WIP: Integrating Python Programming into Analytical Methods for Electrical and Computer Engineering Students

Dr. Robert J Kerestes, University of Pittsburgh

Robert Kerestes, PhD, is an assistant professor of electrical and computer engineering at the University of Pittsburgh's Swanson School of Engineering. Robert was born in Pittsburgh, Pennsylvania. He got his B.S. (2010), his M.S (2012). and his PhD (2014)

Collin Joseph Griffin, University of Pittsburgh

Collin Griffin is a third-year PhD student in the Electrical and Computer Engineering department at the University of Pittsburgh. Collin earned joint bachelor's degrees in computer science and economics in 2022 and a master's degree in electrical and computer engineering in 2024, each from the University of Pittsburgh. Collin's research interests include power systems simulation and modeling, the application of machine learning and artificial intelligence to power systems, and engineering education.

WIP: Teaching Electrical and Computer Engineering Mathematics through Python Programming Exercises and Visualizations

Abstract

This work-in-progress paper presents a set of Python programming exercises developed to enhance student learning and preparedness in an introductory course on analytical methods for electrical and computer engineering (ECE) students. The course, taught by ECE faculty to sophomore-level students, replaces traditional mathematics department courses in ordinary differential equations and linear algebra. Its primary objective is to emphasize the mathematical concepts most relevant to ECE applications and to strengthen student understanding through application-based learning. Course topics include basic linear algebra, first- and second-order differential equations, basic and composite signals, complex numbers, Laplace transforms, and frequency and phase analysis. To support these topics, four Python programming modules have been developed: solving linear systems, applying numerical methods to first- and second-order ODEs with initial conditions, plotting composite signals and their components, and performing frequency and phase analysis of system transfer functions. This paper describes these exercises in detail as part of an ongoing curricular development effort. Future work includes collecting both direct and indirect assessment data to evaluate the effectiveness of the programming exercises in improving student learning outcomes.

1 Introduction

The field of electrical and computer engineering (ECE) requires a substantial underpinning of mathematics. In university-level engineering programs, mathematics courses are typically administered early in the curriculum to ensure students' mathematical maturity before progressing to classes specific applications of these principles. Many students struggle in these early courses, leading to reduced motivation, low performance in later courses, course failure, and in some cases, students switching into different programs entirely. Since the content in these mathematics courses are often not immediately related to students area of interest or perception of engineering work, facilitating intuitive understanding of this content is critical to have students integrate the coursework into their skill sets for future application, rather than view the material as simply a difficult course required for advancing in their degree.

One proposed technique for achieving this deeper level of understanding of engineering mathematics topics is to have students utilize programming languages to develop systematic problem solving techniques and visualizations of abstract concepts. Introducing such exercises early in students curriculum not only helps achieve the critical level of understanding of these core concepts, but also gives students early exposure to programming techniques, which are broadly utilized in many domains of ECE. In this work-in-progress paper, we present a series of programming exercises that are to be administered as a part of a sophomore-level engineering mathematics course in the ECE department. These exercises are crafted to align directly with the in-class curriculum, with focus given to areas where students often have trouble developing intuitive understanding of the topics. The exercises focus on showing how concepts can be represented in new by translating in-class mathematics concepts to respective computer program representations, with systematic methods for solving and visualizing problems. With this additional form of instruction and practice, students should become more confident with course content, and be positioned to succeed in this course and those following it.

2 History of Course

In 2018, the Electrical and Computer Engineering (ECE) program at the University of Pittsburgh undertook a comprehensive overhaul of its undergraduate electrical and computer engineering curricula. A key component of this effort involved rethinking how upper-level mathematical content, including differential equations, linear algebra, probability, and signals and systems, was delivered to engineering students. Rather than relying on traditional service courses taught outside the department, the ECE faculty made a deliberate decision to assume full responsibility for teaching this material within the department. This represented a substantial departure from prior instructional models. Existing literature suggests that mathematics instruction delivered by engineering faculty, when contextualized within engineering applications, can be more effective for engineering students than instruction delivered in isolation by mathematics departments [1].

As part of this curricular redesign, the department developed a two-course sequence. The first course, ECE 0401 – Analytical Methods for Electrical and Computer Engineering, focuses on differential equations, linear algebra, Laplace transforms, frequency analysis, and multivariable calculus. The second course, ECE 0402 – Signals, Systems, and Probability, builds on this foundation and covers Laplace transforms, Fourier series, the Fourier transform, and probability. Students take ECE 0401 and ECE 0402 in their third and fourth terms respectively. This follows their first year curriculum, which includes calculus 1 and 2. Thus, the mathematics concepts presented in ECE 0401 and 0402 are not assumed to have been introduced formally to students prior to their taking the courses.

Both courses were intentionally designed to emphasize not only mathematical theory but also its direct relevance to electrical and computer engineering systems. Each mathematical topic is explicitly connected to a concrete engineering application, minimizing abstraction for abstraction's sake and addressing the common student concern of "when will I ever use this?" This application-driven approach ensures that mathematical concepts are consistently framed as tools for modeling, analyzing, and understanding real engineering systems. As students progress through the ECE curriculum, the knowledge gained through this sequence remains critical, as it continues to be applied

in various contexts in both upper level required courses and electives. For instance, the required courses ECE 1259 Electromagnetics, 1560 Digital Signal Processing, and 1673 Linear Control systems all directly apply the mathematics from 0401 and 0402, making an intuitive understanding of the mathematics in this sequence essential.

In addition, these courses were structured to align closely with other sophomore-level ECE courses, particularly those focused on linear circuit analysis and electronic circuit analysis. By synchronizing mathematical instruction with concurrent engineering coursework, the sequence strengthens students' ability to apply mathematical concepts across multiple domains within the curriculum. To date, however, the emphasis of this two-course sequence has been primarily on theory and application, with relatively limited integration of computational and programming-based approaches. This work-in-progress paper focuses on addressing that gap by incorporating a meaningful programming component into the sequence—specifically in ECE 0401—and positioning computational analysis as a future curricular strength of the ECE program.

3 Background

Programming exercises frequently appear as a supplemental tool in engineering education. Engineering textbooks, including the one used for this course [2] often include a programming component, to help students perform numeric and symbolic calculations, as well as visualize function outputs. Existing research literature has also explored the application of teaching mathematics and engineering concepts using programming. Choice of programming language is one key differentiator in existing literature. In many, like [3–5] MATLAB is utilized as one of the programming languages for each work's respective programming exercises or projects. The work presented in [3] specifically covers an electromagnetics course, and the researchers apply computer algebra systems available in Maple, MATLAB, Mathematica, and MathCAD to help students develop problem representations, leaving them with more time to spend on physical interpretations of problems rather than manual calculations. In others, like [6], students are permitted select the language and platform used for projects with a larger, semester-long scope, tasking students with developing standalone Windows applications.

This course covers also ordinary differential equations, which are well suited as an application of numerical methods and corresponding programming exercises. Numerical methods textbooks like [7] and [8] use MATLAB and the Julia programming language respectively to instruct students on the development of numerical methods solvers, both for the ODEs concerning this class, but also linear systems–which this course covers, but not in the context of numerical methods. In [9], the author develops programming curriculum for an undergraduate chemistry course, where students are also expected to understand the application of numerical methods for systems of differential equations. The work's curriculum was developed to utilize Python and many of its critical scientific packages like numpy, matplotlib, scipy, and sympy, specifically to drive students understanding of core concepts rather than the performance of complicated hand calculations.

Visualization is a critical need for students when approaching complex mathematics topics. When a mathematics concept a necessity for understanding a student's area of interest, and a student struggles with the mathematical component, they may experience frustration and reduced moti-

vation, even in a field that otherwise excites them. Visualization can often serve to assuage the difficulty students have when first being introduced to a new mathematics concept, and help them develop the intuition behind what may otherwise remain an abstract mechanism to simply be memorized for later examination. In [3], the author describes electromagnetics a particularly difficult class for students, making the course "unpopular". The author cites the difficulty with the course as being related to the abstract concepts that are difficult for students to picture, as compared with more concrete courses like those in mechanics, control, or circuits. To address this, the author develops example problems and solutions in Maple and MATLAB for students to visualize two and three dimensional models of electromagentics concepts.

Online resources also offer students convenient supplemental resources to help visualize difficult concepts. For example, the online linear algebra textbook in [10], offers many interactive demos including plotting vectors, linear transformations, eigenvectors, and orthogonal projection. Another notoriously difficult concept for engineering students to grasp is that of convolution. Two online examples of convolution plotting in [11] and [12] allow for students to interactively change the input and impulse response functions, as well as time offsets to see how the convolution is affected. The number of combinations of the two functions available in these interactive demos is very large, but these demos, as well as many other interactive visualizations may fail to fully develop the intuition necessary for students to apply mathematical concepts to engineering problems. We believe that this gap can be remediated by having students program the visualizations themselves, to both understand the problem and how it may be addressed in a systematic manner, as well as generate a visual representation of the problem and its solution.

4 Methodology

For this work, four modules were developed—Linear Algebra, Differential Equations, Signal Theory, and Continuous Time Systems—aligning with the core areas of the course. An additional, ungraded "Module 0" is also available for students to provide students a primer in Python programming, as it is not explicitly covered in the ECE curriculum prior to this course. The four primary modules have between four and nine exercises. Each exercise is represented as an isolated Python function, with exercises also making use of helper functions which the students may be tasked to write. The exercise functions are provided mostly empty, with some starter code provided provided where necessary for clarity.

While programming exercises in related engineering textbooks and existing literature are frequently utilized for explaining mathematical concepts, this work aims to develop a comprehensive set of exercises that connects the four main topics of the course in a manner that builds naturally and emphasizes developing an intuitive understanding of the mechanisms of the mathematical concepts. Each module requires students to develop visualizations systematic procedures for doing calculations or approximating analytical methods presented in class, requiring a deeper analysis of each topic than rote memorization of a formula or technique. The exercises have been specifically developed to augment the course content, rather as just related additional examples. Additionally, due to the related nature of the modules and exercises, the exercises permit and encourage re-use of earlier developed functions, which becomes especially relevant in the final two modules, which rely on common representations of signal functions that students will repeatedly modify and

analyze.

The instructions for each exercise are provided within the context of a custom website created using Markedly Structured Text (MyST) [13], which is a popular web application framework commonly used for creating online textbooks and scientific publications. This medium offers a rich text environment that provides better organization, search functionality, and code highlighting than traditional handouts. It also allows for videos demonstrating setup of the programming environment to be directly embedded into the content, as well as collapsible content for exercise hints or other tangential information. An example of an exercise's webpage is shown in 1.

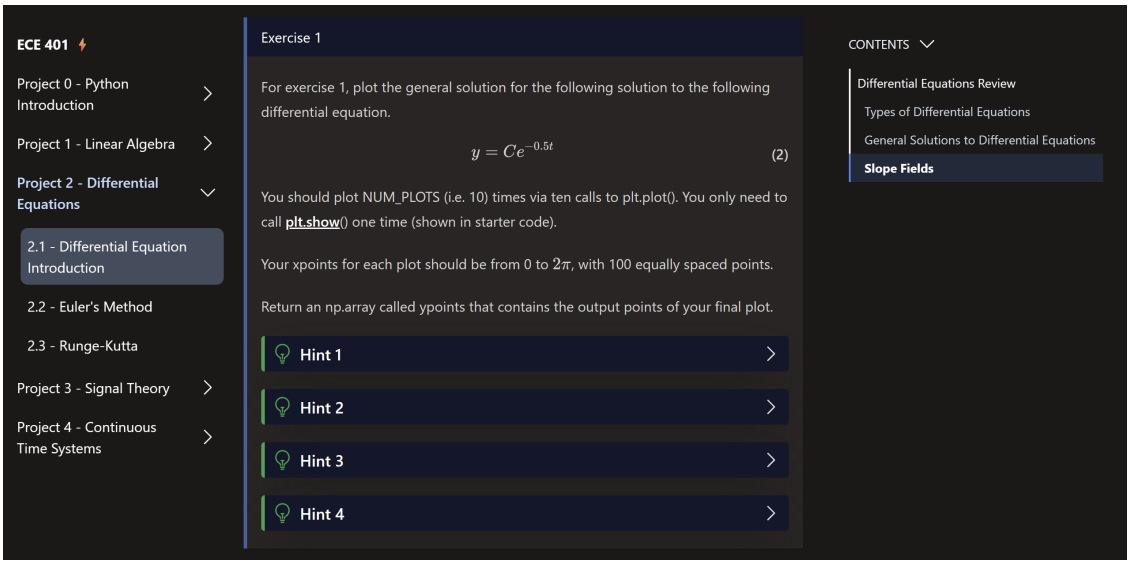


Figure 1: Example Exercise on Course Webpage

Each module is divided into sub-modules, which are further divided into sections that correspond with topics covered in the course. These sub-modules provide instruction through review of course content as well as the direct instruction specific to the implementation of each exercise. The instructional context is derived from the same content covered in class, but differs in its emphasis on how the concepts covered in class can be translated to programming methods.

Exercise instructions consist primarily of a brief description of the task, including setup of inputs, helper functions, and necessary equations. The exercise description also describes the return values, which are used in the automated assessment of the student's code.

The programming exercises described have been developed, but have not yet been administered to students. Thus, the exercises may be subject to change slightly prior to the first term they are administered (Fall 2026).

4.1 Module 1 - Linear Algebra

The first module covers linear algebra and is split into three sub-modules: 1.1 - Vectors and Matrices, 1.2 - Matrix Determinants and Inverses, and 1.3 - Gaussian Elimination and Solving Linear Systems. The learning goals for this module are three-fold:

1. Gain confidence with computer programming representations of vectors and matrices.
2. Create visualizations of vectors and the effect of linear transformations on vectors.
3. Understand how linear systems can be solved systematically through development of a Gaussian Elimination algorithm.

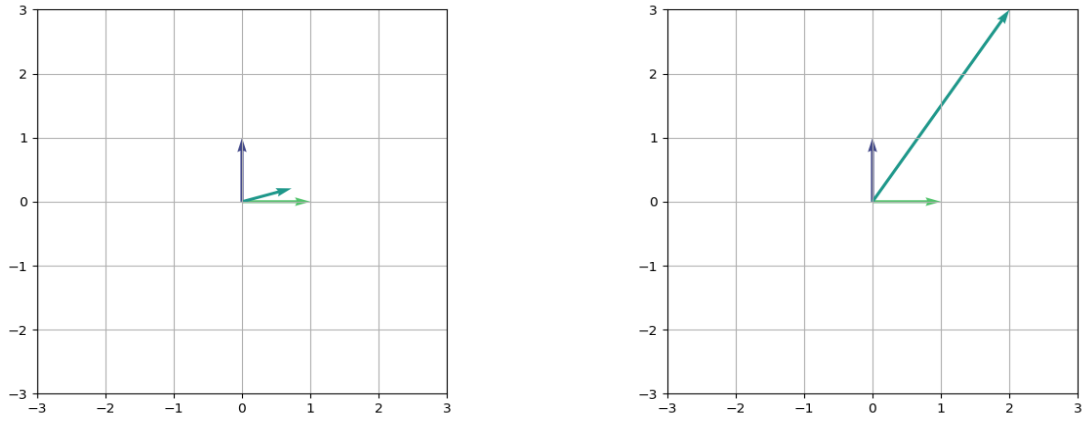
The primary learning objective of this module is to introduce students to representing matrices in a programming environment,

4.1.1 Vectors and Matrices

The goal of this sub-module is to introduce students to representing vectors and matrices in Python. This module also introduces students to the NumPy [14]—a commonly used Python package for creating and operating on matrix objects in Python. Exercise 1 in the module guides the student to compare vanilla Python 2-dimensional lists and NumPy array objects. NumPy arrays are much closer to the mathematical understanding of matrices developed in the course than the vanilla Python 2D lists, and the exercise demonstrates this by comparing the scalar multiplication operation between the two models.

Exercises 2 through 4 are brief exercises each demonstrating basic matrix operations. These operations include matrix addition, subtraction, and multiplication. While simple, these exercises show that how Python can be used to quickly perform matrix calculations that can be time-consuming by hand, and allow the students to practice the most basic usages of matrices in Python.

Exercise 5 establishes an understanding of basis vectors and linear combinations. Starter code is provided for students for this exercise that provides more complex plotting logic, as this is unrelated to the learning objective of the exercise. The students are instructed to run the starter code, which generates a plot of the standard 2D basis vectors, along with a randomly generated vector in between the two bases.



(a) Starter Code Output

(b) Solution Output

Figure 2: Module 1 Exercise 5 Output

This sub-module concludes with Exercise 6, which explores plotting a vector in $\mathbb{R}^3$, then performing a linear transformation that reduces the dimensionality of the vector, and plotting this $\mathbb{R}^2$

vector.

4.1.2 Matrix Determinants

Sub-Module 1.2 focuses on the matrix determinant and inverse operations, and involves two exercises.

Exercise 7 focuses on the determinant, specifically having students create a function that applies co-factor expansion to recursively solve matrices of arbitrary sizes. The primary challenge with this exercise is for the students to determine how to generate the minor matrices through removing rows and columns of the original matrix. The equation provided to students as the basis of the exercise is shown in Equation 1.

$$det(A) = \sum_{j=1}^{n} -1^{i+j} \times a_{ij} \times det(M_{ij}) \tag{1}$$

Exercise 8 explores the matrix inverse operation. Students are instructed to use NumPy functions to determine the inverse, and then apply the inverse to the right-hand side vector to find the solution vector.

4.1.3 Gaussian Elimination and Solving Linear Systems

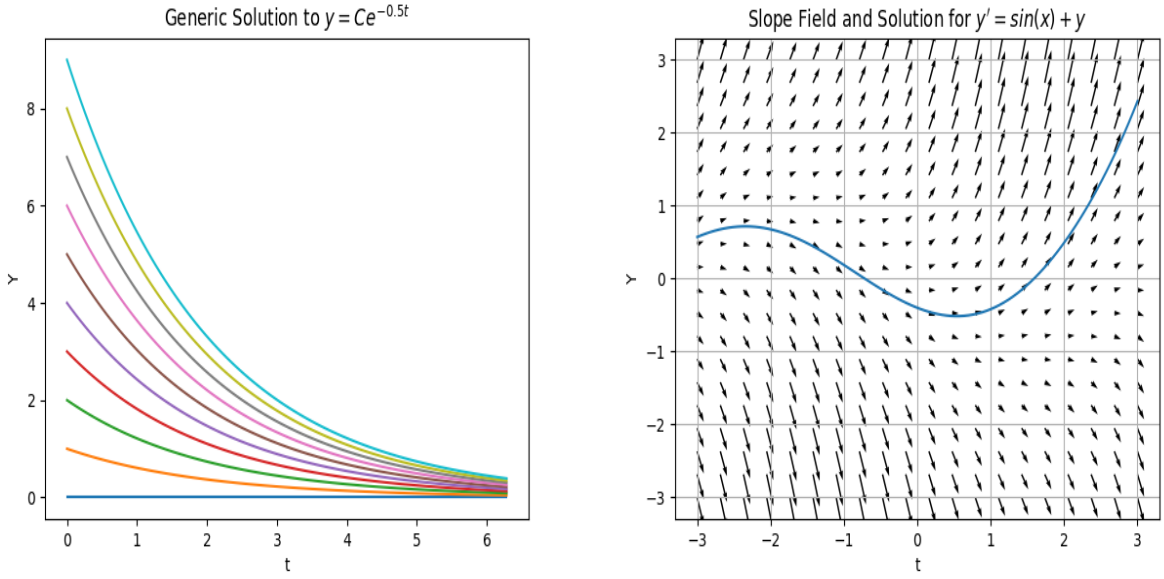
For the final sub-module of Module 1, only one, larger exercise (Exercise 9) is presented to students to have them implement Gaussian Elimination. Gaussian Elimination is presented in class as a method for solving systems of equations, or, when they are not solvable, revealing properties of linear dependence. For the exercise, the students are presented detailed instructions to help assist them in translating the method presented in class into an algorithmic representation in code. When the students are confident in their solution, they are then asked to check it against NumPy's matrix solving method—`np.linalg.solve(A, b)`.

4.2 Module 2 - Differential Equations

Module 2 covers differential equations. The in-class curriculum for this module focuses on developing analytical methods used for solving ordinary differential equations (ODEs). This module complements the in-class instruction by instead focusing on visualizing slope fields and general solution curves to differential equations, as well as developing simple numerical methods for solving ODEs. This module is divided into three sub-modules: 2.1 - Differential Equation Introduction, 2.2 - Euler's Method, and 2.3 - Runge-Kutta Methods with a total of 4 exercises. The learning goals for this section are:

1. Create visualizations of general solutions to differential equations and slope fields.
2. Develop numerical solvers to demonstrate how differential equations may solved in an alternative manner to analytical methods presented in class.
3. Compare multiple numerical solvers for differential equations and contrast level of accuracy and efficiency.

4.2.1 Differential Equation Introduction



(a) 2.1 Exercise 1 Output

(b) 2.1 Exercise 2 Output

Figure 3: Sub-Module 2.1 - Exercises 1 and 2

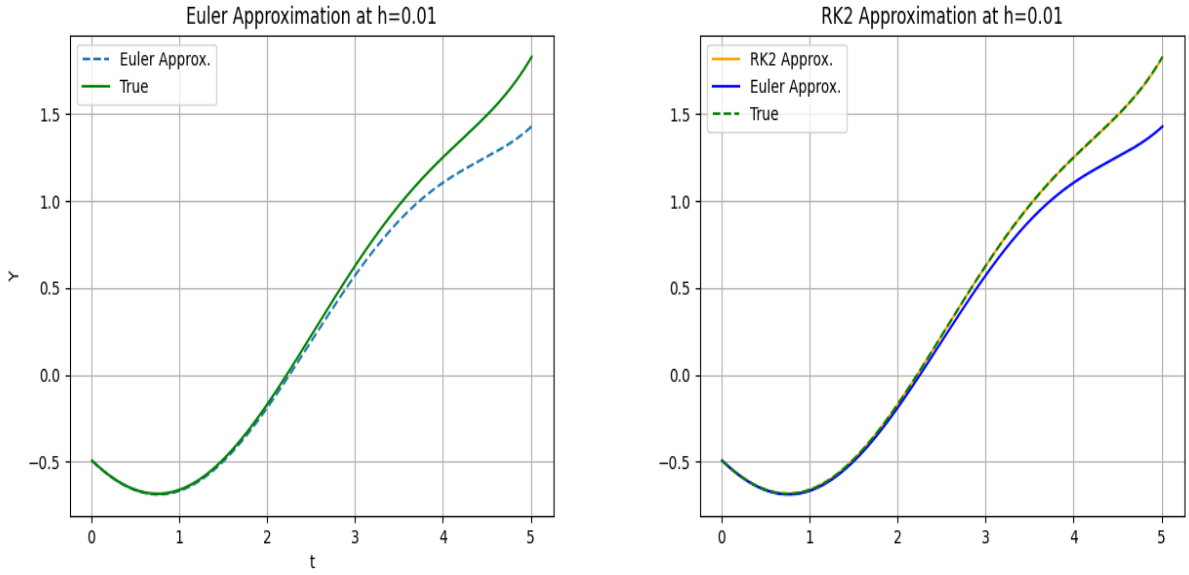
Module 2.1 focuses on the visualizations of differential equations, specifically slope fields and plotting both specific and general solutions to differential equations. The module starts with introducing students to the Matplotlib Python package [15], which is commonly used for plotting functions. This package is used extensively in the remaining exercises for helping students generate visualizations of exercise content.

Exercise 1 of Module 2 has students plot the general solution to $y = Ce^{-0.5t}$ by iterative calls to `plt.plot(...)`, adding all plots to the same image. The students use the index of the iteration to set multiple values for the C constant. The final plot of the exercise should appear as Figure 3a.

Exercise 2 aims to explore how solution curves are developed by having students plot slope fields. This concept is then built upon in the following two numerical solver sub-modules. The students are given a differential equation ($y' = sin(x) + y$) and asked to generate the slope field for it, in the x and y ranges of [-3, 3]. The student is also asked to find the general solution of the differential equation, then plot it at $C = 0.1$. The expected final output of the exercise is shown in Figure 3b.

4.2.2 Euler's Method

Sub-Module 2.2 builds on the slope fields exercise by demonstrating that a numerical method for solving ODEs can be developed by simply "following" the slopes as given by the output of the differential equation. Thus, the following exercise helps students build the intuition behind Euler's method, and make its development in Python easier.



(a) 2.2 Exercise 3 Output

(b) 2.3 Exercise 4 Output

Figure 4: Outputs of Sub-Module 2.2, 2.3

This module only contains a single exercise (Exercise 3) to implement Euler's method, but additionally requires that the student experiment with different step sizes in their implementation to assess the accuracy of the method and develop a basic understanding of discretization error in numerical methods. The student is also asked to plot their Euler Method approximation against the true solution curve. The differential equation and solution curve are the same as Exercise 2. The output curve of this exercise, when assessed on a step size of 0.01, is plotted in Figure 4a.

4.2.3 Runge-Kutta Methods

In the final sub-module of the differential equations module, students implement another numerical ODE method—Runge-Kutta—and compare it against the performance of Euler's Method. Runge-Kutta is a family of methods, the simplest of which is the second order method known as RK2. For Exercise 4, students are provided with the set of equations comprising the RK2 method, as shown in 2. The students then plot this against the output of the Euler's method for the same step size, as well as the true solution. The output for this exercise is shown in Figure 4b

$$\begin{aligned} k1 &= h * f(x_n, y_n) \\ k2 &= h * f(x_n + h, y_n + k1) \\ y_{n+1} &= y_n + \frac{1}{2}k1 + \frac{1}{2}k2 \end{aligned} \tag{2}$$

4.3 Module 3 - Signal Theory

The third module focuses on basic signal theory. The in-class coursework related to this module focuses on basic signal waveforms and signal arithmetic, and the module largely follows this with an additional emphasis on the visualization of these waveforms and modifications to them. This module is broken into two sub-modules: 3.1 - Basic Signals and 3.2 - Signal Operations. The learning goals for this section are:

1. Represent signal functions as equivalent programming language functions.
2. Understand how programming language functions may be operated on mathematically, in the same manner as their in-class mathematical representation.
3. Visualize transformations to signal functions.

4.3.1 Basic Signals

The 3.1 Sub-Module focuses on a set of basic signal waveforms. The three specific waveforms included in this section are the step, pulse, and ramp functions.

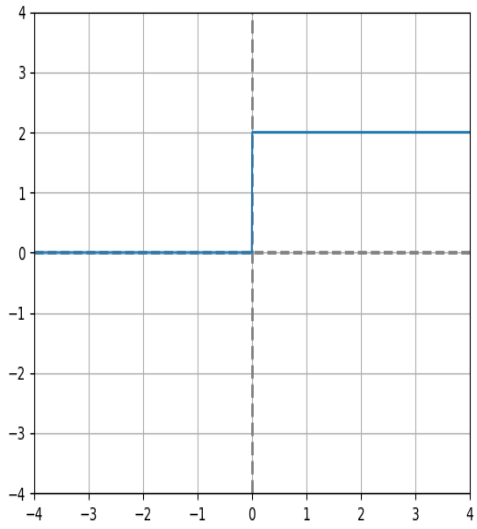
The module starts with an explanation of the relation between mathematical functions, like the basic waveforms described, and Python functions. This explanation includes instruction on using the `np.vectorize(func)` function to allow functions to be mapped over a list of input, rather than explicitly looping over the list and evaluating each input value. This method allows for usage of the functions more akin to their mathematical understanding, using the variable directly rather than constructing a loop to iterate the variable.

The three exercises in this section have the students write functions for and plot the step, pulse, and ramp waveforms respectively. The instructional content for the pulse waveform also includes an explanation of the Delta-Dirac function, which is utilized later in Module 4 for the sampling property and signal convolution.

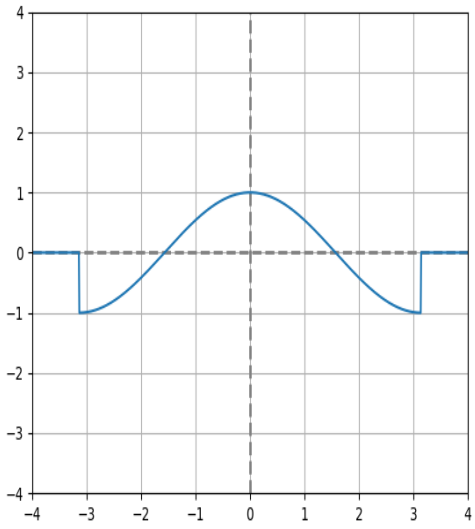
4.3.2 Signal Operations

Sub-Module 3.2 takes the definitions of the waveform functions from the previous sub-module, and applies signal arithmetic and time operations to them. The primary motivation of this sub-module is to help students visualize how these operations shift the outputs of each waveform function and how they are related to their equivalent scalar mathematics context.

Exercise 4 instructs students to add identical signals together, demonstrating that the result is effectively the same as performing scalar multiplication of the signal, as shown in Figure 5a. Exercise 5 instructs the students to multiply the pulse waveform function they developed in the previous sub-module by the `np.cos()` function evaluated on the same time interval. This has the effect of "windowing" the cosine function, and is shown in Figure 5b.



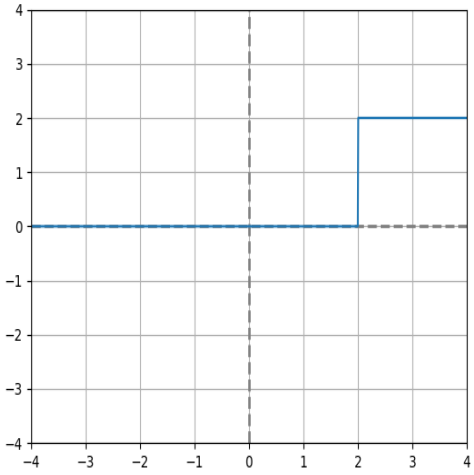
(a) 3.2 Exercise 4 Output



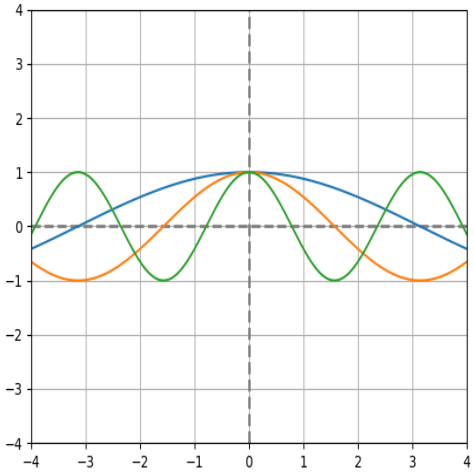
(b) 3.2 Exercise 5 Output

Figure 5: Sub-Module 3.2 Arithmetic Operations Outputs

Exercises 6 and 7 demonstrate the time shifting and scaling operations respectively. Exercise 6 instructs students to subtract and add to the input to the step waveform previously developed, and plot the respective shifts right and left shifts of the function output. The output of the rightward shift (from subtracting from the input time) is shown in Figure 6a Exercise 7 begins by explaining the relation between the scaling factor a applied to the waveform input and the resulting compression or expansion. The expected output of this exercise is shown in Figure 6b.



(a) 3.2 Exercise 6 Output



(b) 3.2 Exercise 7 Output

Figure 6: Sub-Module 3.2 Timescale Operations Outputs

4.4 Module 4 - Continuous Time Systems

The final module covers Continuous Time Systems. The in-class coursework topics aligning with this module include system definitions and properties (linearity, causality, memory, time invariance, and invertibility), convolution, the LaPlace Transform and frequency and phase analysis of transfer functions. This module is broken into three Sub-Modules: 4.1 - System Properties, 4.2 - Sampling Property and Convolution, and 4.3 - Analysis of Transfer Functions. The learning goals for this module are:

1. Analyze properties of LTI systems by programmatically applying additive and scaling operations to systems.
2. Develop method for calculating and plotting convolution of signals.
3. Plot magnitude and phase response of transfer functions, and compare each response to modifications to function input parameters.

4.4.1 System Properties

This sub-module contains instructional content providing a review of the system properties covered in class, as well as a closer analysis of linear time-invariant (LTI) systems. The only exercise for this sub-module has students first plot a sine wave using NumPy and Matplotlib, and then use a provided "biased averager" function derived from an assigned homework problem example from [2] to plot the constant response of the input sine wave signal. The student then plots the biased average, the biased average with the sine input signal multiplied by a constant, and the original biased output multiplied by the same constant. The student should then observe the result in Figure 7, indicating that the "biased averager" system does not satisfy the scaling property of superposition, and is thus not linear.

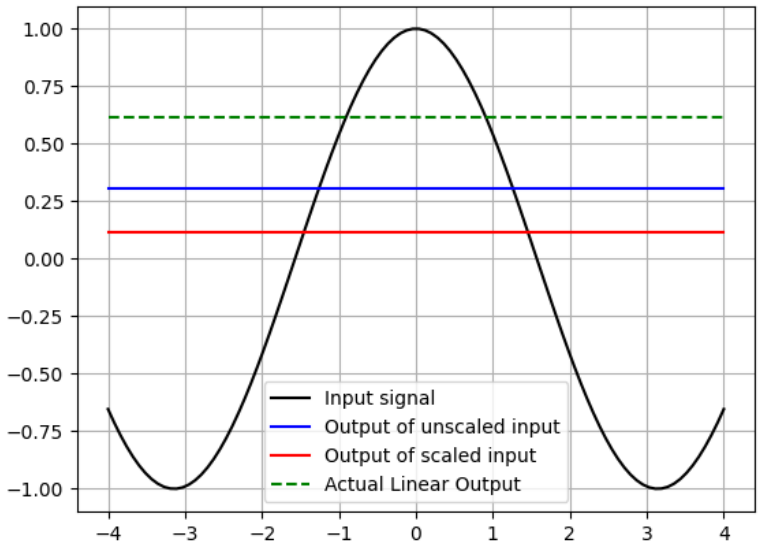
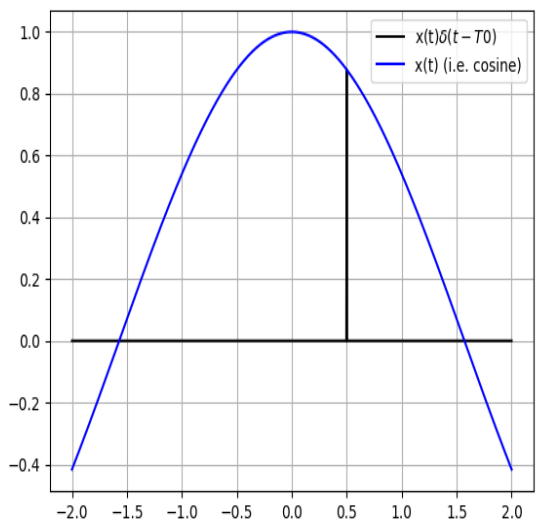


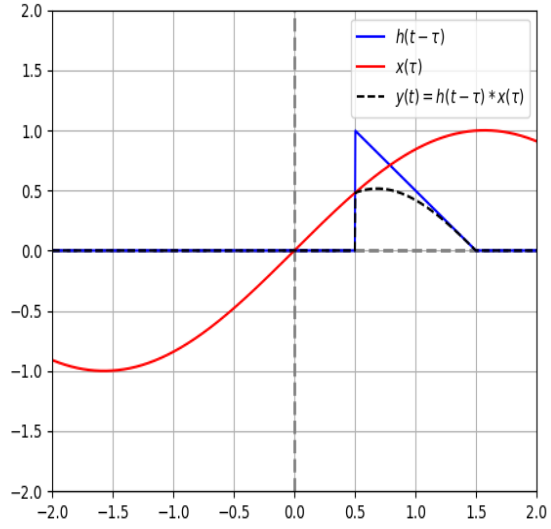
Figure 7: Output of Sub-Module 4.1 - Exercise 1

4.4.2 Sampling Property and Convolution

Sub-Module 4.2 begins with Exercise 2 demonstrating the sampling property of the impulse/Delta-Dirac function. The students are provided with a simple implementation of the Delta-Dirac function, and asked to demonstrate the sampling property by calculating the integral over a specified input domain. The exercise instruction explains that a numerical approach to integration should be taken for the exercise that approximates the integral by taking the sum of many values of t that must include t=0. The students plot both the $x(t)$ signal (cosine in this case) and the evaluation of the product at each point. The exercise reveals that this product is 0 everywhere except at t = 0. The exercise then asks the students to test different offsets to the Delta-Dirac function, which they should observe returns different discrete integral sums and shifts the output plot accordingly. An example of the output plot at offset = 0.5 is shown in Figure 8a.



(a) Sub-Module 4.2 - Exercise 2 ($T0 = 0.5$)



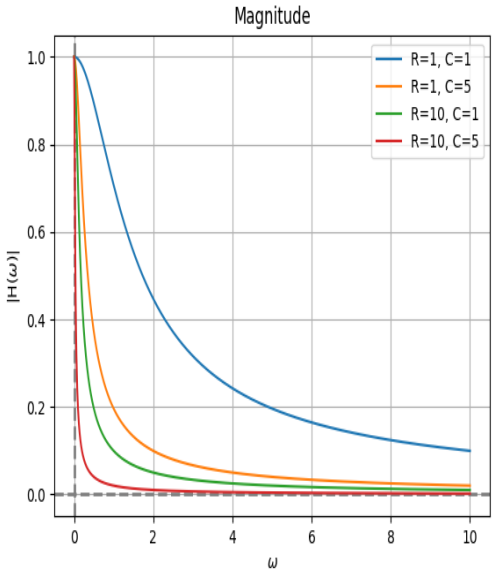
(b) Sub-Module 4.2 - Exercise 3 ($t = 0.5$)

Figure 8: Sub-Module 4.2 Exercises 2 and 3 (Sampling and Convolution)

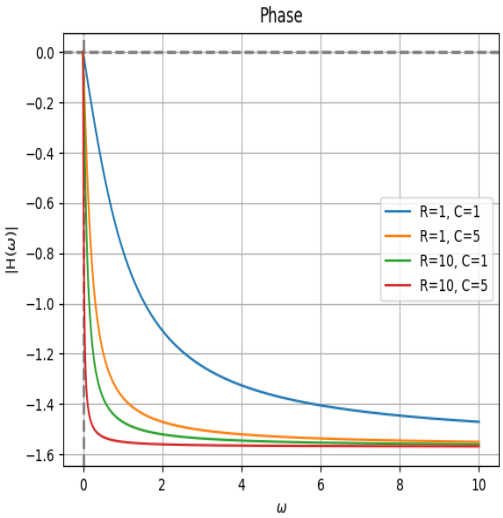
Exercise 3 has students implement the convolution of two signals, using a similar discrete process of evaluating the integral as in Exercise 2. The theory of convolution is developed in class with several worked examples, and this exercise helps complement that instruction by plotting the input, shifted impulse response, and output signals. Students are to use the waveform functions they developed in Module 3 as their impulse response and input waveforms, and are tasked with experimenting with different waveforms substituted for each of these, as well as testing different time shifts. The output of the exercise with $h(t) = r(t)$ (i.e. the ramp function) and $x(t) = np.sin()$ with a time shift of 1.5 is shown in Figure 8b.

4.4.3 Magnitude and Phase Analysis of Frequency Response

The final sub-module has students visualize the phase and magnitude response of transfer functions in the frequency domain. This sub-module's exercises uses the example of an RC circuit that is also used repeatedly during in-class instruction due to its relevance to ECE students.



(a) 4.3 Exercise 4 Output



(b) 4.3 Exercise 5 Output

Figure 9: Sub-Module 4.3 Exercises 4 and 5 (Magnitude and Phase Frequency Response)

The students are to plot the magnitude and phase response to the RC circuit frequency domain transfer function in exercise 4 and 5 respectively, plotting several values for capacitance and resistance in each. The expected results for this exercise are shown in Figure 9.

5 Planned Assessment of Methods

The programming exercises will first be administered in the Fall 2026 term. Each of the four modules corresponds with approximately one-fourth of the course material covered during in-class instructional time, and thus, they will be administered evenly throughout the course. The assessment methods for this work-in-progress study will be conducted throughout the fall 2026 term, with results and analysis being assembled in December 2026 at the conclusion of the term.

The impact of the modules on student outcomes in the course are to be assessed through three components: student performance on each sub-module, student responses to pre- and post-module surveys, and student grades on in-class assignments (exams and quizzes).

Student performance on the modules helps assess whether the modules are closely related enough to in-class concepts that students are able to complete them successfully, and are of sufficient difficulty. The programming exercises for each module are to be assessed for correctness by by passing auto-grader tests, along with manual review of student reflections for each sub-module. Auto-grader tests will depend on specific output being returned from the methods students implement, and thus detailed instructions are included in each exercise related to expected formatting of the output. Multiple assessment cases may be provided, with at least one visible and one hidden case provided where applicable. Following the deadline of each modules exercises, correct code

will be provided to all students. The written reflections to be administered to understand whether the student is able to sufficiently connect the exercises to in-class content. The students will be asked what they did to implement the exercise, draw comparisons between different configurations of the code where applicable, and be given space to add additional comments where desired. At the conclusion of the course, the written responses may be analyzed as an additional dimension for understanding student performance and potential for increase in understanding of the presented topics.

Students are to be given surveys both prior to and following the completion of each sub-module. The first survey, independent of any module, is a two question Likert survey asking students on a scale of 1 (No Experience) to 5 (Extensive Experience) their experience with programming in general, and with Python programming specifically. Then, prior to each sub-module, students are to be given another Likert scale survey asking for their comfortability with each of the topics to be reviewed in the programming sub-module. These topics are a subset of the topics discussed in class, and thus their score on this survey should reflect their level of familiarity following in-class instruction and completion-graded homeworks. The programming sub-modules are to be released to students after the respective topics are covered in class, rather than releasing the entire module at once. After the student has completed and submitted all of the entire module, the students are to be surveyed again, this time with a Likert scale survey asking about all of the topics covered in the module. This survey is to also include a question assessing the student's generative AI (GenAI) usage, with responses scaled from 1 (Never) to 5 (Always). At the end of the term, one additional Likert survey will be administered. This survey will contain the same two questions as the survey at the start of the term, as well four additional an additional questions (one for each module they completed) regarding if the student feels their strength of understanding of each module's concepts has been improved by the exercises, ranging from 1 (Not at All Strengthened) to 5 (Significantly Strengthened). The students will also be given free space to describe any topics they wanted to be covered from the class that were not covered by the exercises, or they believe could be covered more extensively.

For the first time usage of the programming exercises in the fall 2026 term, the exercises will be assigned as optional, extra credit assignments. Typical class sizes for this course have ranged between 60 and 70 students, so the expected sample size of students completing the exercises is estimated to be approximately 20 students. The impact of the exercises on student performance can be compared between those who complete the exercises and those who do not via assignments all students are required to take—nine quizzes, two midterm exams, and one final exam. Additional analysis may be conducted between students who complete some modules, and not others, if this event occurs.

6 Conclusion

The goal of the exercises developed for this study are to improve student understanding of mathematical concepts directly related to electrical engineering. The exercises aim to achieve this through programming exercises, which task students with developing both programmatic solutions to presented mathematics concepts, as well as visualizations of abstract engineering and mathematics concepts. Through the exercises, programming is utilized not only as a means for

helping students understand mathematics by applying libraries or pre-written code, but also as problem solving exercise in itself, as students are tasked with developing systematic methods for solving and visualizing problems.

After the exercises are administered and assessed for the first time, future work may include developing additional assessments for both this class, and its the sequence course (typically taken the following spring semester), and assessing students retention of concepts from this course. Furthermore, as the concepts developed in this class are fundamental to the remainder of the curriculum in ECE, analysis regarding success in this course may be extended to analyze student success in subsequent courses as well.

References

[1] A. K. Ryan Boyd Cartwright and A. Yalcin, "Does it matter who teaches a core mathematics course to engineering undergraduates?" in *2013 ASEE Annual Conference & Exposition*.

[2] A. A. Chaparro, Luis F., *Signals and Systems Using MATLAB® (3rd Edition)*. Elsevier, 2019. [Online]. Available: https://app.knovel.com/hotlink/epub/rcid:kpSSUMATL1/id:kt012EBBQ2/signals-systems-using/continuous-time-systems

[3] R. Belu and A. Belu, "Using symbolic computation, visualization, and computer simulation tools to enhance teaching and learning of engineering electromagnetics," 2009, pp. 14.1333.1–14.1333.18. [Online]. Available: https://peer.asee.org/using-symbolic-computation-visualization-and-computer-simulation-tools-to-enhance-teaching-and-learni

[4] M. Enelund, S. Larsson, and J. Malmqvist, "Integration of a computational mathematics education in the mechanical engineering curriculum."

[5] H. Rahemi and P. LaVergne, "Engineering analysis with matlab programming – an approach to enhance teaching and learning effectiveness," 2009.

[6] G. Bischof, T. Kainz, E. Menard, R. Poetsch, C. Steinmann, M. Sterkl, and C. Tröster, "Bringing differential equations to life by two- and three-dimensional visualizations of numerically simulated dynamic systems," Aug. 2022. [Online]. Available: https://peer.asee.org/bringing-differential-equations-to-life-by-two-and-three-dimensional-visualizations-of-numerically-simula

[7] T. Driscoll and R. Braun, *Fundamentals of Numerical Computation*, ser. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics, 2017. [Online]. Available: https://books.google.com/books?id=W9JEDwAAQBAJ

[8] ——, *Fundamentals of Numerical Computation: Julia Edition*, ser. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics, 2022. [Online]. Available: https://books.google.com/books?id=DiaGEAAAQBAJ

[9] G. R. Hutchison, *Integrating Python into an Undergraduate Mathematics for Chemists Course*, ser. ACS Symposium Series. American Chemical Society, 2021, vol. 1387, p. 123–134. [Online]. Available: https://doi.org/10.1021/bk-2021-1387.ch009

[10] D. Margalit and J. Rabinoff, "Interactive linear algebra," Jun. 2019. [Online]. Available: https://textbooks.math.gatech.edu/ila/index.html

[11] E. Cheever, "Convolution demo and visualization." [Online]. Available: https://lpsa.swarthmore.edu/Convolution/CI.html

[12] phiresky, "Convolution demo." [Online]. Available: https://phiresky.github.io/convolution-demo/

[13] "Myst specification." [Online]. Available: https://mystmd.org/spec

[14] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with numpy," *Nature*, vol. 585, no. 7825, p. 357–362, 2020.

[15] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.