IMPLEMENTING A COHESIVE PROGRAMMING ECOSYSTEM IN MECHANICAL ENGINEERING

by

ZAKARY CHRISTIAN OSTER

B.S., Kansas State University, 2022

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Mechanical Engineering
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas
2024

Approved by:

Major Professor
Jeremy A. Roberts

Abstract

The ability to apply modern solution methods is critical for the success of mechanical engineers. While handwritten solutions were once the state of the art, computer programs provide unparalleled power and speed to solve problems. Because of this, it is critical that academic curriculum teaches students to use programming to solve problems. The current implementation of programming in the department varies, with three different languages being used. The usage rate also varies, with many classes failing to take advantage of the tools provided by programming, and this is reflected by the ability and confidence of students in the department. This project demonstrates the possibility of using one programming language and environment through the mechanical engineering undergraduate curriculum. The completed projects and assignments showcase the varying uses of Python, the Raspberry Pi Pico, Visual Studio Code, and Jupyter Notebooks as it relates to problem-solving in the field of mechanical engineering.

Contents

List of Figures

List of Assignments

List of Abbreviations

CAD	Computer-aided Design
CE	Civil Engineering
CFD	Computational Fluid Dynamics
CHE	Chemical Engineering
CIS	Computer Information Science
CSV	Comma Separated Values
ECE	Electrical and Computer Engineering
EES	Engineering Equation Solver
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
IMSE	Industrial and Manufacturing Systems Engineering
IR	Infrared
LCD	Liquid Crystal Display
LED	Light Emitting Diode
ME	Mechanical Engineering
MNE	Mechanical and Nuclear Engineering
NE	Nuclear Engineering
KSU	Kansas State University
PCB	Printed Circuit Board
Pico	Raspberry Pi Pico
SPI	Serial Peripheral Interface
VSCode	Visual Studio Code

Chapter 1

Introduction and Background

1.1 Introduction

Engineers solve problems. Regardless of field, discipline, and generation, every engineer strives to solve problems. While the need for problem solvers does not change with time, the methods for solving problems certainly do. Consider the process of creating detail drawings. Engineers needed a way to communicate the size, shape, and finish of parts to other engineers, so they hand drew and dimensioned parts. But in the modern day, hand drafting a detail drawing is the last resort. And education has adjusted in kind. While students are taught to draw straight lines (which all lines are) and visualize without the aid of CAD, no class spends time teaching students how to use a drafting table or create detail drawings by hand. Rather, they are taught the basics of visualization and then taught how to use a CAD program, like SOLIDWORKS.

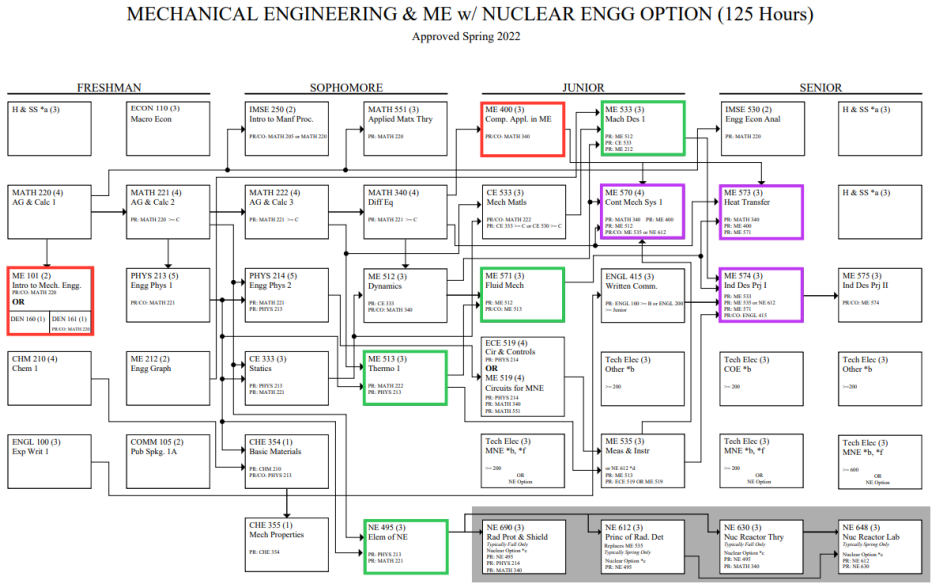
The same could be said for setting the state in a thermodynamics problem, graphing deflection in a load-bearing beam, or finding the Q-value of a fission reaction. At one point, looking up values in a table and plugging them into a calculator or using a ruler and lined paper were state-of-the-art methods for solving these problems. However, as the times change, so do the methods, and with the advent of programming and its ever lowering barrier to entry, these problems can more easily, and more accurately, be solved with a simple script in any number of programming languages.

Despite this, most classes continue to have students spend hours thumbing through tables to solve every problem by hand. In contrast to this, other

classes have recognized the value of modern solution methods and rely heavily on them, only to find that students lack sufficient training to effectively utilize them. Because of this dichotomy, a cohesive and consistent programming ecosystem should be adopted throughout the curriculum of the Mechanical and Nuclear Engineering department.

1.2 Background

The first introduction to programming in MNE curriculum is DEN 161: Engineering Problem Solving, the first-semester introduction-to-engineering class. The class spends a few weeks introducing Python and then a week introducing the Arduino Uno. After this, no programming is used in the curriculum until first semester Junior year, as seen in Figure 1.1 [24].



Red : Teaches Programming
Purple : Uses Programming
Green : Does not utilize Programming, but could

Figure 1.1: MNE Curriculum Flowchart

The next class to use programming is also the only class dedicated to teaching programming. ME 400: Computer Applications in Mechanical En-

gineering begins with an introduction to C++ but then quickly moves to teaching embedded C++ for use on an ESP32. While the class contains phenomenal material as it relates to embedded programming, its fast-paced nature often leaves students confused on the fundamentals of programming.

The next class to utilize programming is ME 570: Control of Mechanical Systems 1. This class uses MATLAB for system analysis and controller design. For the first lab, students complete the MATLAB Onramp, MATLAB's own introductory course to the language. This Onramp, however, does not cover the basics of the control package that the course relies on. Due to the inadequate foundations in programming, many students struggle with the new language, making it difficult to understand the new concepts being taught, such as a PID controller.

The final class that utilizes programming is ME 574: Interdisciplinary Design Project 1, better known as Senior Design 1. This class revolves around the completion and fabrication of a single product. This product always has some electromechanical aspects that need to be controlled by a microcontroller. Neither the device nor language used is moderated by the class instructors. Since this is a project based class where each team is responsible for its own decisions and the project changes every semester, it will not receive a dedicated chapter or assignment redesign.

As it currently stands, the MNE department makes use of three different programming languages in required classes: C++, MATLAB, and Python. While each of these languages and environments come with advantages, and the idea of learning multiple languages has merit, many students find themselves confused by the variety. Rather than feeling confident in one language, they feel unconfident in three.

In addition to the unstable foundation and lack of consistency, the use of programming is absent from several classes that would benefit from its inclusion. Classes that spend substantial time with property tables, iterative design, or numerical methods showcase the power of programming in the field of mechanical engineering.

Based on experience as an instructor of ME 570 and ME 574 for the better part of four years, the majority of students struggle with using MATLAB and C++. This makes it difficult for students to learn engineering principles because they do not understand the tool being used to teach it. But, at the same time, it is important to use modern techniques to solve problems.

1.3 Scope of Work

While the motivation for this work relies on the idea that a more consistent approach to learning and applying programming would benefit students and improve their problem-solving abilities, this project does not seek to prove or verify this claim. Neither does this work set out to prove that the proposed solution is the best solution for the assumed problem. This project merely seeks to identify a potential path forward while demonstrating both the advantages and disadvantages of the curriculum.

This work will take time to point out benefits and drawbacks compared to the current curriculum, but given that no other possible solution is considered, it would be premature to draw conclusions relating to the "best" path forward. However, there will be a brief discussion of alternative options in the concluding chapters of the paper.

1.4 Structure of Work

Since this body of work does not set out to prove a hypothesis, opting instead to demonstrate a consistent usage and application of programming across the MNE curriculum, the format of this paper will be adjusted accordingly. The next chapter, titled "A Cohesive Programming Curriculum," will discuss the foundation of the new implementation, such as the language, hardware, and development environment. It will also discuss the general goal of the changes.

Subsequent chapters will each be dedicated to a class in the MNE curriculum. These chapters will give a course overview, describe the new or altered assignments, and give a list of deliverables for that course. Complete assignment descriptions and implementations are not contained within the chapters themselves, only representative portions are included. The assignments are contained in a repository on GitHub, which can be found in Appendix A. Each chapter will have a folder of the same name dedicated to it in the repository. This folder will contain everything an instructor would need to assign and grade the new assignments.

Chapter 2

A Cohesive Programming Curriculum

2.1 Curriculum Proposal

The primary concerns addressed in Chapter 1 are as follows:

1. Students lack a solid foundation in the fundamentals of programming.
2. Programming is an under utilized problem-solving technique.

To address the first issue, this paper would like to make two proposals: the adoption of a single programming language and the restructuring of electronics classes in the department. First, the goal of adopting a single programming language is to give students a more thorough understanding in a single language rather than a shallow understanding of four languages. The language chosen in this project is Python. In conjunction with Python, a consistent environment should be used as well. A consistent environment will prevent unnecessary confusions from IDE specific features and setup requirements. The chosen environment for this project is Visual Studio Code. To replace the work done on the Arduino Uno and ESP32, a Raspberry Pi Pico will be used thanks to its seamless integration with both Python and Visual Studio Code. For more details regarding these choices, see Sections 2.2-2.5.

Second, the goal of restructuring electronics classes is to give students a dedicated "fundamentals of programming" class rather than a microcontroller

class that teaches programming out of necessity. This change, however, will not be addressed for the bulk of this work. Significant changes to the courses required in the curriculum are difficult to pass, and always come with trade-offs. As such, discussion of this topic will be saved for reflection in the conclusion of this paper.

To address the second primary concern, the under-utilization of programming in the curriculum, one more change is being proposed: the addition of assignments or projects that make use of programming software to relevant classes. The majority of this paper, as well as the supporting repository, is dedicated to creating assignments that utilize programming for classes that do not have one, translating assignments from other languages in classes that do utilize programming, and creating a list of Python packages that could be used throughout the department.

The goal of these changes is twofold: teaching students how to use modern problem-solving techniques to find solutions and giving students a chance to reinforce the programming skills they were taught. By giving students a more consistent approach to programming, as well as more use cases for the value of programming, students will become more confident and more capable as problem solvers and engineers.

Through the course of this project, many Python packages and Visual Studio Code extensions will be utilized. Thankfully, the installation and usage steps are simple, and can be combined into a single step in both cases. The Visual Studio Code extensions can all by downloaded at once using an extension pack called KSU Mechanical Engineering Extension Pack. The necessary Python packages can also all be installed at once from a requirements file. In some cases, it may be preferable to use Anaconda, a Python distribution that comes many standard packages, to prevent the need for students to install packages on their own. To see a full list of the applications, packages, and extensions and their versions, see Appendix B. For installation instructions, see the GitHub repository linked in Appendix A.

2.2 Python

Python is a multipurpose, interpreted programming language that places emphasis on code readability. The language was originally introduced by Guido van Rossum in 1991 and has gone through three major versions, the most recent being the aptly named Python 3. Using Python, particularly for

first time programmers, comes with the following benefits:

- Python is an interpreted language; therefore, users do not need to install and configure a compiler. The setup process for a compiler can often be confusing and a major barrier to entry for new users. Since Python does not have a compiler, the only download that is necessary to run a .py file is Python itself. The lack of a compiler does come with some downsides, namely speed, but these will be addressed later on.

- Python is designed to be human-readable. This means that the syntax of the language closely matches the structure of the plain English. This emphasis on readability greatly reduces the burden on new users and bolsters code comprehension, even in more experienced coders. Consider a program that prints the phrase "Hello world!" to the terminal. In C++, this would be done with the following code:

```
#include <iostream>

int main() {
    std::cout << "Hello world!" << std::endl;
    return 0;
}
```

  Not only is this code not human-readable, but it requires the use of imports, functions, return values, namespaces, and non-obvious syntax. In Python, however, only one line of code is needed:

```
print("Hello world!")
```

- Python is a multipurpose, general-use language. The language is adept at everything from data science to building a website to powering a microcontroller (using MicroPython). With such a wide breadth of use cases, learning the language opens the door to countless different applications and projects.

- Python is open-source and currently one of the most used languages. Thanks to its popularity, many tools exist to make programming with Python easier, more powerful, and more accessible such as integrations with Jupyter Notebooks and Visual Studio Code. However, this can potentially be two-edged sword, as will be discussed later.

- Python has an easy to use package manager called pip, which is a recursive acronym for pip installs packages. This package manager makes it incredibly easy to both install and use non-standard libraries in Python.

Python provides a solid starting point for new programmers while also being fully capable of high-level, advanced programming making it a good language for users of all skill levels. However, it does not come without concerns of its own, most notably version controlling. This will be addressed at length in a future chapter.

2.3 MicroPython

MicroPython is a slimmed version of Python that is designed to run on microcontrollers, such as the Raspberry Pi Pico or ESP32. This Python derivative removes some high level features in exchange for a smaller interpreter and a machine library dedicated to interacting with the hardware of the microcontroller, which Python would normally not interact with.

While MicroPython comes with all the benefits of Python, it also comes with one of the drawbacks of Python: speed. Unlike traditional low level languages like C/C++, MicroPython is not compiled, making it significantly slower than comparable languages. In some applications, the difference in speed has no effect on the application. Others, like a control system, heavily depend on fast loop times and are greatly impacted by speed.

2.4 Raspberry Pi Pico

The Raspberry Pi Pico is a small-but-powerful microcontroller made by Raspberry Pi. A microcontroller is an electronic input-output device capable of running uploaded code. It is differentiated from a microprocessor, such as a Raspberry Pi 5 or a desktop computer, by the fact that it does not run an operating system.

The using a Pico, as a newer competitor in the saturated microcontroller market, begs the question of "why not use the Arduino Uno or ESP32?" While all three devices support ADC, SPI, I2C, and have around 40 GPIO pins, only the ESP32 and Pico support MicroPython, as well as built in wireless and Bluetooth connectivity.

A case could be made for using the ESP32 over the Pico, given its rise in popularity in the home automation space. However, the ESP32 comes with greater restrictions in GPIO usage and the majority of its documentation and drivers are for C++ programs rather than MicroPython, giving the Pico an edge in user-friendliness.

As was the case with Python and MicroPython, the Pico does not come without drawbacks. Almost all sensors come with drivers written in C++ for the Arduino Uno or ESP32. This is not the case for the Pico. Some sensors made specifically to work with the Pico come with drivers, and others may be found on GitHub thanks to the work of another tinkering user, but many pieces of hardware do not have driver support for the Pico. This could end up being a limiting factor for students trying to do unguided projects.

2.5 Visual Studio Code

Visual Studio Code is a cross-platform, highly customizable development environment created by Microsoft. It is one of the most used and most accessible development environments available, and supports use with any language. VSCode can almost be thought of as a Word processor where spell check looks for syntax errors and keywords instead of grammatical mistakes. By installing extensions, which can be done from inside VSCode, developers can write code for nearly any microcontroller or processor and in nearly any language.

For the purposes of the Mechanical Engineering Department, extensions for Python, MicroPython and the Pico, and Jupyter Notebooks (a type of interactive programming environment that will be utilized later) will be used. The extensions are both easy to use and install, but share a downside with Python: version control. Since these extensions are always in active development and get updated to work with the newest version of Python and Visual Studio Code, which may not always be desirable. This issue will be discussed in greater detail in a later chapter.

2.5.1 Jupyter Notebooks

Jupyter is an open-source interactive programming environment that allows text and code to seamlessly integrate. With more than 40 supported languages, Jupyter gives educators the ability to place formatted text (through

the use of markdown) next to linted code. For an example of a Jupyter Notebook in VSCode, see Figure 2.1.

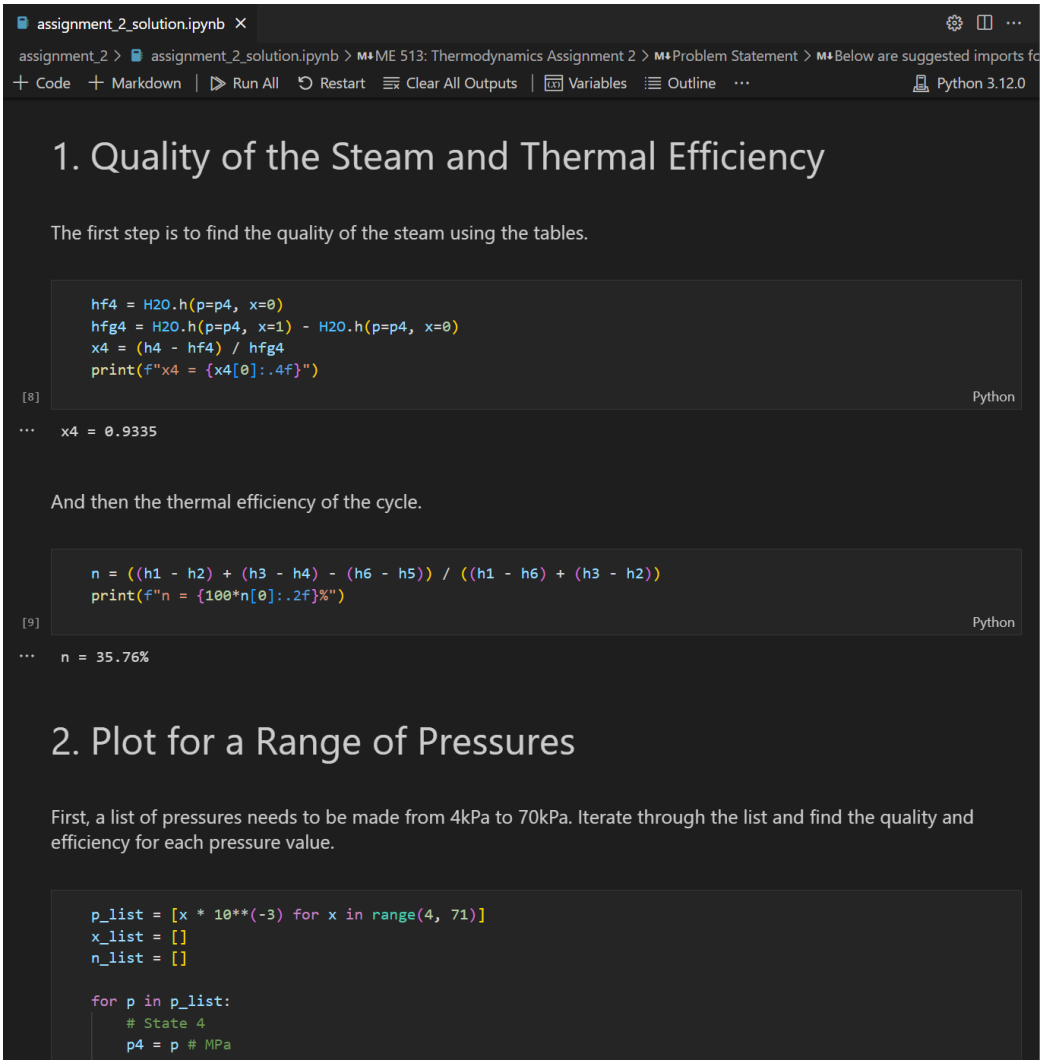


Figure 2.1: Example of a Jupyter Notebook in VSCode

The flow provided by Jupyter Notebooks works well, both for giving assignments and doing in class examples. Jupyter Notebooks also have slideshow capabilities, which allows interactive code to fit directly into a lecture presentation without needing to switch between monitors or tabs.

Chapter 3

DEN 161: Engineering Problem Solving

3.1 Course Overview

DEN 161: Engineering Problem Solving is a lab-style course that complements the lecture-oriented DEN 160: Engineering Orientation. The class focuses on providing hands-on, problem-solving experiences through projects from multiple engineering disciplines. While these projects serve as the students' introduction to different engineering disciplines, they also develop the core tools needed to be a successful engineer.

The current iteration of DEN 161 has three sections of interest to this body of work: data analysis using Microsoft Excel, data analysis using Python, and embedded programming using an Arduino Uno.

Microsoft Excel is used to introduce the concept of data analysis to students. Students are tasked with manipulating data and finding different statistical properties of given data. This proves to be an effective point of entry given to data analysis since many students are familiar with Microsoft Excel or Google Sheets.

Python is then introduced as an alternative method of solving the same problems. The lectures and assignments focus on data calculations to verify designs and general code inspection to understand how the program works. These lectures are done using Jupyter Notebook in Anaconda's Spyder IDE.

Following the introduction to Python, a Stoplight Activity is assigned. This project introduces students to circuitry and microcontrollers through

the creation of a stoplight using 3 LEDs and an Arduino Uno. The Arduino Uno is programmed using C++ in the Arduino IDE.

3.2 Project Redesign

Thanks to the solid programming core created by the instructors, the proposed changes are minor, and result in minor changes to the current curriculum. Two changes are proposed: the adoption of Visual Studio Code as a development environment and the migration from Arduino Unos to Raspberry Pi Picos.

The class currently uses an IDE by the name of Spyder. Spyder is a popular IDE for data science applications and has the to ability to seamlessly integrate with Jupyter Notebooks. However, Spyder does not have the ability to work with a MicroPython device, such as the RPi Pico. Visual Studio Code, on the other hand, has an extension that integrates Pico controls directly into the interface, making it a one-stop-shop for both the data analysis and embedded systems development in DEN 161. Visual Studio Code also has Jupyter Notebook extensions that allow for a first class experience.

The second proposed change is transitioning from the Arduino Uno to a Raspberry Pi Pico. The reason for this change is twofold. First, the Pico can run using MicroPython, a lightweight implementation of Python, which has the same syntax as Python. This allows students to focus on understanding one language, Python, rather than learning both Python and C++. Switching to the Pico also opens the door to using a single development environment. While the Arduino can be programmed using Visual Studio Code, the setup process is non-trivial, and requires a strong understanding of the operating and file system of the computer.

With these changes in mind, the first two assignments created serve as a segue from using Excel for data analysis to using Python. The first of the two, Assignment 3.1, requires students to convert tire rpm data to speed data, find several statistical properties of the data, and then plot it. The assignment is done in class both in Excel and Python to showcase the differences.

DEN 161: Assignment 3.1

Problem Statement

You are given data for the tire rotations per minute in a file called ‘tire_rpm.csv‘ and are tasked with finding the max, min, mean, mode, and median speed of the

vehicle. Bonus points for a graph!

Problem Solution

For the full solution, see Appendix A. The following shows the process of reading information from a .csv file.

To solve this problem, let's split it into steps and address the issues one at a time.

1. Read and parse the data from the .csv file
2. Convert the RPM data into vehicle speed
3. Find the max, min, mean, mode, and median

Step 1: Read and parse the data

To begin, we will import the built-in csv library. This Python package makes it easy to parse data from csv files, and takes away the burden of writing code to split the data. We will also import the math library.

```
import csv
import math
```

We will start by creating an empty list for our rpm values to be stored in. Next we will open the .csv file and read it.

```
with open("tire_rpm_example.csv", "r") as file_contents:
csv_reader = csv.reader(file_contents,delimiter=",")
rpm = []
for rpm_row in csv_reader:
    for rpm_value in rpm_row:
        rpm.append(int(rpm_value))
print(f"Tire RPM: {rpm}")
```

After completing the in-class example, a similar assignment is given for homework, as shown in Assignment 3.2. Rather than writing the code from scratch, students must make several changes and add comments to the file.

DEN 161: Assignment 3.2

Problem Statement

Make the following changes to the given Python file:

1. Change the input file to the homework data set.
2. Change the tire diameter to utilize user input.
3. Change the print statements to have 0 decimal places.

4. Anywhere there is a #COMMENT, leave a comment explaining what the following lines of code do.

Optionally, complete the bonus assignment question at the bottom of the file.

Problem Solution

For the full solution, see Appendix A. The following shows the changed file name, user input, and descriptive comments.

```
# Import statements to make library methods
# available in this file
import csv
import math
import statistics
from matplotlib import pyplot

# Open and read data from a csv file into Python
with open("tire_rpm_homework.csv","r",
          encoding="utf-8") as file_contents:
    csv_reader = csv.reader(file_contents,delimiter=",")
    rpm = []
    for rpm_value in next(csv_reader):
        # Turn the rpm_value (which would be a string)
        # into an integer and add it to a list of rpms
        rpm.append(int(rpm_value))

# Ask the user to input the tire diameter
tire_diameter = input("What is the tire diameter?")
```

The third assignment, shown in Assignment 3.3 was translated from the current Stoplight project used in the class [3]. An example file demonstrating how to operate the LEDs serves as the starting point for the project. To complete the assignment, students need to rearrange the `pin.high()` calls and change the sleep timings.

DEN 161: Assignment 3.3

Problem Statement

Using the Raspberry Pi Pico and three LEDs given out in class, write a program that will operate the LEDs like a stoplight. For bonus assignment credit, write a program that will blink SOS in Morse code.

Problem Solution

For the full bonus assignment solution and the example file, see Appendix A. The following shows the entire solution to the stoplight portion of the assignment.

```
from machine import Pin
import time

# Declare the pins and the pin mode
red_led = Pin(18, Pin.OUT)
yellow_led = Pin(17, Pin.OUT)
green_led = Pin(16, Pin.OUT)

# Begin looping phase
while True:
    # Turn on the red led for 5 seconds
    green_led.low()
    yellow_led.low()
    red_led.high()
    time.sleep(5)

    # Turn green led on and red led off for 5 seconds
    green_led.high()
    red_led.low()
    time.sleep(5)

    # Turn yellow led on and green led off for 2 seconds
    yellow_led.high()
    green_led.low()
    time.sleep(2)
```

The proposed changes aim to increase student understanding by reducing the number of systems they are introduced to. Instead of two languages and two editors, students will only need to learn one language in one editor. The work done by these projects directly correlates with Abet Student Outcomes 6 and 7 and weakly correlates with Outcomes 1 and 3, as seen in Appendix C.

3.3 Project Redesign Assessment

The project proposal for DEN 161 is unique from the other classes in this work because the proposed changes were implemented into the standard class curriculum. The second week of Python was replaced with the tire RPM problem and the Arduino Uno was replaced by the Raspberry Pi Pico in the stoplight project. The adoption was largely successful, and the instructors felt that it was a step in the correct direction. That is not to say that there were no pain points.

The biggest hindrance encountered in the Fall 2023 was general file system comprehension. Many students did not know where downloaded files go or how the file system (both File Explorer and Finder) were structured. This proved to be an issue for both the tire RPM question and the stoplight example. For Python to locate the .csv file with the RPM data, either the full path to the file needs to be provided, relative to the current working directory, or the file needed to be in the same folder as the script. To get the MicroPico extension to work correctly in Visual Studio Code, a working directory needs to be set.

Both of these issues can likely be solved with the same two methods. First, time will be spent in a previous lecture to talk about the file system. This is basic information that is required to effectively use a computer. Second, files for assignments will be distributed in zip files. This will nearly guarantee that files have the correct relative location. This will introduce a new issue of trying to link and edit files in a zip folder, but hopefully it is an easier fix than trying to hunt down files on student's laptops.

The Fall 2023 semester also found that using Visual Studio Code for both the data analysis and Pico portions of the class is better for students than using Spyder for the data analysis and VSCode for the Pico.

An additional pre-class assignment will also be created that will require students to verify that they have done the necessary installations before coming to class. Though this is not an issue directly caused by the changes made, it did directly inhibit student learning.

The final issue, and perhaps the most concerning, is the unreliability of the MicroPico extension when not used exactly as intended. While no connection issues ever occurred during testing, a non-trivial number of students experienced issues with the MicroPico extension. Since no issues have been encountered by the instructors at this point, it is hard to pinpoint the exact cause. Fixes for different issues can be found in the "usage-and-installation"

folder in the GitHub Repository.

3.4 Project Deliverables

In the GitHub repository associated with this paper, which can be found in Appendix A, the folder titled "3-engineering-problem-solving," there are several folders containing different problems and examples. The folder also contains a README that details what is in each file and what software is needed to complete the assignments.

The first folder is a tire RPM example problem that is to be completed in class, after having completed the Excel lectures and the introduction to Python lectures. This will show students the benefits of using a language like Python over Excel. To go along with the example problem is a similar tire RPM homework assignment. The code needed to complete the assignment is all provided to students, with a handful of tasks left up to them, as detailed in the homework file.

Finally, the stoplight folder contains the starter code to give students for the stoplight assignment, a solution key, and a file demonstrating Morse code on the Pico. Installation instructions for the using the Pico can be found in the "usage-and-installation" folder in the repository.

Chapter 4

ME 513: Thermodynamics

4.1 Course Overview

ME 513: Thermodynamics is a lecture based class that focuses on the interplay of heat, work, and energy in both open and closed loop systems. While there is no lab portion to the class, significant lecture time is spent solving problems and working examples, and a new homework is assigned every lecture to give students a chance to apply the new concepts taught that day.

Most problems in thermodynamics follow a similar solution path. First, students must recognize the different states in the system. Once the different states are identified, the values for temperature, power, energy, entropy, etc. must be found for each state. The exact values needed, and retrievable, vary depending on the given system. This process is known as setting the state. Once the state has been set, known laws of thermodynamics can be applied and the question can be solved.

For most questions, the majority of time is spent setting the state. This involves using known properties, usually given in the problem statement, to look up values in property tables in a thermodynamics textbook. This often proves to be a very time-consuming process, especially when interpolation is required to get property values. And once iteration is introduced, such as in a design problem, manually searching through tables becomes impractical, if not impossible. This is where the use of a programming language would become highly beneficial.

Thermodynamics, as it currently stands, make no use of programming to complete assignments. This can be attributed to ME 513 coming prior to ME

400, the only programming class in the curriculum. However, with the recent programming additions made to DEN 161, students have been introduced to programming and, with the help of an example, should be capable of solving these questions as a small project or assignment.

4.2 New Assignments

Since no assignments currently make use of programming, two new assignments have been adapted from *Fundamentals of Engineering Thermodynamics, 9th Edition* [15] to demonstrate how programming could be beneficial to the course. Both assignments will make use of a Python library called PYroMat [12], a free, open source library dedicated to making thermodynamic properties readily available. They will also both make use of Jupyter Notebooks for cleanly presenting questions, commentaries, and code while solving the question.

The first assignment, Assignment 4.1, is a five state system that does not require iteration or plotting. It follows the structure of a typical question in thermodynamics well. The student must identify the states, list the known properties for each state, and then either search for additional values in a table or calculate them with known values. After this, the laws of thermodynamics are applied, equations are balanced, and the questions solved.

ME 513: Assignment 4.1

Problem Statement

Water is the working fluid in a Rankine cycle. Steam exits the steam generator at 1500 lbf/in^2 and 1100°F. Due to heat transfer and frictional effects in the line connecting the steam generator and turbine, the pressure and temperature at the turbine inlet are reduced to 1400 lbf/in^2 and 1000°F, respectively. Both the turbine and pump have isentropic efficiencies of 85%. Pressure at the condenser inlet is 2 lbf/in^2, but due to frictional effects the condensate exits the condenser at a pressure of 1.5 lbf/in^2 and a temperature of 110°F. The condensate is pumped to 1600 lbf/in^2 before entering the steam generator. The net power output of the cycle is 1x10^9 Btu/h. Cooling water experiences a temperature increase from 60°F to 76°F, with negligible pressure drop, as it passes through the condenser. Determine for the cycle:

1. the mass flow rate of steam, in lb/h.
2. the rate of heat transfer, in Btu/h, to the working fluid passing through the steam generator.

3. the thermal efficiency.

4. the mass flow rate of cooling water, in lb/h.

Be sure to leave specify all assumptions and comment on the functionality of the code. To access the thermodynamic tables for water, import PYroMat using the cell below.

Problem Solution

For the full solution, see Appendix A. Below is an example of importing PYroMat, setting the correct units, and creating an object for a specific substance.

```
import pyromat

pyromat.config["unit_energy"] = "BTU"
pyromat.config["unit_force"] = "lbf"
pyromat.config["unit_mass"] = "lb"
pyromat.config["unit_temperature"] = "F"
H2O = pyromat.get("mp.H2O")
```

To find specific property data, such as enthalpy and entropy, call the appropriate method on the substance object, H2O in this case, and pass the temperature and pressure.

```
# State 1
p1 = 1400 # lbf/in^2
T1 = 1000 # deg F
h1 = H2O.h(T=T1, p=p1) # Btu/lb
s1 = H2O.s(T=T1, p=p1) # Btu/(lb*F)
```

The second assignment, Assignment 4.2, better showcases the benefits of programming. This question uses both iteration and plotting to show the changes in quality and thermal efficiency as the pressure increases. If done by hand, this would be a difficult and tedious task. But when programming, it only requires a few extra lines of code.

ME 513: Assignment 4.2

Problem Statement

Steam heated at constant pressure in a steam generator enters the first stage of a supercritical reheat cycle at 28 MPa, 520°C. Steam exiting the first-stage turbine at 6 MPa is reheated at constant pressure to 500°C. Each turbine stage has an isentropic efficiency of 78% while the pump has an isentropic efficiency of 82%. Saturated liquid exits the condenser that operates at constant pressure, p.

1. For p = 6 kPa, determine the quality of the steam exiting the second stage of the turbine and the thermal efficiency.
2. Plot the quantities of part (1) versus p ranging from 4 kPa to 70 kPa.

Be sure to leave specify all assumptions and comment on the functionality of the code.

Problem Solution

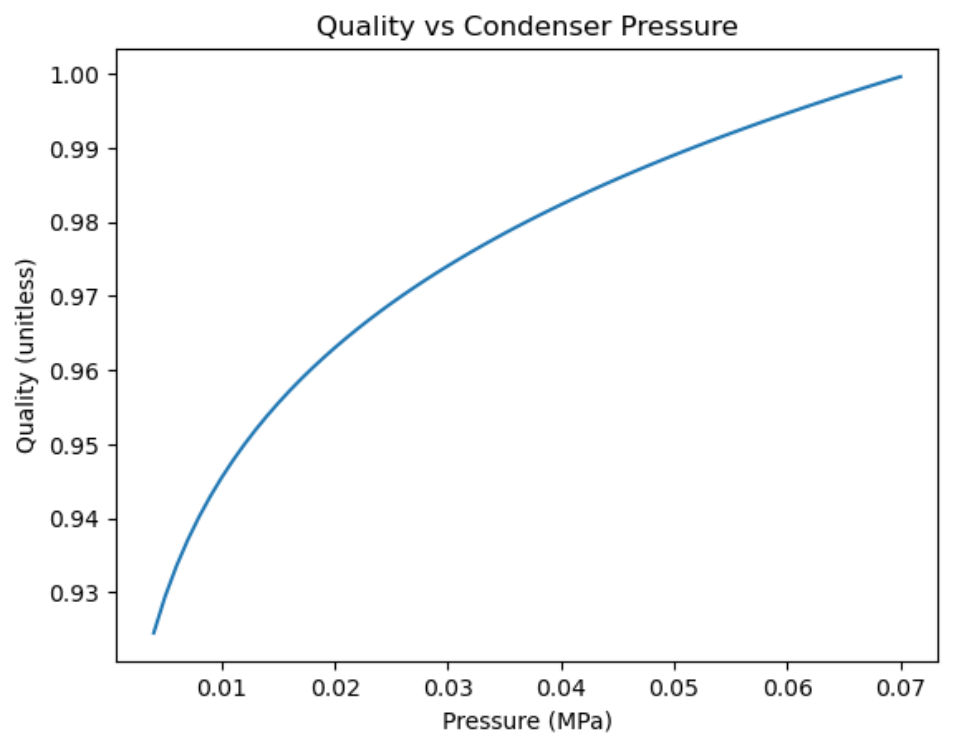
For the full solution, see Appendix A. Below is an example of iterating through varying pressure values to create a list of quality values.

```
p_list = [x * 10**(-3) for x in range(4, 71)]
x_list = []
n_list = []

for p in p_list:
    # State 4
    p4 = p # MPa
    x4s = H2O.T(s=s3, p=p4, quality=True)[1]
    h4s = H2O.h(p=p4, x=x4s)
    h4 = h3 - nt2 * (h3 - h4s)

    # State 5
    p5 = p # MPa
    h5 = H2O.h(p=p5, x=0)
    v5 = H2O.v(p=p5, x=0)

    # State 6
    h6 = h5 + 1000 * v5 * (p6 - p5) / (np1)

    # Find the quality of the steam
    hf4 = H2O.h(p=p4, x=0)
    hfg4 = H2O.h(p=p4, x=1) - H2O.h(p=p4, x=0)
    x4 = (h4 - hf4) / hfg4
    x_list.append(x4)

    # Find the thermal efficiency of the cycle
    n = ((h1-h2)+(h3-h4)-(h6-h5))/((h1-h6)+(h3-h2))
    n_list.append(n)
```

With the quality and pressures in hand, a graph can be generated with matplotlib.

```
from matplotlib import pyplot as plt
```

```
plt.figure()
plt.plot(p_list, x_list)
plt.title("Quality vs Condenser Pressure")
plt.xlabel("Pressure (MPa)")
plt.ylabel("Quality (unitless)")
plt.show()
```



Using programming in this manner gives students a better idea of how real problems are solved by introducing them to a more efficient and powerful solution method. These questions would also directly correlate to Abet Student Outcomes 1, 6, and 7 and weakly correlate to Outcome 3, as seen in Appendix C.

4.3 Project Deliverables

In the GitHub repository associated with this paper, which can be found in Appendix A, the folder titled "4-thermodynamics" contains both the problem statements and solution guides for the two questions introduced in the previous section. The folder also contains a README that details what is in each file and what software is needed to complete the assignments.

For these projects to be added to the class, the instructor would simply need to give the skeleton files to students as a problem statement. It may be beneficial to use Assignment 4.1 as an in-class example both to serve as a reminder of how to use Python with Jupyter Notebooks as well as a demonstration of how to use PYroMat to access material properties and change property units.

Chapter 5

NE 495: Elements of Nuclear Engineering

5.1 Course Overview

NE 495: Elements of Nuclear Engineering is a lecture based class that focuses on introducing the fundamental concepts of chemistry and physics that form the field of nuclear engineering to mechanical engineering students. The class also serves as the sole entry point into the nuclear program at K-State.

As class with no lab section, homework assignments and tests make up the majority of the grade for the class. Currently, the class also has one project where students use a Geiger counter developed for the class to measure ionizing radiation.

As is the case with ME 513: Thermodynamics, this class comes before ME 400 and, therefore, does not utilize programming for any homework assignments or projects. With the changes to DEN 161, and with the addition of programming to other classes, such as ME 513, students should be able to solve these problems.

5.2 New Assignments

While no assignments currently make use of programming, many questions lend themselves nicely to being solved with programming. This is thanks to the heavy reliance on tables and constants used in solving NE problems. To demonstrate, three questions, one from an exam and two from a homework,

have been solved using Python with the help of a few open source libraries: mendeleev [13], physdata [7], and scipy [26].

Unfortunately, no single library contains the full collection of properties to solve the range of questions presented in NE 495. While inconvenient, this issue serves to highlight the benefits of making an MNE library that can wraps around the necessary packages

The first assignment, Assignment 5.1, taken from the first exam in the class, is a Q-value question that uses the mendeleev library to pull element properties. The library contains information on the elements and their isotopes that, once familiar, provide quick access to the mass and abundance values needed to solve Q-value problems. Unfortunately, the library outputs a list of isotopes and has no direct method for retrieving a specific mass number.

NE 495: Assignment 5.1

Problem Statement

It has been proposed to extract uranium from seawater to produce nuclear reactor fuel. Assume that the total volume of the oceans is 1.3×10^9 km^3, the uranium concentration in the ocean is 3 parts per billion water molecules, and the total annual electricity consumption in the U.S. is 4,095 Billion kWh (1 kWh = 3.6×10^6 J). Assume further that every single 235U nucleus fissions according to the reaction n + 235U $\rightarrow$ 139Ba + 95Kr + 2 n and that 30% of all the energy can be recovered for electrical production. How many years could the oceanic uranium power the U.S.?

Be sure to specify all assumptions, provide commentary on the numbers being calculated, and comment on the functionality of the code. The necessary imports are included below.

Problem Solution

For the full solution, see Appendix A. Below is an example of importing and using the mendeleev library to get atomic weights.

```
import mendeleev as md

H = md.element("Hydrogen")
O = md.element("Oxygen")

water_molecule_mass = 2*H.atomic_weight+O.atomic_weight
```

Other physical constants can be found in scipy's constants library.

```
from scipy.constants import physical_constants
```

```
# 931.5 MeV / amu
c_2 = physical_constants[
    "atomic mass constant energy equivalent in MeV"][0]

# 1.008 amu
neutron_mass=physical_constants["neutron mass in u"][0]
```

The second and third questions, both in Assignment 5.2, are taken from Homework 20 and deal with photon interactions. These questions make use of the physdata library to get attenuation coefficients for different molecules (water, in this example). Similar to mendeleev, this library does not have an easy, built-in method for getting this coefficient values at non-standard energy values. To get around this, students are encouraged to use numpy.interp to find the desired value.

NE 495: Assignment 5.2

Problem Statement

What fraction (not percent) of 2.5 MeV photons interact within 1 foot of water? What fraction (not percent) of 2.5 MeV photons are absorbed by 1 foot of water?

Problem Solution

For the full solution, see Appendix A. Below is a walkthrough of the solution to the first question.

First, get the table of attenuation coefficients using physdata. The list is converted to a numpy array for easier handling in the next step.

```
# cm-1
water_table = np.array(
    xray.fetch_coefficients("water", 1))
```

Next, interpolate using ‘numpy‘’s ‘interp‘ function to get the correct coefficient. Here column 0 is the energy levels and column 1 is the interaction coefficient.

```
mew = np.interp(2.5, water_table[:,0], water_table[:,1])
```

Using the equation $\frac{I}{I_0} = exp(-\mu x)$, we can get the ratio that does not interact with the water. To get the ratio that does interact with the water, take $1 - l_{ratio}$.

```
x = 12 * 2.54 # 12 inches * 2.54 cm/inch
I_ratio = np.exp(-mew * x)
interact_ratio = 1 - I_ratio
print(f"I/I0 = {interact_ratio:.3f}")
```

Using programming in this manner gives students a better idea of how real problems are solved by introducing them to a more efficient and powerful solution method. These questions would also directly correlate to Abet Student Outcomes 1, 6, and 7, as seen in Appendix C. With the addition of a question that utilizes basic plotting, a weak correlation to Outcome 3 could also be added.

5.3 Project Deliverables

In the GitHub repository associated with this paper, which can be found in Appendix A, the folder titled "5-elements-of-nuclear-engineering" contains both the problem statements and solution guides for the three questions introduced in the previous section. The folder also contains a README that details what is in each file and what software is needed to complete the assignments.

For these projects to be added to the class, the instructor would simply need to give the skeleton files to students as a problem statement. It may be beneficial to use one of the questions as an in-class example both to serve as a reminder of how to use Python with Jupyter Notebooks as well as a demonstration of how to use the data libraries to access material properties.

Chapter 6

ME 400: Computer Applications in Mechanical Engineering

6.1 Course Overview

ME 400: Computer Applications in Mechanical Engineering is a lecture based class with one lab per week. The lectures initially emphasis learning the basic tools of programming, looping, functions, classes, etc., while the remainder of the class focuses on embedded programming and microcontroller hardware, which is the primary objective of the class.

To complete labs, students use an ESP32, a powerful microcontroller made popular by the home automation community, programmed with C++, an industry standard in embedded applications. Since the class does not have any circuit prerequisites and does not teach, nor have the time to teach, circuit theory, the class instructors have created PCBs that are plug-and-play for students to use. These custom PCBs are designed to have the ESP32 as well as several other electronic components plug directly into it to give them the ability to work together. Additional components include a small LCD touchscreen, buttons, LEDs, and buzzers.

The labs in ME 400 take an iterative approach where subsequent labs will add features and functionality on top of the previous week's lab assignment. This gives students the experience of completing a relatively complex project without overwhelming them from the start.

ME 400 serves as the primary introduction to programming in the mechanical department. It is also the only class that spends significant teaching

about programming and the only class that gives instructions on microcontrollers. Unfortunately, the class comes late in the curriculum, after many classes that would benefit from using programming, and focuses the majority of its time on embedded programming, often to the detriment of a fundamental understanding of programming. This is a topic of much debate, and will be addressed in the concluding chapters.

6.2 Project Redesign

Since ME 400 is a programming course, no new assignments are being created for this project. Instead, a pre-existing lab has been translated from C++ on the ESP32 to MicroPython on the Raspberry Pi Pico. The translated lab exercise, partially shown in Assignment 6.1, is the concluding assignment in a string of lab exercises dedicated to programming the game "Simon Says" [2].

The goal of the exercise is to create a replayable, memory-based game that imitates the children's game Simon Says. To do this, students have to use the LCD screen, five buttons, four LEDs, a buzzer, and an IR remote. The lab exercise chosen aims to demonstrate how the Pico and MicroPython can handle the same projects and electronics that the ESP32 and C++ can.

ME 400: Assignment 6.1

Problem Statement

For the full problem statement, please see Appendix A. Complete each of the following steps. When you have completed all steps assigned, upload the files associated with the exercise on Canvas. Make sure the filenames you upload as part of your exercise submission have the exact names specified in each steps of this exercise. If the filenames do not match exactly, then your submission will not be graded.

Background

Simon is an electronic game of short-term memory skill invented by Ralph H. Baer and Howard J. Morrison and programmed by Lenny Cope. The game is implemented using a combination of hardware and software, and the physical interface consists of four quadrants as shown in Figure 1.

When the game starts, a random sequence is presented to the user by lighting the associated quadrants and playing a tone that is specific to each quadrant. Then the user is required to repeat the sequence. If the user succeeds, then the series becomes progressively longer and more complex. Once the user fails to enter the correct sequence, the game is over.

A simple version of this game will be implemented as part of this exercise. This task will test your understanding of incorporating and writing functions, passing arguments to functions, implementing logic in loops, implementing branching logic, and working with arrays.

Submission

Upload the files exercise_07.py to Canvas once the exercise has been completed. The code will be evaluated as follows:

1. It will be evaluated to make sure the proper coding techniques were used to implement the associated functionality.
2. It will be evaluated to ensure the associated code runs successfully.
3. It will be evaluated to ensure the associated code returns accurate and complete results.
4. It will be evaluated to ensure that the correct file names specified for each problem were uploaded to Canvas.
5. It will be evaluated to make sure the implementation uses correct indentation.
6. It will be evaluated to determine if suitable comments have been included.

Problem Solution

For the full solution, see Appendix A. Below is an example of importing the drivers and creating objects for each piece of hardware used.

```
import time
from random import randrange, seed
from machine import Pin, PWM, SPI

from ir_remote_driver import IRRemote
from lcd_display_driver import Display, color565,
                               XglcdFont
```

The infrared remote will be attached to GPIO pin 15 and set as an input.

```
ir_remote = IRRemote(Pin(15, Pin.IN))
```

The piezo buzzer will be attached to pin 9 and will use pulse width modulation. The `buzzer_tones` list contains the pitches for the buzzer to play.

```
buzzer = PWM(Pin(9))
buzzer_tones = [131, 165, 196, 262]
```

Next, the display needs to be attached using an SPI connection.

```
spi = SPI(
```

```
    0,
    baudrate=10000000,
    polarity=1,
    phase=1,
    bits=8,
    firstbit=SPI.MSB,
    sck=Pin(18),
    mosi=Pin(19),
    miso=Pin(16)
)
display = Display(
    spi, dc=Pin(20), cs=Pin(17), rst=Pin(21))
display.clear()
```

Finally, a font to use for any printed text to the display must be added.

```
espresso_dolce = XglcdFont(
    'fonts/EspressoDolce18x24.c', 18, 24)
```

To use the LCD screen and the IR remote, drivers needed to be created for the Pico. The drivers provided in the repository are modified versions of rdagger's micropython-ili9341 library [17] and Peter Hinch's micropython_ir library [9]. The modified drivers would be given to students, and they would not be expected to modify them in any way.

As previously mentioned, the class uses several custom PCBs to facilitate the use of different circuit components. However, no circuit board was designed for this project, so the wiring was done by hand, as seen in Figure 6.1. Only small changes to hole locations and traces would be needed to adapt the current PCBs into a useable form for the Pico.

Since no new assignments are being added to the class, the learning objectives for the class do not change.

6.3 Project Deliverables

In the GitHub repository associated with this paper, which can be found in Appendix A, the folder titled "6-computer-applications-in-me" contains the rewritten lab assignment, code skeleton, and solution for the exercise mentioned above. The folder also contains a README that details what is in each file and what software is needed to complete the assignments.

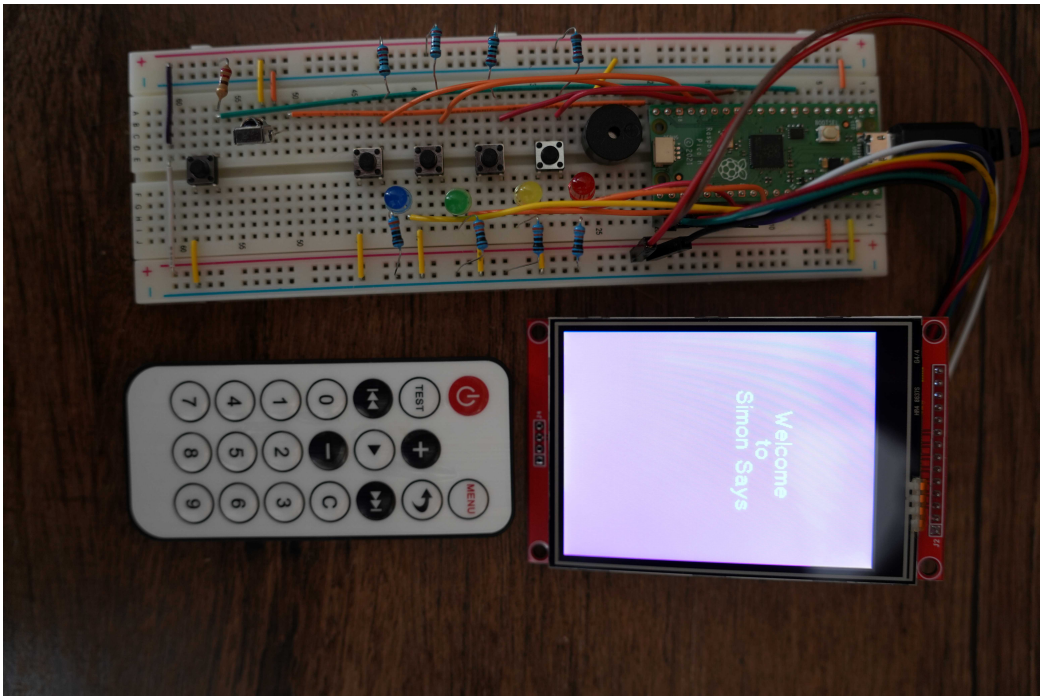


Figure 6.1: Hardware for Simon Says Game

Installation instructions can be found in the "usage-and-installation" folder in the GitHub repository.

The integration of these curriculum changes would not be a simple task. Since the entire class is structured around C++ and the ESP32, every lecture, assignment, quiz, and test would have to be translated. In addition to this, new PCBs would need to be designed for use with the Pico.

Chapter 7

ME 571: Fluid Mechanics

7.1 Course Overview

ME 571: Fluid Mechanics is a lecture based class that focuses on the analysis and kinematics of fluids through the application of the conservation equations, dimensional analysis, and typical flow applications. The course is computation heavy and requires a thorough understanding of the principles at play and the mathematics associated with them. As such, fluid mechanics provides several opportunities to show the value of programming by allowing students to focus on the principles at play rather than the advanced mathematics.

Due to the complexity of many problems in fluid mechanics, many software packages have been created to relieve the computational burden from engineers. A few examples include:

- System of Equation Solvers
  - The most common of which is Engineering Equation Solver (EES), a solver that contains a multitude of built-in functions and properties for thermal and fluid sciences [11].
  - Python and Excel also serve as capable equation solvers, though the experience is not nearly as tailored as EES.
- Computational Fluid Dynamics (CFD)
  - A CFD is a simulation program that uses numerical methods to solve fluid flow problems. They often output graphics, such as

flow streamlines, that allow for a quick and easy analysis of the problem.

– Many popular CFD packages exist, such as ANSYS Fluent or SOLIDWORKS Flow Simulation.

With software solutions being common in both industry and research, introducing students to programmatic solution methods becomes even more important.

7.2 New Assignment

Currently, no assignments in ME 571: Fluid Mechanics make use of programming, but not for the lack of options. The study of fluid mechanics, as with many areas of engineering, requires the liberal application of differentials to solve problems and model systems. When given as assignments, questions need to carefully crafted to ensure a symbolic solution can be found, since finding a numerical solution is difficult and time-consuming. With the addition of programming, however, finding a numerical solution becomes a trivial matter (assuming standard numerical methods are appropriate for the problem).

Python has two libraries that excel at solving differentials, 'sympy' [14] for symbolic solutions and 'scipy' [26] for numerical solutions. The following assignment adapted from *Fundamentals of Fluid Mechanics, 7th Edition* [16], Assignment 7.1 below, shows an example of using both sympy and scipy to solve a question that requires both symbolic and numerical differentials.

ME 571: Assignment 7.1

Problem Statement

As a valve is opened, water flows through the diffuser shown in Fig. P4.31 at an increasing flow rate so that the velocity along the centerline is given by $\mathbf{V} = u\hat{\mathbf{i}} = V_0(1 - \mathrm{e}^{-ct})(1 - x/l)\hat{\mathbf{i}}$, where u, c, and l are constants. Determine the acceleration as a function of x and t. If $V_0 = 10ft/s$ and $l = 5$, what value of c (other than $c = 0$) is needed to make the acceleration 0 for any x at $t = 1s$? Explain how the acceleration can be zero if the flow rate is increasing with time? Be sure to leave specify all assumptions and comment on the functionality of the code.

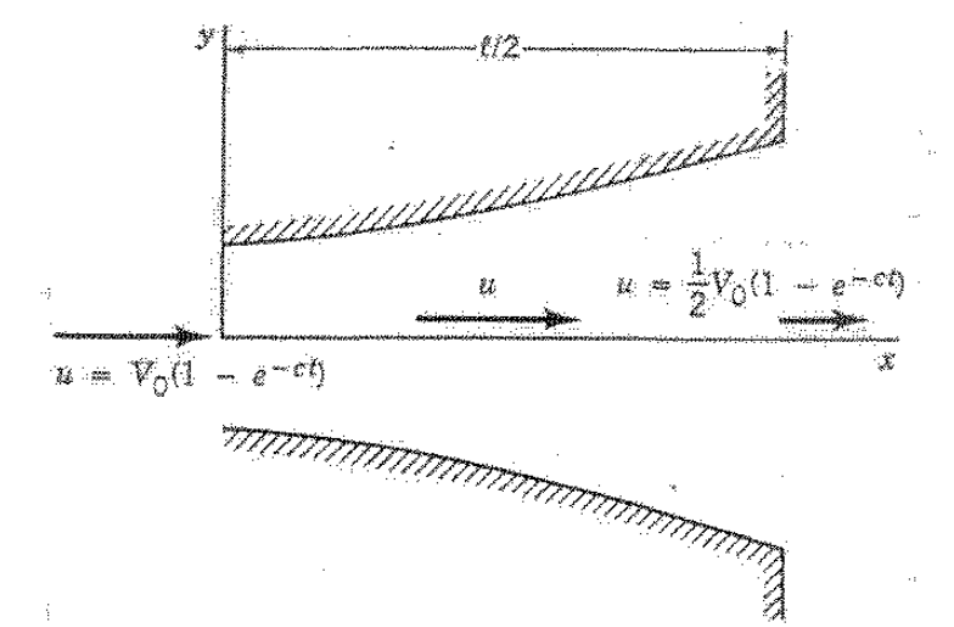


FIGURE P4.31

Problem Solution

For the full solution, see Appendix A. Two solutions are presented in the repository, and the entirety of solution 2 is shown below.

```
from sympy import symbols, lambdify, diff, exp
from scipy.optimize import fsolve
from matplotlib import pyplot as plt
```

After importing the necessary libraries, we can set up the symbols that represent the constants and variables used in the equation. After that, create an equation for the velocity and use the 'diff' function to get the acceleration equation.

```
c, x, t, l, v = symbols("c x t l v")
vel = v*(1 - exp(-c * t))*(1 - x/l)
accel = diff(vel, t) + vel * diff(vel, x)
```

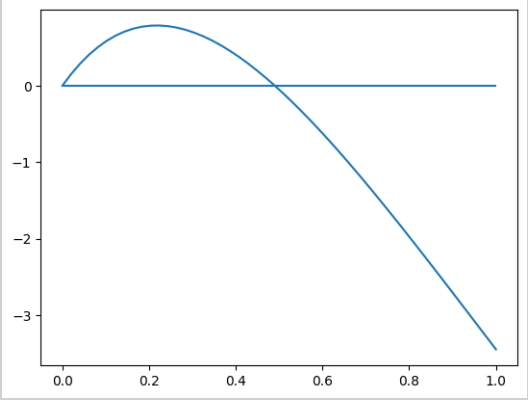
After getting the acceleration equation, we can substitute in the known values and then lambdify the equation, so we can use a numerical solver on it. Since the x-value does not matter for this equation, we can put in any value we want. We will opt for 1.

```
accel = accel.subs({x: 1, t: 1, l: 5, v: 10})
```

```
acceleration = lambdify([c], accel, "scipy")
```

To use the fsolve function, you need to have an initial guess at the value. To get that estimate, graph the acceleration function and observe where it crosses the x-axis.

```
x = linspace(0, 1)
plt.hlines(0, 0, 1)
plt.plot(x, acceleration(x))
```



Inspecting the graph shows an estimate value of 0.5. With that guess in hand, we can use fsolve to find the value.

```
c = fsolve(acceleration, 0.5)
print(f"c = {c[0]:.4f} 1/s")
```

This gives an output of $c = 0.491 \frac{1}{s}$.

While the second solution is shown in Assignment 7.1, the first solution assumes the student solved for the acceleration equation by hand and then used Python to find the numerical solution.

7.3 Project Deliverables

In the GitHub repository associated with this paper, which can be found in Appendix A, the folder titled "7-fluid-mechanics" contains both the problem statement and solution guide for the problem introduced in the previous section. The folder also contains a README that details what is in each file

and what software is needed to complete the assignments.

For these projects to be added to the class, the instructor would simply need to give the skeleton files to students as a problem statement. Given the use of specialized solving methods used in the solution, an in-class example would be all but required.

Chapter 8

ME 533: Machine Design

8.1 Course Overview

ME 533: Machine Design is a lecture based class that focuses on stress/strain analysis, load determination, and failure theories. Despite being the first machine design class in the mechanical engineering curriculum, the material is a continuation of topics covered in two required civil engineering courses: CE 333: Statics and CE 533: Mechanics of Materials.

As a higher level course that relies on well established theories and analysis methods, machine design presents several opportunities for programmatic solutions to problems, though Excel calculators may be more appropriate in some cases. Despite this, machine design does not require programming for assignments or projects, though this fact has varied depending on the current instructor.

Homework assignments typically involve resolving loads on various members, analyzing the internal stresses, and determining the safety of a design using the appropriate failure theories. When appropriate, deformation plots are drawn to highlight points of failure. The class finishes with discussions on spring and fastener design, which involves heavy iteration.

8.2 New Tool for Solving Assignments

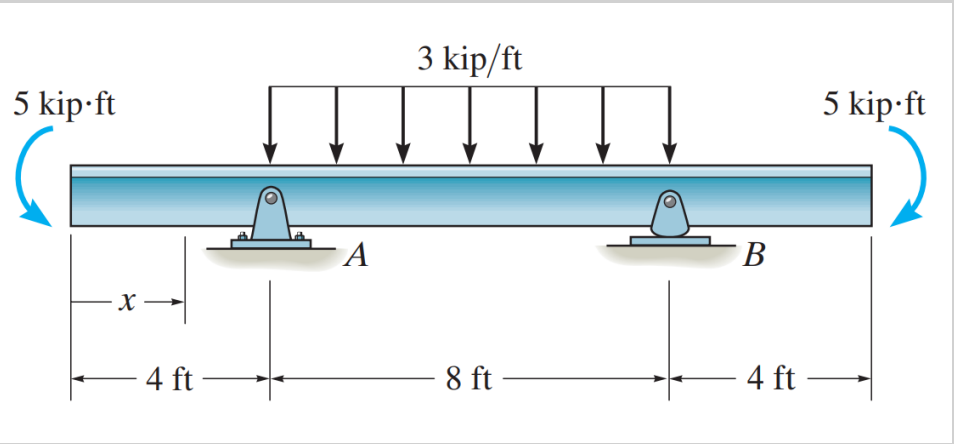
Since no assignments currently make use of programming, a new assignment with two questions adapted from *Mechanics of Materials, 10th Edition* [8] have been solved to demonstrate potential uses for programming in the

course. Rather than writing a Python script to solve iterative design or strictly numerical problems (see chapters on Thermodynamics, Fluid Mechanics, or Heat Transfer for examples of this), Assignment 8.1 focuses on finding and plotting beam deflection using singularity functions.

ME 533: Assignment 8.1

Problem Statement

The following beam is subjected to the load shown. Determine the equation for deformation. The beam is made of aluminum and has an I = 156 in^4. Assume EI is constant. Graph the shear, moment, slope, and deflection.

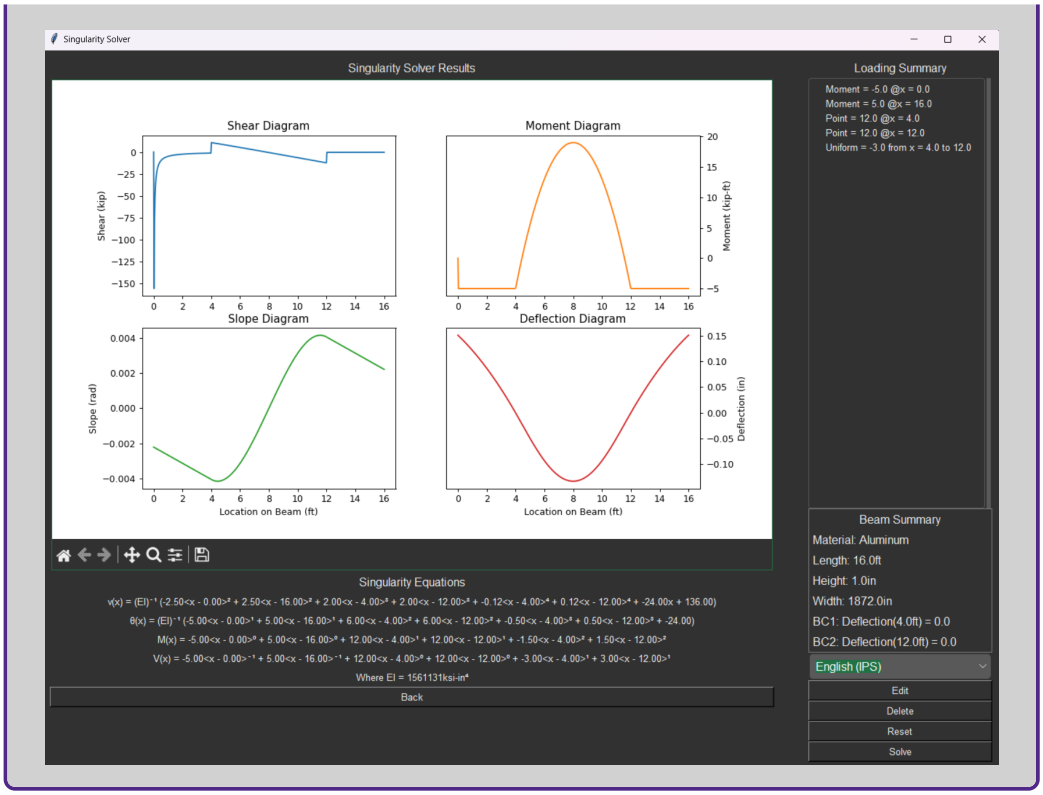


Problem Solution

For the full solution, see Appendix A. The following code shows how to launch the Singularity Solver from a Jupyter Notebook.

```
import subprocess

output = subprocess.run(
    ["python", "-m", "singularity"],
    shell=True,
    capture_output=True,
    cwd="./MNE/machine_design"
)
print(output.stdout.decode("ascii"))
```

After inputting the beam attributes, loading state, and boundary conditions, the following graph is produced.



The most simple adoption of programming into singularity function assignments would be requiring computer generated plots. Students would first solve for reaction forces, create a singularity equation, integrate several times, and then create a plot using the deformation equation. The solution presented here takes a higher level approach by accepting the loading forces on the bar and then creating and integrating the singularity equation automatically. This is done through a graphical user interface developed and executed in Python.

The GUI can be launched in Python (Assignment 8.1) or through the terminal as a Python module. Once launched, the application will open in a new window, which is shown in Figure 8.1. The home page contains instructions for using the solver, so the details here will be sparing. The second page is dedicated to beam configuration, including length, cross-section, material, and boundary conditions. The next page presents various forms of loading to add to the beam. The last page allows custom materials to be added to the list of materials.

While the GUI currently automates significant portions of singularity

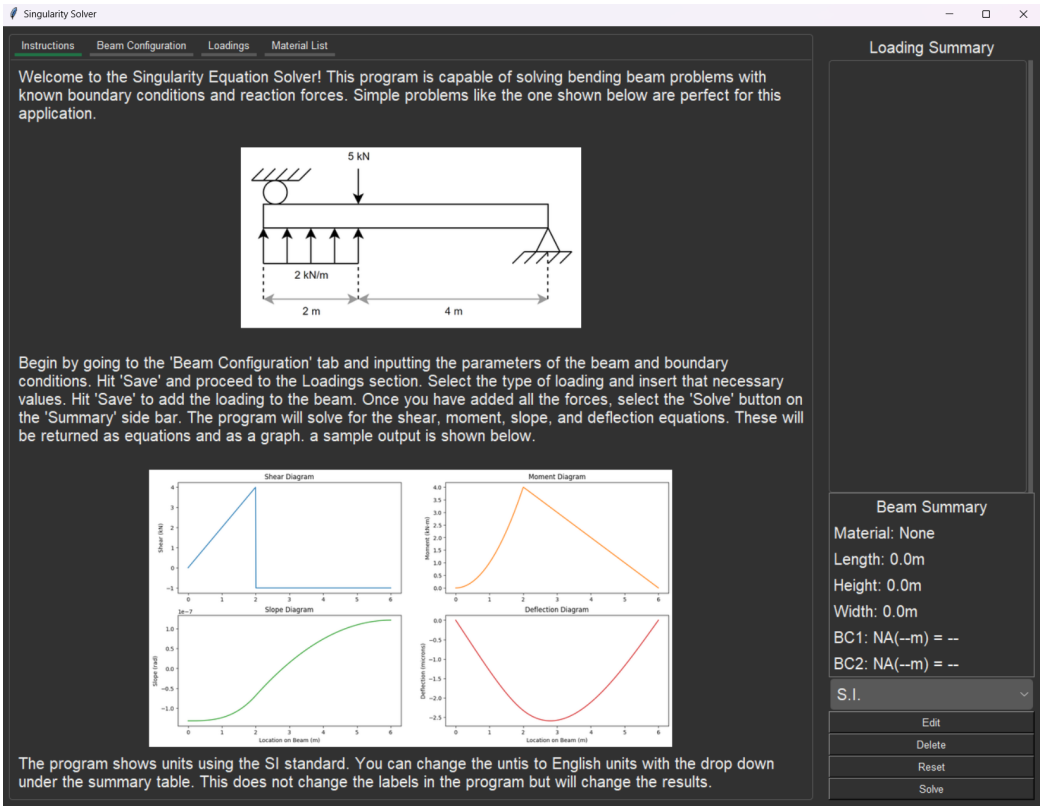


Figure 8.1: Singularity Solver GUI

problems, several features should be added at a later date, including non-rectangular cross-sections (though a workaround is presented) and the ability to add supports to the beam. This would give the program the ability to calculate the reaction forces and determine boundary conditions without relying on the user to correctly find and input them.

Since this application takes most of the work out of the hands of the students, it would not be a good tool for teaching singularity functions. Rather, the application would be best utilized when finding the shear, moment, slope, or deformation of a beam is one part in the context of a larger problem. Additionally, it would serve as a realistic look at how most analysis is done in industry - by a computer.

Using programming in this manner gives students a better idea of how real problems are solved by introducing them to a more efficient and powerful solution method. These questions would also directly correlate to Abet

Student Outcomes 1, 6, and 7, as seen in Appendix C and weekly correlates to Outcome 3.

8.3 Project Deliverables

In the GitHub repository associated with this paper, which can be found in Appendix A, the folder titled "8-machine-design" contains both the problem statements and solution guides for the two questions introduced in the previous section. The folder also contains a README that details what is in each file and what software is needed to complete the assignments.

For these projects to be added to the class, the instructor would need to give the skeleton files to students as a problem statement and the MNE folder as Python library. Using one of the two questions as an in-class example would serve as a good opportunity to remind students how to use Python with Jupyter Notebooks and to demonstrate how to open and use the singularity function GUI.

Chapter 9

ME 570: Control of Mechanical Systems I

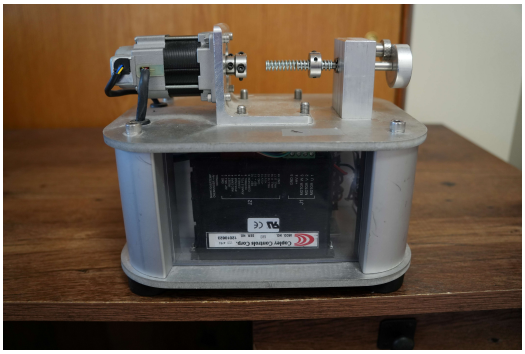
9.1 Course Overview

ME 570: Control of Mechanical Systems is a mixed class with two lectures and one lab every week. The lectures focus on teaching the principles and theory of controls while the lab focuses on designing and testing control systems. Most labs are dedicated to using and designing PID controllers using different methods, such as frequency analysis or pole analysis.

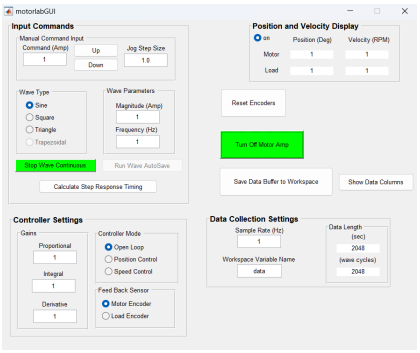
To complete the labs, the class makes use of a custom motor rig called a motorlab, show in Figure 9.1. To go along with the motorlab, a graphical user interface for controlling the motorlab was also created by Dale Schinstock, also shown in Figure 9.1. This GUI runs as an application through MATLAB and allows the students to both send commands to the motor and collect runtime data from the motor.

Controls also relies heavily on MATLAB for the completion of lab and homework assignments. All work is done using .mlx files through a licensed copy of MATLAB for a more interactive working environment. To access the controls related commands, a package called the Controls Toolbox has to be purchased from MATLAB.

While Controls uses programming more than any class besides ME 400, little to no instruction is provided on how to program in MATLAB. The first lab is dedicated to taking the MATLAB Onramp starter course, which takes about 1 to 2 hours to complete. However, the Onramp does not address



(a) The motorlab apparatus



(b) The motorlabGUI interface

Figure 9.1: The motorlab system designed by Dale Schinstock

most of the work done in Controls, such as plotting and transfer functions. The Onramp does teach students the basics of MATLAB syntax, but most students come out of the Onramp just as confused by MATLAB as they were going in.

This disconnect between students and MATLAB becomes a barrier that prevents students from understanding controls. Many students spend more time trying to understand MATLAB than they do learning controls.

9.2 Lab Assignment Redesigns

Since ME 570 already fully utilizes programming in the course, no new assignments are being created. Instead, three pre-existing labs have been translated from MATLAB to Python. While MATLAB is the industry standard when it comes to control systems, the license is expensive, students do not have a solid understanding of it, and we do not make use of its most powerful feature, Simulink. In addition to this, Python has a library, aptly named "control," which has a MATLAB module that imitates the Control Toolbox, both in form and functionality, from MATLAB. In combination with Jupyter Notebooks and a few custom plotting modifications, using Python will give a very similar experience to MATLAB, and will give students that pursue the field of controls any easy transition to MATLAB.

All three translated labs make use of the motorlab, which has had its GUI turned into an application, courtesy of the MATLAB Compiler. This allows the application to be run from an app icon or from the terminal. These labs

also make use of a custom plot command. The plot command hijacks the standard matplotlib [10] plot command and adds the ability to create data tips. The direction of the data tip changes based on which mouse click is used, and a double click in the plot window will remove all data tips. More custom functions could be added later to imitate MATLAB functionality as needed, but this was the only one required for completing these labs.

The first translated lab, Lab 4, is a standard controls assignment that makes use of the motorlab, reading the csv output from the motorlab, and plotting the results [20].

ME 570: Assignment 9.1

Problem Statement

For the full problem statement, please see Appendix A. Estimate the damping ratio by fitting an envelope to the step response.

Problem Solution

For the full solution, see Appendix A. The solution shown below highlights the use of data from the motorlabGUI and plotting with custom data tips.

To achieve an environment similar to MATLAB, the following imports are required at the top of the file.

```
%matplotlib widget
from matplotlib.pyplot import figure, xlabel, ylabel,
                               legend, show, title, xlim,
                               ylim
from custom_functions import plot
from control.matlab import tf, step
from numpy import polyfit, array
from math import pi, sqrt
from pandas import read_csv
```

```
J = 1.1e-5 + 0.19e-5
ks = k_estimate
Kdc = kt/ks
wn = sqrt(ks / J)
zeta_estimate = 0.05

# Generate a step response from a first order system
# with a pole equal to the real part of the
# 2nd order poles
real_part = zeta_estimate * wn
envelope_TF = tf(Kdc * real_part, [1, real_part])
```

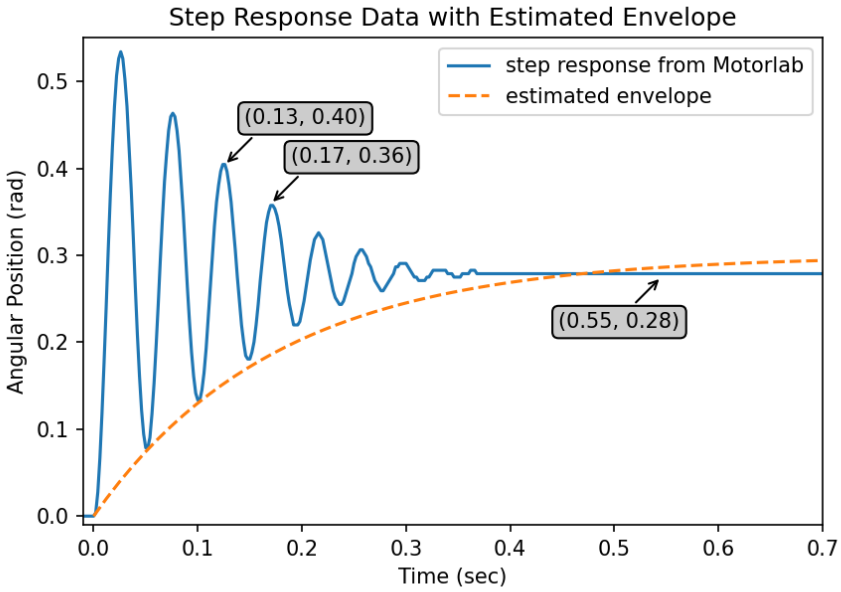
```
[envelope_y, envelope_time] = step(envelope_TF)
```

The following code assumes that the motorlabGUI data was collected and saved with the name "stepdata.csv." Here, 'read_csv' and 'iloc' from the 'pandas' library are used for clean file reading.

```
# Extract data from the generated step response
stepdata = read_csv("stepdata.csv", header=None)

# extract time column of the data matrix
dataTime = stepdata.iloc[:,0]

# extract third angle column of the data matrix
dataAngle = stepdata.iloc[:,2]

# convert to rad
dataTh = kdr * dataAngle
```

After the data has been loaded into the workbook, a graph of the step response can be plotted.

```
plot(dataTime, dataTh, envelope_time, envelope_y, '--')
ylabel('Angular Position (rad)')
xlabel('Time (sec)')
ylim(-0.01, 0.55)
xlim(-0.01, 0.7)
title('Step Response Data with Estimated Envelope')
legend(['step response from Motorlab','estimated
                              envelope'])
show()
```



The second lab, Lab 10 [18], builds on Lab 4 by adding root locus plots and sisotool. While the sisotool command in Python is not as powerful as the full designer in MATLAB, it does allow for interactive plots where moving poles updates the graphs automatically.

ME 570: Assignment 9.2

Problem Statement

For the full problem statement, please see Appendix A. Create a controller for the motorlab and plot the poles using sisotool.

Problem Solution

For the full solution, see Appendix A. Below is an example of setting up a transfer function and a controller and then using sisotool to analyze the system.

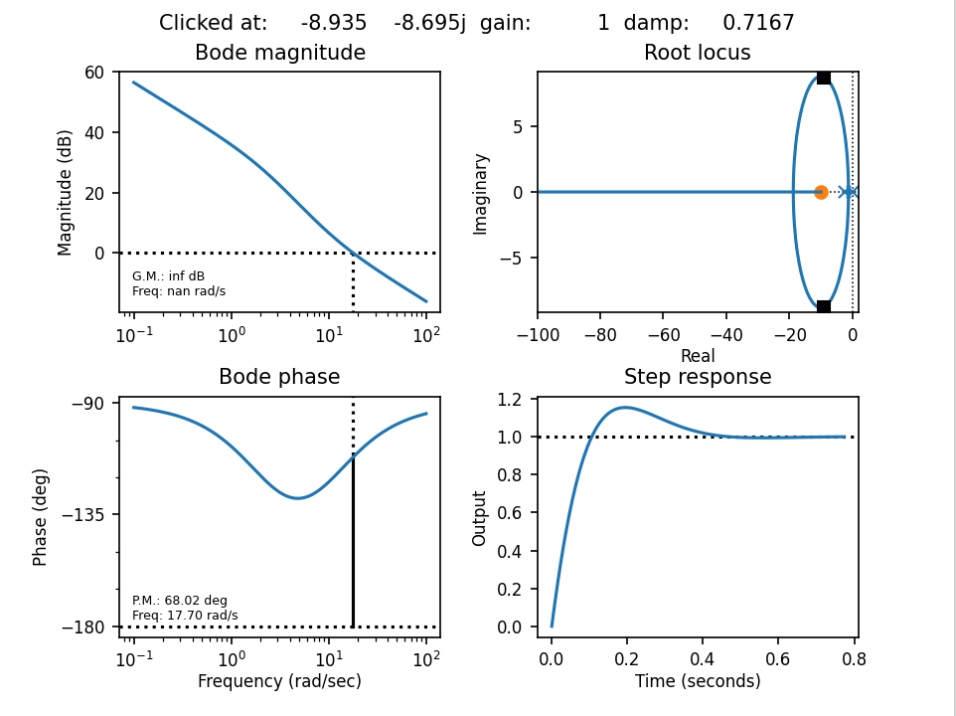
```
b = 3e-5
kdr = 180/pi
J = 1.29e-5
kt = 0.05
```

```
s = tf('s')
Gm= kt*kdr / (J*s**2+b*s)
Gc2 = 10*.00007 + .00007*s
```

After the transfer function and controller have been made, sisotool can be used to adjust the poles.

```
figure()
sisotool(Gc2*Gm)
show()
```

Grabbing and moving the black squares will allow the user to move the pole locations.



The third lab, Lab 13, focuses on frequency response design methods and utilizes Bode plots [19].

ME 570: Assignment 9.3

Problem Statement

For the full problem statement, please see Appendix A. Given the natural fre-

quency and dc gain derived from the table (not shown here), construct a new transfer function, that will hopefully better match the data.

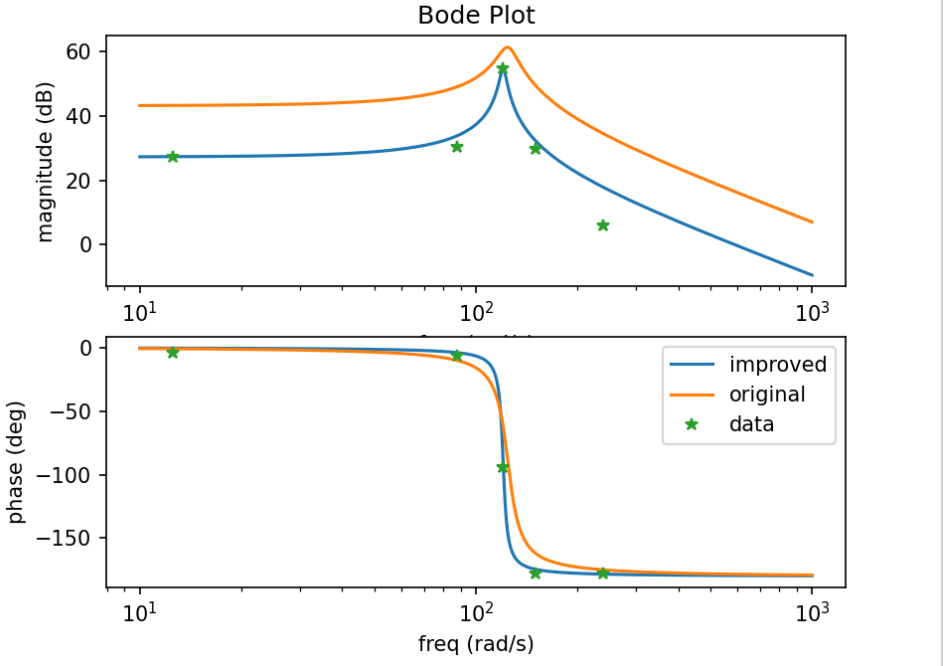
Problem Solution

For the full solution, see Appendix A. The followign code is an example of using Bode in Python to do frequency analysis.

```
wn = 19.18*2*pi # natural freq from data (rad/s)
Kdc = 23 # dc gain from data in deg/Amp
Mwn = 540 # magnitude ratio at wn from data in deg/Amp
zeta = Kdc/Mwn/2 # calc damp ratio using Mwn and Kdc
Gnew = (Kdc*wn**2) / (s**2 + 2*zeta*wn*s + wn**2)

mnew, pnew, wnew = bode(Gnew) # get mag, phase, freq
mnew = 20 * log10(mnew)
pnew = (180 / pi) * pnew # Convert radians to degrees
```

With the new data models complete, graph them using a logarithmic plot.

```
figure()
subplot(2, 1, 1)
semilogx(wnew, mnew, w, m, wdata, magdata, '*')
title('Bode Plot')
ylabel('magnitude (dB)')
xlabel('freq (rad/s)')

subplot(2, 1, 2);
semilogx(wnew, pnew, w, p, wdata, phdata, '*')
ylabel('phase (deg)');
xlabel('freq (rad/s)')
legend(['improved', 'original', 'data'])
show()
```



The labs chosen aim to showcase how the major features used in MATLAB can be emulated in Python. Some features, like Simulink, have no direct correlation. However, Simulink is only used in one lab, and students only need to make two small changes. Python also requires more library imports, but these can all be handled by the skeleton and do not pose much concern.

Since no new assignments are being added to the class, the learning objectives for the class do not change.

9.3 Project Deliverables

In the GitHub repository associated with this paper, which can be found in Appendix A, the folder titled "9-control-of-mechanical-systems" contains the lab assignments, code skeletons, and solutions for the three lab assignments explained above. The folder also contains a README that details what is in each file and what software is needed to complete the assignments.

The act of integrating these labs into the class would not be as simple as the integrations for other classes. Since every lab uses MATLAB, the

other 11 labs would also need to be translated. In addition to this, every computer in the lab would need to get the correct applications, extensions, and motorlabGUI executable installed. Instructions for installing everything needed can be found in the "usage-and-installation" folder in the GitHub repository.

Chapter 10

ME 573: Heat Transfer

10.1 Course Overview

ME 573: Heat Transfer is lecture based class that focuses on analyzing systems using the three modes of heat transfer: conduction, convection, and radiation. The study of heat transfer is dense with equations, analysis methods, and tables, most of which provide opportunities for programming to be used for solving problems, whether that be Excel calculators or Python. Of these, perhaps the most practical application is the design of heat exchangers. Accordingly, the only project in the class is an iterative heat exchanger design problem, which requires the use of Excel or Python.

10.2 Project Modifications

Since a project that makes use of programming already exists for Heat Transfer, no changes have been made to the assignment. The problem statement comes from *Fundamentals of Heat and Mass Transfer, 7th Edition* [1] and can be found unchanged in the 10-heat-transfer repository along with a solution made using Python in a Jupyter Notebook.

The project assignment, shown in Assignment 10.1, deals with the design of a shell and tube heat exchanger. The solution to the problem requires table data, iteration, and plotting, making it a prime example of the practical benefits of programming in the field. A library that fully implements the IF-97 standard called 'pyXSteam' is used for steam and water properties [4].

ME 573: Assignment 10.1

Problem Statement

A heat exchanger manufacturing firm has hired your team to design a shell and tube heat exchanger. The heat exchanger is expected to heat 10,000 $\frac{kg}{h}$ of water flowing through the tubes from 16 to 84 °C. The hot engine oil flowing through the shell is used for heating the water. The oil makes a single shell pass, entering at 160 °C and leaving at 94 °C, with an average overall heat transfer coefficient $U_h = 400\frac{W}{m^2K}$. The water flows through 11 brass tubes (k = 137 $\frac{W}{mK}$) of 22.9 mm inside diameter and 25.4 mm outside diameter, with each tube making four passes through the shell, as shown below.

Declare known variables, make necessary imports, and interpolate for property values.

Problem Solution

For the full solution, see Appendix A. The following code shows how to access and use the steam tables using pyXSteam.

```
from pyXSteam.XSteam import XSteam

# m/kg/sec/K/MPa/W
steam_table = XSteam(XSteam.UNIT_SYSTEM_BARE)

v = steam_table.vL_t(323) # m^3/kg
Cpc = 1000*steam_table.CpL_t(323) # J/(kg*K)
mew = steam_table.my_pt(0.1, 323) # N*s/m^2
kwater = steam_table.tcL_t(323) # W/(m*K)
```

Unfortunately, no Python library with engine oil data currently exists, so values will have to come from a textbook. The following function was made using scipy.optimize.curve_fit and a fourth order polynomial with the data from Appendix A: Table A.5 from *Fundamentals of Heat and Mass Transfer* by Bergman, Lavine, Incropera, and Dewitt.

```
def oil_specific_heat(x):
    a,b,c,d,e = 7.57640461e-07, -1.16948468e-03,  6.
                                76308996e-01, -1.
                                69259798e+02, 1.
                                72835141e+04
    return (a*x**4+b*x**3+c*x**2+d*x+e)

Cph = oil_specific_heat(400) # J/(kg*K)
```

Since no new assignments or requirements are being added, the learning objectives for the class do not change.

10.3 Project Deliverables

In the GitHub repository associated with this paper, which can be found in Appendix A, the folder titled "10-heat-trasnfer" contains both the problem statement and solution guide for the project introduced in the previous section. The folder also contains a README that details what is in each file and what software is needed to complete the assignments.

For these projects to be added to the class, the instructor would simply need to give the skeleton files to students as a problem statement. Giving an example of how to set the units and pull data from the pyXSteam library would be a useful addition to the in-class presentation of the project.

Chapter 11

Discussion

11.1 Issues and Concerns

While the previous chapters proved that Python can be effectively used to solve problems in the mechanical engineering field, it did little to address the issues that would come with the adoption of a unified programming ecosystem, both as it relates to the concept itself as well as the limitations of the proposed solution.

11.1.1 Version Control

While open source software is often viewed as desirable thanks to the no-cost price tag and active development that it promotes, it comes with a downside often avoided by premium software: version control. Python releases a new minor version every year and regularly releases patches for supported versions. On top of that, several of the libraries and extensions used in this project are written and controlled by various groups and individuals that make no guarantee of longevity or future support. This leaves a substantial burden on the department to moderate and maintain a locally hosted version of Python and the libraries required by each class, as well as Visual Studio Code and its extensions. As the number of classes utilizing programming increases, so does the difficulty of validating a functioning stack.

With that said, the majority of the software being used is well maintained and rarely makes breaking changes. Libraries like numpy, matplotlib, and scipy are staples of the Python ecosystem. Libraries like mendeleev or

pyXSteam that rarely see updates and rely on older versions of more popular libraries present a larger concern.

Likewise, updates to Visual Studio Code and its extensions often amount to bug fixes or small feature changes and pose little threat to the functionality of the development environment as it relates to the mechanical department.

11.1.2 Industry Standards

By adopting Python as the only language taught in core curriculum classes, students would lose exposure to C/C++, the standard for embedded software design, and MATLAB, the standard for control system design. While these languages would hopefully still be taught through electives such as ME 615: Applications in Mechatronics and ME 640: Control of Mechanical Systems II, the general student body may never learn either language.

Both C/C++ and MATLAB are deeply rooted as the standards in their fields, and for good reason. C/C++ is a compiled language that is designed to access lower levels of hardware than Python (though MicroPython does enable low-level access, it is still an interpreted language and can never match the speed of compiled code), and MATLAB has decades of controls development in the form of Simulink, the true standard for control system design.

With that being said, Python is the industry standard for data analysis and numerical methods thanks to the powerful libraries and integration with Jupyter Notebooks, which provide a smooth workflow.

With that being said, Python is the industry standard for data analysis and numerical methods, and these are the applications that many mechanical engineers are interested in. While plenty of mechanical engineers work with embedded devices or design control systems, these jobs are transitioning more towards the Electrical and Computer Engineering field. The typical mechanical engineers will be interested in analyzing data produced by a machine or with running preliminary calculations before committing the time and resources to a lengthy simulation or prototype. These are the applications that will see the most benefit from learning Python.

11.1.3 Opportunity Cost

With the addition of programming to a class, most classes would need to devote at least a portion of one lecture to showcasing the libraries needed to solve homework problems. For classes before ME 400, an entire lecture

may be necessary. This would fit best alongside a typical by-hand, in-class example to compare the two methods.

The addition of programming would also lead to additional questions for instructors regarding the setup and installation of the environment. While the setup steps are relatively easy compared to other languages, it will still be unfamiliar to students and require teachers to be knowledgeable on the subject.

11.2 Recommendations

I recommend classes that make significant use of any property tables, iterative design, or graphing add at least a single assignment that requires the use of Python and the relevant library to complete an assignment typical of the subject field. As it relates to this project, that would include DEN 161: Engineering Problem Solving, ME 513: Thermodynamics, NE 495: Elements of Nuclear Engineering, ME 571: Fluid Mechanics, ME 533: Machine Design, and ME 573: Heat Transfer. The addition of programming to these classes will increase students' abilities to efficiently solve problems with little downside.

Given that this paper was not afforded the time to implement these changes and observe the change in students, further recommendations are hinged on the result of the first recommendation.

Should students continue to struggle with programming, especially in classes prior to ME 400, I recommend that a programming class either be added or moved into the first four semesters of the degree, preferably the first three. The nature of this change could take any of several forms.

The easiest change would involve moving ME 400 forward two semesters in the flowchart and moving CHE 354/355 and IMSE 250 back two semesters. Currently, ME 400 does not utilize information from MATH 340 and neither CHE 354/355 nor IMSE 250 are prerequisites for other classes, making this change low friction.

Another possible change is the addition of a programming class, like CIS 209: Computer Programming for Engineers, into the curriculum. In addition to giving students a better foundation in programming through the introduction of an extra class, this change would also free up time in ME 400.

Finally, a more drastic change would be a redesign of the ME 400 and ECE

519: Circuits and Controls/ME 519: Circuits for MNE classes. Rather than acting as a microcontroller class, ME 400 can transition into a dedicated programming class that spends time teaching the basics of programming, how to use external libraries, and how to solve basic problems using Jupyter Notebooks. Then, ECE/ME 519 could be transitioned to a circuit and microcontroller class that lectures on circuit theory and the fundamentals of microcontrollers while utilizing labs that give practical experience building circuits and writing embedded code. The scope of this class would reduce the amount of time spent on both circuit theory and microcontrollers, which may not be an acceptable trade-off. With that being said, the majority of the information taught in ME 519 is either repeated in ME 535: Measurements and Instrumentation or is not used again in the curriculum.

If, however, the students continue to struggle after ME 400, I would recommend converting ME 400 and ME 570 into Python-based classes to give students a more consistent approach to programming as well as moving ME 400 earlier in the curriculum to solidify the foundations before requiring its use in other mechanical classes.

Chapter 12

Conclusion

12.1 Future Work

While this project shows the possibility of uniting programming in the mechanical engineering curriculum, several additional areas need to be explored before a wide-spread adoption.

The first is a study to substantiate the assumed claim that a more consistent approach to programming would improve students' abilities to solve engineering problems. Additionally, alternative programming languages should be reviewed, such as the use of C++, MATLAB, Octave, or Scilab. Finally, continued work on the creation, documentation, and distribution of an MNE library that contains 3rd party libraries, equations, and tables will solidify the cohesive nature of the programming curriculum and create an easier method for version controlling the software downloaded by students.

12.2 Conclusion

The work in this project demonstrates that a cohesive and consistent programming ecosystem could be adopted throughout the curriculum of the Mechanical and Nuclear Engineering department at Kansas State. Whether by an adapted or translated assignment, each relevant class was able to utilize Python as a problem-solving tool.

This work showcases both the simplicity of setup and ease of use for students while demonstrating the capabilities and convenience of Python thanks to the emphasis on code readability and plethora of engineering related li-

braries. In cases of translation, Python was able to closely imitate the current language, both in for embedded programming and controls, while maintaining the same working environment and syntax. By introducing a unified programming curriculum to the core Kansas State Mechanical Engineering classes, students will be better equipped to solve real engineering problems.

Bibliography

[1] Theodore L. Bergman, Adrienne S. Lavine, Frank P. Incropera, and David P. DeWitt. *Fundamentals of Heat and Mass Transfer, 7th Edition.* Wiley and Sons, 2011.

[2] Ronald Brockhoff. Exercise #7: Simon says. Lab Assignment, 2023.

[3] Bailey Brown. Arduino uno stoplight activity. Class Assignment, 2020.

[4] drunsinn and Magnus Holmgren. pyxsteam. `https://github.com/drunsinn/pyXSteam`, 2020.

[5] Sawyer Fuller, Ben Greiner, Jason Moore, Richard Murray, René van Paassen, and Rory Yorke. The Python Control Systems Library (python-control). In *60th IEEE Conference on Decision and Control (CDC)*, pages 4875–4881. IEEE, 2021.

[6] Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585:357–362, 2020.

[7] Guillermo Hernández. physdata. `https://github.com/Dih5/physdata`, 2016.

[8] Russel C. Hibbeler. *Mechanics of Materials, 10th Edition.* Pearson, 2016.

[9] Peter Hinch. micropython_ir. `https://github.com/peterhinch/micropython_ir`, 2020.

[10] John D. Hunter. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[11] S. A. Klein. Ees–engineering equation solver. `https://fchartsoftware.com/`, 1975.

[12] Chris Martin, Joe Ranalli, and Jacob Moore. chmarti1/pyromat: Version 2.2.4, October 2022.

[13] Łukasz Mentel. mendeleev - A Python package with properties of chemical elements, ions, isotopes and methods to manipulate and visualize periodic table., March 2021.

[14] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, Amit Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Fransesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrmam, and Anthony Scopatz. SymPy: symbolic computing in Python. *PeerJ Computer Science*, 3, January 2017.

[15] Michael J. Moran, Howard N. Shapiro, Daisie D. Boettner, and Margaret B. Bailey. *Fundamentals of Engineering Thermodynamics, 9th Edition*. Wiley and Sons, 2018.

[16] Bruce R. Munson, Alric P. Rothmayer, Theodore H. Okiishi, and Wade W. Huebsch. *Fundamentals of Fluid Mechanics, 7th Edition*. Wiley and Sons, 2012.

[17] rdagger. micropython-ili9341. `https://github.com/rdagger/micropython-ili9341`, 2020.

[18] Dale Schinstock and Constance Lare. Laboratory #10. Lab Assignment.

[19] Dale Schinstock and Constance Lare. Laboratory #13. Lab Assignment.

[20] Dale Schinstock and Constance Lare. Laboratory #4. Lab Assignment.

[21] Steven Silvester. ipympl. https://github.com/matplotlib/ipympl, 2017.

[22] Steven Silvester. ipykernel. https://github.com/ipython/ipykernel, 2021.

[23] The pandas development team. pandas-dev/pandas: Pandas.

[24] Kansas State University. K-state 8 curriculum. https://www.mne.k-state.edu/student-success/advising/k-state-8/, 2022.

[25] Kansas State University. Program accreditation and information. https://www.mne.k-state.edu/academics/accreditation/, 2023.

[26] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

Appendix A

Project Repository

Full project documentation can be found at the following GitHub repository:

`https://github.com/mKiloLA/python-based-mne/tree/develop`

Implementing a Cohesive Programming Ecosystem in Mechanical Engineering

This repository serves as the project base for a thesis entitled *Implementing a Cohesive Programming Ecosystem in Mechanical Engineering*. Of the ten folders, eight are dedicated to different courses in the Mechanical Engineering curriculum at Kansas State University. Each class folder contains the resources needed to give and evaluate various engineering problems related to the subject. As the title suggests, the primary goal of the project is to promote the use of a single programming language and environment to give students a stronger foundation in programming. As such, each assignment is solved using Python in Visual Studio Code to give a consistent learning environment. For a more detailed look into the reasoning for the project and background on the classes, see the associated thesis.

Folders in the Repository

The follow directory only lists the top level folders. For a breakdown of each project, please see the README files in each project folder.

- DEN 161: Engineering Problem Solving: Assignment information for an Intro to Engineering class.

- ME 513: Thermodynamics: Assignment information for a Thermodynamics class.
- NE 495: Elements of Nuclear Engineering: Assignment information for a Nuclear Engineering class.
- ME 400: Computer Applications in Mechanical Engineering: Assignment information for a Computer Applications class.
- ME 571: Fluid Mechanics: Assignment information for a Fluid Mechanics class.
- ME 533: Machine Design: Assignment information for a Machine Design class.
- ME 570: Control of Mechanical Systems: Assignment information for a Control Systems class.
- ME 573: Heat Transfer: Assignment information for a Heat Transfer class.
- Thesis: Source files for the Thesis associated with this repository.
- Usage and Installation: Instructions for installing and setting up the environment needed to complete the projects in this repository.

Usage and Installation

Since this repository a represents a collection of projects, the required software installation and usage varies by class. Each class has a 'README.md' file that details the software, Visual Studio Code extensions, and Python packages required to run complete the assignments. In general, each project requires a download of Python, Visual Studio Code, and a handful of extensions for Visual Studio Code. The usage-and-installation contains guides for installing Python, Visual Studio Code, and the extensions. The Python installation guide also contains information on how to install the Python packages used across the entire project. Alternatively, the dependencies can be installed from the requirements.txt file by running the follow command from the root directory of 'usage-and-installation':

```
pip install -r requirements.txt
```

For additional information, see the 'README.md' file in usage-and-installation.

Appendix B

Software Versions

Through the entirety of this paper, the following application, package, and extension versions are used:

- Applications
  - Python 3.12.0
    - Optionally, Python 3.11.5 through Anaconda 2023.09
  - Visual Studio Code 1.85
- Python Packages
  - control v0.9.4
  - ipykernel v6.27.1
  - ipympl v0.9.3
  - matplotlib v3.8.2
  - mendeleev v0.15.0
  - numpy v1.26.2
  - pandas v2.1.4
  - physdata v0.2.0
  - PYroMat v2.2.4
  - pyXSteam v0.4.9
  - scipy v1.11.4

  - sympy v1.12

- Visual Studio Code Extensions
  - Python v2023.22.1
  - Pylance v2023.12.1
  - MicroPico v3.5.0
  - Jupyter v2023.11.1003402403
  - Jupyter Keymap v1.1.2
  - Jupyter Notebook Renderers v1.0.17
  - Jupyter Cell Tags v0.1.8
  - Jupyter Slide Show v0.1.5
  - IntelliCode v1.2.30
  - IntelliCode API Usage Examples v0.2.8
  - vscode-pdf v1.2.2
  - Excel Viewer v4.2.58

Appendix C

Abet Student Outcomes

The following excerpt is taken directly from K-State's website [25]:

Student outcomes describe what students are expected to know and be able to do by the time of graduation. These relate to the knowledge, skills and behaviors that students acquire as they progress through the program. The mechanical engineering program will enable students to attain the following, by the time of graduation:

1. an ability to identify, formulate, and solve complex engineering problems by applying principles of engineering, science, and mathematics

2. an ability to apply engineering design to produce solutions that meet specified needs with consideration of public health, safety, and welfare, as well as global, cultural, social, environmental, and economic factors

3. an ability to communicate effectively with a range of audiences

4. an ability to recognize ethical and professional responsibilities in engineering situations and make informed judgments, which must consider the impact of engineering solutions in global, economic, environmental, and societal contexts

5. an ability to function effectively on a team whose members together provide leadership, create a collaborative and inclusive environment, establish goals, plan tasks, and meet objectives

6. an ability to develop and conduct appropriate experimentation, analyze and interpret data, and use engineering judgment to draw conclusions

7. an ability to acquire and apply new knowledge as needed, using appropriate learning strategies.

https://www.mne.k-state.edu/academics/accreditation/