Learning MATLAB:
Beginning Scientific Computing

Ian McCamey
Professor Tae Eun Kim

Introduction

This report contains an overview of my work in completing MATH 3607, Beginning Scientific Computing, a course specializing in the programming and numeric computing platform MATLAB at The Ohio State University during the Spring of 2023. The course was designed as an introduction to the mathematical theory of algorithms used to solve problems that typically arise in sciences, engineering, and finance. Specifically, several key concepts taught in this course are applicable to the main responsibilities of actuaries, data scientists, and other quantitative thinkers.

My purpose in presenting this document publicly is to showcase my experience in utilizing statistical software to solve complex mathematical problems via numerical computations. In addition, given the complexity of the mathematics involved, I hope to reduce each problem into a digestible form for a non-technical reader to understand, as this kind of conceptual simiplification is necessary when communicating with others in the professional world.

It is worthy to note that the programming language R is more commonly used in the actuarial profession for data-driven calculations and analysis. However, it and MATLAB share many commonalities. In particular, both

- are used for numerical computations,
- support functional programming,
- support array programming (vectorized operations),
- allow plotting of functions and data, and
- allow implementation of algorithms.

As to where they differ, an experienced programmer knows that their syntactical differences are minor compared to their conceptual similarities. However, R admittedly has a greater ability to handle and manipulate large datasets, while MATLAB can apply the theories of linear algebra far better than R. A near-comprehensive comparison between the two languages can be found here in a PDF from the University of Maine.

Let's begin to see how MATLAB can be used to gain insight about the nature of our world through mathematical models and computation.

Annuities.

A basic type of investment is an annuity: one makes monthly deposits of size P for n months at a fixed annual effective interest rate i, and at maturity collects the amount

$$F = Pa_{\overline{n}|i} = P\left(\frac{(1+i)^n - 1}{i}\right).$$

Say you want to create an annuity for a term of 300 months and final value of \$1,000,000. Using MATLAB, you can make a table of the interest rate you will need to get for each of the different contribution values $P = 500, 550, \ldots, 1000$. Implementing this into code, we can write:

```
n = 300;
F = 1000000;

for P = 500:50:1000
    if P == 500
        fprintf(' %10s %12s\n', 'Payment, P', 'Rate, r')
        fprintf(' %25s\n', repmat('-', 1, 25))
    end
    % function definition
    f = @(i) P*((1 + i)^n - 1)/i - S;
    % finding roots
    i = fzero(f, [eps 1]);
    fprintf(' %10d %12.6f\n', P, i)
end
```

which prints the following to the console:

```
 Payment, P      Rate, r
 -------------------------
        500     0.123512
        550     0.118146
        600     0.113200
        650     0.108607
        700     0.104317
        750     0.100287
        800     0.096485
        850     0.092884
        900     0.089461
        950     0.086198
       1000     0.083078
```

Random Walks.

In a computer, data is stored as a finite number of bits. Therefore, not every number can be stored *exactly*; there is a small error between the number and its computer-based (floating point) representation. It's reasonable to expect, then, that floating point errors accumulate randomly during a long computation.

On average we expect as many errors to be negative as positive, so they tend to partially cancel out. In fact, a classical result of probability is that, as the number of computations increases, the average of all of these random errors is very near 0.

Here is a plot of 50 simulations of this randomly compounding error in MATLAB.

```
clf
rng(6960)
n = 10000;
n_walks = 50;

for i = 1:n_walks
    % generate random errors
    r = randn(n,1);
    % accumulate errors
    y = cumsum(sign(r));
    plot(y(1:5:end)), hold on
end
title('50 Random Walks')
```