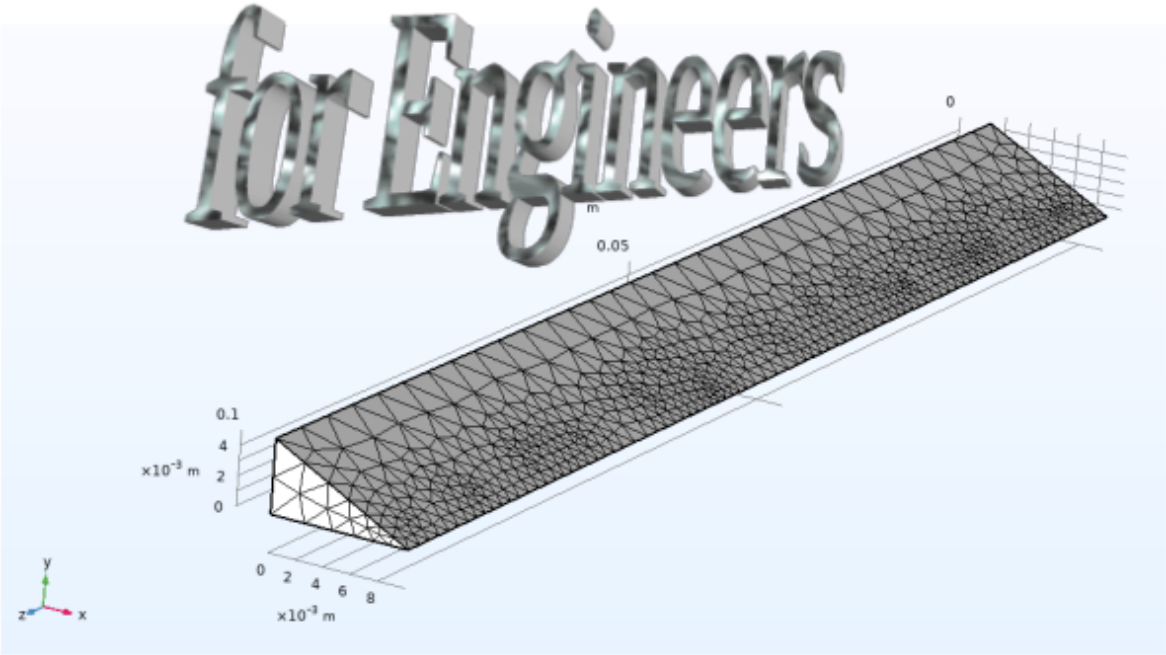
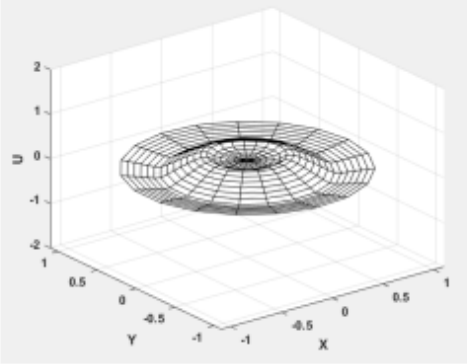
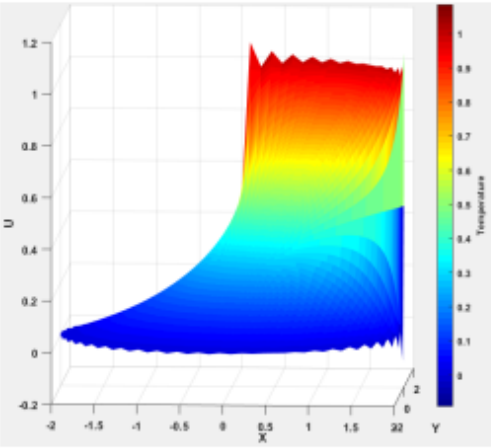
Mathematical Methods for Engineers







Stu Blair

Mighty Goat Press

THE ONLY WAY TO LEARN MATHEMATICS IS TO DO MATHEMATICS.

P.R. HALMOS

THE PURPOSE OF COMPUTING IS INSIGHT, NOT PICTURES

L.N. TREFETHEN

NEVER DO A CALCULATION UNTIL YOU ALREADY KNOW THE ANSWER.

J.A. WHEELER

UNITED STATES NAVAL ACADEMY

MATHEMATICAL METHODS FOR ENGINEERS

MIGHTY GOAT PRESS

PUBLISHED BY MIGHTY GOAT PRESS

First printing, April 2023

Contents

List of Figures

List of Tables

Preface

The purpose of this text is to provide a concise reference for engineering students who would like to strengthen their conceptual understanding and practical proficiency in analytical and numerical methods in engineering. The material is based on a sequence of two courses taught at the United States Naval Academy.

Analytical Methods

The first course focused on analytical methods for linear ordinary and partial differential equations. All students came into the course having taken a three-semester sequence of calculus along with a course in ordinary differential equations. The analytical methods portion quickly reviews methods for constant coefficient linear equations and proceeds to methods for non-constant coefficients including Cauchy-Euler equations, power series methods, and method of Frobenieus. After a review of Fourier Series methods and an introduction to Fourier-Legendre and Fourier-Bessel expansions we thoroughly explore solutions to second-order, linear, partial differential equations. Since many students are also studying nuclear engineering, there is a heavy focus on addressing boundary value problems in cylindrical and spherical coordinate systems that are applicable to other topics of interest such as reactor physics. There is also heavy emphasis on heat transfer applications that students will see later on in their undergraduate curriculum.

The materials presented are based heavily on Professor Dennis Zill's excellent book.[1] We lightly select from chapters 1-3 for review; chapter 5 for series solution methods; and chapters 12-14 for Fourier Series and solutions to linear boundary value problems. Material from that text is used throughout this book.

What distinguishes this course from Prof Zill's work is the incorporation of computational tools in the solution process. These "semi-analytical methods" are presented here in MATLAB[2] owing to the students preparation with that tool. Other open-source tools like Octave[3] and Python,[4] of course, could be used.

[1] Dennis G Zill. *Advanced engineering mathematics*. Jones & Bartlett Learning, 2020

[2] Inc. The Math Works. Matlab, v2022a, 2022. URL https://www.mathworks.com/

[3] John W. Eaton, David Bateman, Søren Hauberg, and Rik Wehbring. *GNU Octave version 5.2.0 manual: a high-level interactive language for numerical computations*, 2020. URL https://www.gnu.org/software/octave/doc/v5.2.0/

[4] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009. ISBN 1441412697

Part I

Introduction and Review

Lecture 1 - Introduction, Definitions and Terminology

Objectives

The objectives of this lecture are:

- Provide an overview of course content
- Define basic terms related to differential equations
- Provide examples of classification schemes for differential equations

Course Introduction

The basic topics that we will cover include:

Part II

Power Series Methods

Part III

Back Matter

Bibliography

John W. Eaton, David Bateman, Søren Hauberg, and Rik Wehbring. *GNU Octave version 5.2.0 manual: a high-level interactive language for numerical computations*, 2020. URL https://www.gnu.org/software/octave/doc/v5.2.0/.

Inc. The Math Works. Matlab, v2022a, 2022. URL https://www.mathworks.com/.

Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009. ISBN 1441412697.

Dennis G Zill. *Advanced engineering mathematics*. Jones & Bartlett Learning, 2020.

Appendices

Matlab Style Rules

1. **rule:** All scripts will start with the commands: **clear**, **clc**, and **close 'all'**

   rationale: No script should depend upon any data visible in the MATLAB workspace when the script starts. By omitting these commands, residual data within the workspace may hide errors.

2. **rule:** Your code must be documented with enough details such that a reader unfamiliar with your work will know what you are doing.

   rationale: Code documentation is a habit. For more significant projects readers may need help in deciding what the author of the code intended. For your own code, the most likely reader is you—a few months into the future.

3. **rule:** Function and variable names must be meaningful and reasonable in length.

   rationale: Failing to do either make code harder to read and maintain.

4. **rule:** All outputs from the code **must** be meaningful; numbers should be formatted, part of a sentence, and include units. Graphs should be readable and axis labels should make sense and include units.

   rationale: Code output is a form of communication. It is important that this communication be clear and unambiguous.

5. **rule:** Do not leave warnings from the Code Analyzer unaddressed.

   rationale: Sometimes Code Analyzer warnings can be safely ignored. Most of the time the warning points to a stylistic error that would be unacceptable in software that you use. Occasionally these warnings are indicative of a hidden error.

6. **rule:** Use the "smart indentation tool" to format the indentation of your code.

rationale: This tool improves code readability. It will also occasionally point out errors that you did not see before.

7. **rule:** Pre-allocate arrays; if possible initialize with **NaN** values.

   rationale: Pre-allocation improves performance and helps readability. Initialization with **NaN** helps avoid a range of potential logical errors.

8. **rule:** Avoid "magic numbers" — i.e. hard-coded constants.

   rationale: Constants included in your code tend to hide your program logic. Also, "magic numbers" make code maintenance more difficult and error prone.

9. **rule:** Only write one statement per line.

   rationale: Multi-statement-lines hurt code readability in almost all cases.

10. **rule:** Do not write excessively long lines of code; use the line continuation "..." and indentation to spread long expressions over several lines.

    rationale: Following this rule improves code readability.