dates and Times Matter 61
2.2.1 First Challenge: Day Count Conventions 62
2.2.2 Second Challenge: Date Formats 63
2.3 Dates and Times in MATLAB 64
2.3.1 Datetime Variables 64
2.3.2 Date Conversions 73
2.3.3 Date Generation Functions with Serial Number Outputs 79
2.3.4 Duration Arrays 83
2.3.5 Calendar Duration Variables 86
2.3.6 Date Calculations and Operations 89
2.3.7 Plotting Date Variables Introduction 94
References 95

Chapter 3 **Basic Programming with MATLAB®** 97
3.1 Introduction 97
3.1.1 Algorithms 101 97
3.1.2 Go DIY or Use Built-In Code? 98
3.2 MATLAB Scripts and Functions 99
3.2.1 Scripts 99
3.2.2 Developing Functions 106
3.2.3 If Statements 112
3.2.4 Modular Programming 115
3.2.5 User Message Formats 121
3.2.6 Testing and Debugging 124
References 127

Chapter 4 **Working with Financial Data** 129
4.1 Introduction 129
4.2 Accessing Financial Data 129
4.2.1 Closing Prices versus Adjusted Close Prices for Stocks 130
4.2.2 Data Download Examples 131
4.2.3 Importing Data Interactively 133
4.2.4 Automating Data Imports with a Script 138
4.2.5 Automating Data Imports with a Function 140
4.2.6 Importing Data Programmatically 147
4.3 Working with Spreadsheet Data 154
4.3.1 Importing Spreadsheet Data with Import Tool 154
4.3.2 Importing Spreadsheet Data Programmatically 154
4.4 Data Visualization 156
4.4.1 Built-In Plot Functions 156
4.4.2 Using the Plot Tools 158
4.4.3 Plotting with Commands 159
4.4.4 Other Plot Tools 162
4.4.5 Built-In Financial Charts 173
References 176

Part II Financial Calculations with MATLAB 177

Chapter 5 **The Time Value of Money** 179
5.1 Introduction 179
5.2 Finance Background 180
5.2.1 Future Value with Single Cash Flows 180
5.2.2 Future Value with Multiple Cash Flows 185
5.2.3 Present Value with Single Cash Flows 187
5.2.4 Present Value with Multiple Variable Cash Flows 188

5.3 MATLAB Time Value of Money Functions 189
5.3.1 Future Value of Fixed Periodic Payments 190
5.3.2 Future Value of Variable Payments 191
5.3.3 Present Value of Fixed Payments 193
5.3.4 Present Value of Variable Payments 194
5.4 Internal Rate of Return 197
5.5 Effective Interest Rates 198
5.6 Compound Annual Growth Rate 198
5.7 Continuous Interest 200
5.8 Loans 200
References 202

Chapter 6 **Bonds** 203
6.1 Introduction 203
6.2 Finance Background 204
6.2.1 Bond Classifications 204
6.2.2 Bond Terminology 205
6.3 MATLAB Bond Functions 206
6.3.1 US Treasury Bills 206
6.3.2 Bond Valuation Principles 208
6.3.3 Calculating Bond Prices 209
6.3.4 Calculating Bond Yields 212
6.3.5 Calculating a Bond's Total Return 214
6.3.6 Pricing Discount Bonds 216
6.4 Bond Analytics 216
6.4.1 Interest Rate Risk 217
6.4.2 Measuring Rate Sensitivity 219
6.4.3 Yield Curves 227
6.5 Callable Bonds 229
References 231
Further Reading 231

Chapter 7 **Dealing with Uncertainty and Risk** 233
7.1 Introduction 233
7.2 Overview of Financial Risk 234
7.3 Data Insights 234
7.3.1 Visualizing Data 235
7.3.2 Basic Single Series Plots 237

7.3.3 Basic Multiple Series Plots 237
7.3.4 Adding Plot Customization 238
7.3.5 Histograms 239
7.3.6 Measures of Central Location 241
7.3.7 Measures of Data Dispersion 243
7.4 Data Relationships 249
7.4.1 Covariance and Correlation 251
7.4.2 Correlation Coefficients 252
7.5 Creating a Basic Simulation Model 253
7.6 Value at Risk (VaR) 258
References 261
Further Reading 262

Chapter 8 **Equity Derivatives** 263
8.1 Introduction 263
8.2 Options 264
8.2.1 Option Quotes 265
8.2.2 Market Mechanics 266
8.2.3 Factors in Option Valuation 267
8.3 Option Pricing Models 268
8.3.1 Arbitrage 269
8.3.2 Binomial Option Pricing 270
8.3.3 Black-Scholes 274
8.4 Options' Uses 276
8.4.1 Hedging 277
8.4.2 Speculation and Leverage 277
8.4.3 Customizing Payoff Profiles 278
8.5 Appendix: Other Types of Derivatives 279
8.5.1 Commodity and Energy 279
8.5.2 Credit 279
8.5.3 Exotic Options 280
References 281
Further Reading 281

Chapter 9 **Portfolios** 283
9.1 Introduction 283
9.2 Finance Background 283
9.3 Portfolio Optimization 285
9.4 MATLAB Portfolio Object 286

9.4.1 Object-Oriented Programming (OOP) 286
9.4.2 A Basic Example 287
9.4.3 Using Data Stored in a Table Format 294
References 296

Chapter 10 **Regression and Time Series** 297
10.1 Introduction 297
10.2 Basic Regression 297
10.2.1 Understanding Least Squares 300
10.2.2 Model Notation 301
10.2.3 Fitting a Polynomial with polyfit and polyval 303
10.2.4 Linear Regression Methods 305
10.3 Working with Time Series 308
10.3.1 Step 1: Load the Data (Single Series) 308
10.3.2 Step 2: Create the FTS Object 309
10.3.3 Step 3: Using FTS Tools 311
References 314

Appendix 1 **Sharing Your Work** 315
A1.1 Introduction 315
A1.2 Publishing a Script 316
A1.2.1 Publishing with Code Sections 317
A1.2.2 futureValueCalc3 319
A1.2.3 Formatting Options 321
A1.2.4 Working with Live Scripts 322
A1.2.5 Editing and Control 325
References 326

Appendix 2 **Reference for Included MATLAB® Functions** 327

Index 335

Introduction

Why You Should Read This Book

If you're planning a career in corporate or investment finance or already working in one of those areas, you're probably proficient with financial calculators and spreadsheets. Those technologies have proven their value, and it's likely they will remain essential tools for many years. (I still use a 30-year-old Hewlett Packard 12C calculator regularly and it works perfectly, albeit a bit slower than newer models.)

But the nature of data and analytics are changing, and those changes are influencing financial analysis and management. Traditional financial data still drive decisions, but those data are being supplemented by increasing volumes of nontraditional information and new computational tools. Consider these headlines from recent years, which are just a small sample of the articles on these themes:

- "Stop Using Excel, Finance Chiefs Tell Staffs" (*Wall Street Journal*, 1/22/1017)
- "The Quants Run Wall Street Now" (*Wall Street Journal*, article series in May 2017)
- "At New Digital Lenders, Math Rules" (*New York Times*, 1/19/2016)
- "Leveraging Data to Own the Engaged Customer" (Utility Analytics, 11/4/2015)
- "The Morning Ledger: The Rising Profile of Financial Planning and Analysis" (*Wall Street Journal*, 12/22/2015)
- "How Computers Trawl a Sea of Data for Stock Picks" (*Wall Street Journal*, 4/1/2015)
- "As Big Data and AI Take Hold, What Will It Take to Be an Effective Executive?" (*Wall Street Journal*, 1/23/2015)

I believe this paradigm shift requires a new approach to financial analysis and management. Specifically, finance professionals must supplement their calculators and spreadsheets with more flexible and powerful computational platforms. These platforms can work with the new data models while still providing the tools needed for traditional financial analysis. As the headlines suggest, remaining competitive in financial analysis and management will require an understanding of and skill with computational finance. This knowledge will allow you to access data from multiple sources, develop customized financial analytics, and then distribute your tools and findings across a variety of platforms.

The Intended Reader

Transitioning to the new paradigm is a challenge, though, because it means learning about computational finance. Other authors have addressed this topic, but they focused on advanced material for readers who combine extensive math, statistics, programming, and finance backgrounds, such as financial engineers and academics.

In contrast, I wrote this book for readers seeking an introductory text that links traditional finance material to the MATLAB computational platform. This includes upper-level undergraduate finance students, graduate students, finance practitioners, and those with STEM backgrounds seeking to learn about finance. My assumption is that your background will be: (1) A business student or finance professional who is comfortable with finance theory but has modest computer programming experience beyond spreadsheets, or (2) A STEM student or professional who has a more extensive programming background but less experience with finance.

I'm also assuming you have completed first courses in linear algebra and statistics and will have access to MATLAB and the required MATLAB Toolboxes. Many universities have MATLAB licenses, but if you must buy the software, it's very inexpensive for students, and the MATLAB Home edition makes it readily affordable for nonacademic users. (Pricing details are available on the mathworks.com site.)

Why MATLAB®?

That's a fair question, because there are a host of programming languages being used in finance. But there's a question-and-answer

dialogue I've seen numerous times on web message boards for quantitative and computational finance that helps answer the question. It goes something like this:

Q. I'm thinking of getting into quantitative finance (or applying to a quant educational program) and need advice on programming languages. Should I start with MATLAB or Python? R or S? C++ or Java?

A. Yes.

The answer is a bit snarky, so the respondent usually explains that learning a programming language is not a one-and-done lifetime proposition. People change employers during their careers and the new employer might emphasize a different language. Computer technologies and programming languages evolve, too, and it's necessary to keep up with those changes, as those of us who started programming with punched cards and card readers can attest.

I have no business affiliation with The MathWorks but I believe the MATLAB software is well-suited for an introduction to computational finance for several reasons:

- It's an integrated development environment that combines a code editor, compiler, debugger, interpreter, and graphics capability in a well-designed graphical user interface.
- It's relatively easy to develop basic MATLAB skills. Of course, it takes time and effort to learn any computer language but the program's consistent syntax usage and extensive documentation improve user productivity.
- The finance-related MATLAB Toolboxes provide access to multiple financial functions running tested algorithms, which can save many programming hours and much frustration. Additional MATLAB Toolboxes can make it easier to move into other areas, such as big data analytics, as well.
- MATLAB is used in numerous financial firms, other industries, and over 5,000 universities. If you're a student, your school probably has a MATLAB license.
- Prices for students and educators have always been low, and several years ago The MathWorks began offering inexpensive personal licenses.
- Users can access multiple training and support channels through general and specialized books, online and live

training courses, and formal and informal (community) support resources. I've completed several of the online training programs offered by The MathWorks, and they were very good.
- Finally, I believe the knowledge and skills developed in learning MATLAB make it easier to subsequently learn other programming languages.

How to Use This Book

Part I introduces the MATLAB syntax and how to use the program. If you're new to MATLAB or need a review, start with those chapters. For a deeper introduction, you can supplement that material with the resources online The MathWorks offers, including the no-cost MATLAB Onramp course at matlabacademy.mathworks.com. That course uses an interactive format and takes about two hours to complete. Other online tutorials can be found at www.mathworks.com/support/learn-with-matlab-tutorials.html. If you have the time and funds, the MATLAB Fundamentals course is an excellent in-depth introduction.

Part II demonstrates how MATLAB can be used as a computational platform in finance. The material in Chapter 5, "The Time Value of Money," has general applications throughout the remaining chapters, so I suggest reviewing that material. The text reviews the underlying finance material being discussed in each chapter and includes suggestions for further reading.

Finally, practice using the program interactively or programmatically by entering commands in the MATLAB Command window as you work through the examples. Learning to use software is somewhat like learning to drive. Reading a book on safe driving gives you an intellectual perspective but it makes driving sound deceptively easy. Coding—like getting behind the steering wheel and pulling into high-speed traffic for the first time—is best experienced hands-on. Fortunately, writing code is a lot less nerve-wracking than highway driving.

Font Conventions

The book uses several different font styles to help you distinguish the material:

Bold: Function names, reserved keywords, matrices, and vectors

Monospaced italic: Command window inputs. Example:

```
x = 7
```

`Monospaced:` MATLAB output and responses. Example:

```
x =
     7
```

`Monospaced starting with %`: Code comment lines that do not execute

Normally spaced lines starting with %: Text comments

About the Author

I have worked as a freelance finance writer since the mid-1980s, and during that time I have written for many of the financial service industry's leading publications. These include *Bloomberg Wealth Manager, CFA Institute Magazine, Institutional Investor online, Financial Planning, Journal of Accountancy,* and the *Journal of Financial Planning.* Earlier in my career I published a technology book for financial advisors, *The Financial Advisor's Analytical Toolbox* (Irwin), and one for consumers, *Fast Forward MBA in Personal Finance* (Wiley). I have also written numerous print and web articles for custom publishers and many of the largest U.S. and international financial services firms. My primary experience as a writer and the focus for many of my articles has been explaining complex finance topics and technologies to readers.

My first exposure to MATLAB was in the mid-1990s when I was doing research for my first book, which included a discussion of the software's financial modeling capabilities. My use of the program intensified while I was studying for a PhD in finance, and I believe my experience at that time supports the premise for this book. The lack of available resources to link finance theory with the requisite computer programming made that aspect of the work more difficult than it needed to be. I chose not to finish my dissertation and left school to write full-time, but I continued to use the software and periodically work through new financial mathematics and MATLAB texts to stay current. I am a MathWorks Certified MATLAB Associate and am working toward The MathWorks Certified MATLAB Professional designation.

MathWorks Information

The material in this book was developed using the MATLAB R2016B, 2017A, and 2017B releases and MATLAB Toolboxes for the same releases.

For MATLAB and Simulink 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: mathworks.com

How to buy: www.mathworks.com/store

References

Hope, Bradley. "How Computers Trawl a Sea of Data for Stock Picks." *Wall Street Journal*, April 1, 2015. Accessed January 15, 2016. https://www.wsj.com/articles/how-computers-trawl-a-sea-of-data-for-stock-picks-1427941801.

Lohr, Steve. "At New Digital Lenders, Math Rules." *New York Times*, January 19, 2016. Accessed January 20, 2016. https://bits.blogs.nytimes.com/2016/01/19/at-new-digital-lenders-math-rules/.

Shumsky, Tatyana. "Stop Using Excel, Finance Chiefs Tell Staffs." *Wall Street Journal*, January 22, 2017. Accessed January 22, 2017. https://www.wsj.com/articles/stop-using-excel-finance-chiefs-tell-staffs-1511346601.

Willhite, James. "The Morning Ledger: The Rising Profile of Financial Planning and Analysis." *Wall Street Journal*, December 22, 2015. Accessed January 15, 2016. https://blogs.wsj.com/cfo/2015/12/22/the-morning-ledger-the-rising-profile-of-financial-planning-analysis/.

Wladawsky-Berger, Irving. "As Big Data and AI Take Hold, What Will It Take to Be an Effective Executive?" *Wall Street Journal*, January 23, 2015. Accessed February 1, 2016. https://blogs.wsj.com/cio/2015/01/23/as-big-data-and-ai-take-hold-what-will-it-take-to-be-an-effective-executive/.

Zuckerman, Gregory and Bradley Hope. "The Quants Run Wall Street Now," *Wall Street Journal*. May 21, 2017. Accessed June 1, 2017. https://www.wsj.com/articles/the-quants-run-wall-street-now-1495389108.

Foundations of Computational Finance with MATLAB®

PART I

MATLAB Conventions and Basic Skills

CHAPTER 1

Working with MATLAB® Data

1.1 Introduction

MATLAB® is an abbreviation of "matrix laboratory," and while the ability to work with matrices is still an essential part of the program, the software also works with numerous other data types. This chapter examines several of the different data types you are likely to encounter and the functions needed to manipulate them.

This material and the subsequent chapters assume you know how to open MATLAB, enter commands in the Command Window, and create and identify variables and their types in the Workspace. If you lack those skills, consider working through the MATLAB Onramp training program, which is available free online in The MathWorks® MATLAB Academy (matlabacademy.mathworks.com) and takes just a few hours to complete.

Key concepts introduced in this chapter include:

- MATLAB array types
- Flexible data structures

Software required for this chapter: MATLAB base program.

1.2 Arrays

An array is a data series arranged in rows and columns. The usual notations to denote the number of rows and columns are $r \times c$ (rows

Table 1.1 MATLAB Data Terminology

Term	Size	Example
Scalar	1×1 (1 row by 1 column)	7
Row vector	$1 \times n$ (1 row by n columns)	[1 2 3]
Column vector	$m \times 1$ (m rows by 1 column)	$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$
Matrix	$m \times n$ (m rows by n columns)	$\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$

by columns) or $m \times n$ (also signifying rows by columns). Table 1.1 shows the MATLAB terminology used to distinguish arrays.

The term *array* in MATLAB is potentially confusing because the program allows for aggregating multiple data types in arrays, so it's often easiest to think of an array as a container for holding multiple values in one variable (except for scalars, which have one value). In some instances, those values are of the same type: numbers or characters (letters, for example), but other array types can hold different value types within one variable. Table 1.2 summarizes the more common array types; subsequent sections cover each type in more detail.

1.2.1 Numerical Arrays

Recall that scalars in MATLAB are 1×1 arrays, row vectors are $1 \times n$, and column vectors measure $m \times 1$. You can create a scalar by entering a numeric value at the input prompt (all inputs shown in italic):

```
7
ans =
     7
```

Usually, it's more practical to assign the input to a variable so you can reuse it:

```
x = 7
x =
     7
```

Table 1.2 MATLAB Array Types

Array Type	Description	Example
Cell	Cells can contain any data type including strings, numbers, or combinations of the two.	Row vector cell array: {1, 'a', 'text', 1:10} 2 × 2 cell array matrix: {1, 'a'; 'text', 1:10}
Character	Sequence of characters, typically short pieces of text	'*a b c*'
Dates and times	Used to represent dates, times, and durations. Covered in Chapter 2.	Datetime('1-Jan-2016') Duration(6,1,15) [6 hours, 1 minute, 15 seconds]

Logical	False (0) or true (1) values in response to a logical evaluation of a relationship ($x > y$, for example)	`val = 5 < 3`
		`val =`
		`logical`
		`0`

String	Stores text	"Hello" (note use of double quote marks versus singles for character array)
Structure	Groups logically related data into data containers called fields. Each field can contain any data type.	Structure Name: Employee Fields: Employee.LastName Employee.FirstName Employee.HireDate

Note that you can't enter a character by itself because MATLAB won't recognize it:

```
a
Undefined function or variable 'a'
```

You can enter `'a'` within single quote marks and it will be assigned to the ans variable, but again, that's not very useful for later reference. It's generally good practice to create named variables with your work to avoid retyping the data, should you need to reuse them later.

The variable-naming rules for MATLAB are straightforward:

- Start with a letter.
- Use only letters, numbers, and underscores.
- Keep the name's length under 63 characters.

This book's convention will be to use mixed cases with variables and functions whenever it's practical. In those instances, names will begin with a lowercase letter. Insert an uppercase letter for improved readability if the name contains two or more words:

```
myVariable = value
```

MATLAB provides several methods for creating vectors from the Command window. Enter the numbers within square brackets for a row vector:

Row Vector

```
x = [1 2 3 4]
x =
     1     2     3     4
```

Column Vector

Separate the elements with semicolons or add an apostrophe to transpose a row vector:

```
x = [1; 2; 3; 4]
x =
     1
     2
     3
     4
or

x = [1 2 3 4]'
x =
     1
     2
     3
     4
```

Matrix

The vector-creation techniques also apply to matrices. Put square brackets around the elements and separate rows with semicolons. Here's a 2 × 3 example:

```
x = [1 2 3; 3 4 5]
x =
     1     2     3
     3     4     5
```

The transpose operator functions the same way as with vectors:

```
x'
x =
     1     4
     2     5
     3     6
```

Concatenation

You can concatenate (join) compatibly sized vectors to create matrices by enclosing the vectors in square brackets. Also, note the text's use of comment lines that begin with %, which is the same syntax for MATLAB comments. Comment lines do not run as commands or inputs—their purpose is to provide documentation for users.

```
a = [1 2 3];
b = [4 5 6];
% Horizontal concatenation
c = [a b]
c =
     1     2     3     4     5     6

% Vertical concatenation
d = [a;b]
d =
     1     2     3
     4     5     6
```

MATLAB generates an error code if you try to concatenate incompatibly sized vectors (or matrices):

```
x = [9; 10]
x =
     9
    10
% Stack d over x
y = [d; x]

Error using vertcat
Dimensions of matrices being concatenated are not consistent.
```

Vector Generation Functions

Several methods and functions allow you to create vectors and matrices more efficiently than entering each element manually, as shown in Table 1.3.

Table 1.3 Functions that Create Vectors

Method / Function	Description	Examples

colon operator (:)	Creates a vector from *x* to *y* in increments of *dt*. Default *dt* value = 1. Format: `z = x:dt:y`	`z = 1: 5`
		`z =`
		`1 2 3 4 5`
		`z = 1:3:10`
		`z =`
		`1 4 7 10`

linspace	Creates a vector with a set number of elements
	`z = linspace(x, y, numberElements)`
		`z = linspace(1,2,3)`
		`z =`
		`1.00 1.50 2.00`

ones	Creates an array with each element equal to 1; can also generate matrices	`% 1 x 2 vector`
		`z = ones(1,2)`
		`z =`
		`1`
		`% 2 x 2 matrix`
		`z = ones(2)`
		`z =`
		`1 1`
		`1 1`

zeros	Creates an array with each element equal to 0; can also generate matrices	`% 1 x 2 vector`
		`x = zeros(1,2)`
		`x =`
		`0 0`
		`% 2 x 2 matrix`
		`x = zeros(2)`
		`x =`
		`0 0`
		`0 0`

eye	Creates an identity matrix with ones on the diagonal and zeros elsewhere	`z = eye(2)`
		`z =`
		`1 0`
		`0 1`

Random-Number Generation Functions

Subsequent chapters review investment risk, and probabilities will factor into those discussions. MATLAB includes several functions that produce arrays of random numbers as shown in Table 1.4.

Table 1.4 Functions that Generate Random Numbers

Function	Description	Examples

rand(size)	Generates a uniformly distributed random number or sequence of numbers between 0 and 1	`% Single value`
		`x = rand()`
		`x =`
		`0.9649`
		
		`% 1 x 2 Vector`
		`x = rand(1,2)`
		`x =`
		`0.1576 0.9706`
		
		`% 2 x 2 Matrix`
		`x = rand(2)`
		`x =`
		`0.4854 0.1419`
		`0.8003 0.4218`

randi(maximumValue)
randi(maximumValue,n)
randi(maximumValue,r,c)
	Generates uniformly distributed random integers between 1 and *maximumValue*	`% Random scalar between 1 and 100`
		`x = randi(100)`
		`x =`
		`83`
		
		`% 1 x 2 random vector between 1 and 100`
		`x = randi(100,1,2)`
		`x =`
		`70 32`
		
		`% 2 x 2 random matrix between 1 and 100`
		`x = randi(100,2)`
		`x =`
		`96 44`
		`4 39`

(*continued*)

Table 1.4 (*continued*)

Function	Description	Examples

randn(size)	Generates a normally distributed random number or sequence of numbers	`% Normally distributed random scalar`
		`x = randn`
		`x =`
		`0.3129`
		
		`% Normally distributed random 1 x 2 vector`
		`x = randn(1,2)`
		`x =`
		`-0.8649 -0.0301`
		
		`% Normally distributed random 2 x 2 matrix`
		`x = randn(2)`
		`x =`
		`-0.1649 1.0933`
		`0.6277 1.1093`

It's worth noting that these results are pseudorandom numbers. The explanation of a pseudorandom versus a genuine random number is technical, but essentially, pseudorandom numbers are based on an algorithm whose sequence can be replicated if you repeat the initial settings. The MATLAB documentation has details on how the program produces values for the random number generator functions.

1.2.2 Math Calculations with Scalars, Vectors, and Matrices

MATLAB can perform extensive mathematical operations on data. The array's structure—scalar, vector, or matrix—will influence the operations' applications.

Scalars

Scalars' operations include addition, subtraction, multiplication, division, and exponentiation plus numerous more specialized function calls. Table 1.5 lists the notation for the most common operations.

Table 1.5 Common Scalar Operations

Operation	Symbol	Example
Addition	+	Numerical: 100 + 200 Variable: *x* + *y* Character: 'a' + 1 Date: datetime('1-Jan-2018') + 5
Subtraction (or negation)	–	Same as addition plus negation
Multiplication	*	100 * 200 *x* * *y* *x* * 5
Division (by)	/	10 / 5 = 2 *x* / *y*
Division (into)	\	10 \ 5 = 0.50 *x* \ *y*
Exponentiation	^	2^3 = 8 *x*^*y*
Square root	Function: sqrt()	sqrt(144) = 12

To avoid conflict, these operators follow a precedence from highest to lowest:

Parentheses: ( )
Exponentiation: ^
Negation: –
Multiplication and division: *, /, and \
Addition and subtraction: +, –

For example, (2 + 6) * 3 is not equal to 2 + 6 * 3:

```
(2 + 6)* 3
ans =
    24

2 + 6 * 3
ans =
    20
```

Vectors and Matrices

Many MATLAB math functions are vectorized; that is, you can apply them directly to vectors and matrices as well as scalars. The **sqrt** function is an example of this feature:

```
x = [81 144];
sqrt(x)
ans =
     9    12

y = randi(100,2)
y =
    82    13
    91    92

sqrt(y)
ans =
    9.0554    3.6056
    9.5394    9.5917
```

Addition/subtraction and multiplication/division with scalars also work with vectors and matrices:

```
x + 10
ans =
    91   154

y * 2
ans =
   164    26
   182   184
```

However, other math operations between vectors and matrices are more complicated. Assume that you have an investment portfolio with two positions (numShares) and their current market prices (mktVals). The data are stored in two column vectors:

```
numShares = [100;200]
numShares =
   100
   200

mktVals = [38;65]
mktVals =
    38
    65
```

The portfolio value is (100 × $38) + (200 × $65) or $16,800. An intuitive way to calculate that value is to multiply *numShares* times *mktVals* but that produces an error:

```
portVal = numShares * mktVals
Error using  *
Inner matrix dimensions must agree.
```

This example illustrates the need for *array operations*. When two vectors or matrices are the same size, you can operate on their corresponding elements. In MATLAB notation, this requires the use of "dot" notation:

```
Multiplication: .*
Division: ./
Exponentiation: .^

Examples:

portVal = numShares .* mktVals
portVal =
        3800
       13000

% The positions' values can be summed in the same calculation using
 the sum function:

portValTotal = sum(numShares .* mktVals)
portValTotal =
       16800

% Division with ./ reverses the multiplication
portVal ./ numShares
ans =
    38
    65

% Element by element matrix multiplication (versus matrix
 multiplication)

a = [1 2; 3 4]
a =
     1     2
     3     4

b = [5 6; 7 8]
b =
     5     6
     7     8
```

```
c = a .* b
c =
     5    12
    21    32

% Vector and matrix exponentiation
% Exponentiation with ^ works on square matrices but not on other
 sized vectors and matrices

x = [1 2;3 4];
x^2
ans =
     7    10
    15    22

x = [1 2 3];
x^2
Error using  ^
One argument must be a square matrix and the other must be a scalar.
Use POWER (.^) for elementwise power.

% Use .^ for element-by-element exponentiation
x = [1 2 3]:
x.^2
ans =
     1     4     9
```

Array multiplication and division proceed element by element so the arrays must be the same size. Matrix multiplication and division do not require same-sized arrays but they do require row-column compatibility. This is usually expressed as shown in Table 1.6.

In words, the number of rows in the second matrix must equal the number of columns in the first matrix. If that condition is satisfied, matrix multiplication uses the standard * (star) notation:

```
% 2 x 3
a = [1 2 3; 4 5 6]
a =
     1     2     3
     4     5     6
```

Table 1.6 Row-column Compatibility

Matrix A	Matrix B	Result
Size: $m \times n$	$n \times p$	$m \times p$
Example: 2×3	3×1	2×1

```
% 3 x 1
b = [7 8 9]'
b =
     7
     8
     9

% Solution is 2 x 1
a * b
ans =
    50
   122
```

Reshaping Arrays

You're likely to encounter array data stored differently than the row-column shape you need for a particular analysis. The MATLAB **reshape** function allows you to rearrange the data, provided the new shape has the same number of elements as the original shape. Suppose that you receive the data in a 10 × 1 vector but a more logical layout would be a 2 × 5 matrix. The **reshape** function syntax is:

reshape(original_array, desired number of rows, desired number of columns)

```
% A is the original 10 x 1 matrix
A = rand(10,1);

% Reshape to 5 x 2
B = reshape(A,5,2)
B =
    0.8147    0.0975
    0.9058    0.2785
    0.1270    0.5469
    0.9134    0.9575
    0.6324    0.9649
```

The **reshape** function has other features. You can reshape the original data into additional dimensions, assuming you have sufficient data points. For example, you could reshape the A matrix into three dimensions (row, column, depth) with an optional fourth argument (2, in this example):