PGE 310 Fall 2021 Homework 2

September 14, 2021

Welcome to your first assignment using Python!

This homework is split up into two problems. In Problem 1, you will practice coding scalar, vector, and matrix operations by translating Homework 1 into Python. Next, Problem 2 will help you practice some graphing and visualization using Matplotlib.

Please submit this assignment as a Python file (.py) on Github classroom. For more comprehensive instructions on how to test and submit your code, refer to the posted instructions document on Canvas.

And as always, feel free to contact Dr. P, Alex, Bernie, or Ziming with problems, questions, or comments. Learning to code can be difficult (and rewarding!), and we're here to help you overcome the learning curve!

A Quick Review of Functions

Before we get started, let's review a bit about functions. Functions can be used in Python (or any other programming language) to quickly and efficiently rerun/reuse code without having to copy and paste it elsewhere. Using functions will help you keep your code organized, readable, and easily debuggable. Here are some key points to remember:

- In Python, functions are declared with the keyword `def` which is then followed by the function name and the input arguments in parentheses. For example, in the cell below, the function `example_function` is declared with input arguments `num1` and `num2`.
- Everything you want to compute inside a function should be indented, as Python looks for indentation (specifically in increments of 4 spaces) to figure out what is inside and outside a function.
- Any variables you want to save from a function should be placed inside a `return` statement at the end of the function. This is because variables declared or changed inside functions are local variables - or in other words variables inside a function can only be accessed from within the function, even if they are named the same as a variable outside the function.
- Finally, it is important to know that functions can only be called **after** they have been declared. For example, the following will **NOT** work because the function is called before it is defined:

```
# The following code will give an error!
first_number = 1
second_number = 4
answer = example_function(first_number, second_number)
```

```
print(answer)

def example_function(num1, num2):
    sum = num1 + num2
    return sum
```

Next is the corrected example…

Also note how the function is declared with input arguments `a` and `b` but the variables `first_number` and `second_number` are passed when calling the function. When you call a function, you can pass in different variable names from what is declared in the function. The important thing is that the data types (number, string, list, etc.) of what is declared and what is passed match with what is computed within function. And please keep in mind that it is good practice to use variable names that are desciptive of what they represent. Abbreviations and short hand are good too if they are easily recognizable or if you comment what they mean!

[ ]:
```
first_number = 1
second_number = 4

def example_function(num1, num2):
    sum = num1 + num2
    return sum

answer = example_function(first_number, second_number)
print(answer)
```

Although functions might seem intimidating right now, keep in mind that you have already seen a number of different functions! Python has a number of built-in functions that we've covered in class, including `print()` and `len()`. We have also used several functions from the NumPy library, such as `numpy.eye()` and `numpy.matmul()`. Check out the NumPy documentation for these functions; take note of what they need as input arguments and what they return.

Going forward, we will use functions to automatically give you feedback and grade your homework on GitHub Classroom. The system is able to work automatically by finding the functions we provided and testing them in a process called "unit testing". **Please do not change the function names because the unit tests will look for the function names provided. If you change the function names, the automated grading system will not work.**

Import the required packages and modules

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate
```

Problem 1: Scalar, Vector, and Matrix Operations

In the following problem, you will translate Homework 1 into Python. To do so, first recreate the scalars, vectors, and matrices from Homework 1 in the following cell. Then, in the proceeding cells,

please compute the questions from Homework 1.

[ ]:
```
# Fill in the scalars and vectors.
# a and b are short for alpha and beta.
a =  # alpha
b =  # beta

# u, v, w are 1-D numpy array
u =
v =
w =

# x, A, B are 2-D numpy array
x =
A =
B =
```

[ ]:
```
# 1 (a)
def problem_1a(w, u):
    # type in the expression after the 'result =' in the next line
    result =
    return result

answer_1a = problem_1a(w, u)
print(answer_1a)
```

[ ]:
```
# 1 (b)
def problem_1b(v, w):
    # type in the expression after the 'result =' in the next line
    result =
    return result

answer_1b = problem_1b(v, w)
print(answer_1b)
```

[ ]:
```
# 1 (c)
def problem_1c(v, w):
    # type in the expression after the 'result =' in the next line
    result_in_radians =
    result_in_degree =
    return result_in_radians, result_in_degree

answer_1c = problem_1c(v,w)
print(answer_1c)
```

[ ]:
```
# 1 (d)
def problem_1d(u, v, w, a, b):
```

```
    u = u.reshape((1,4))
    v = v.reshape((1,4))
    w = w.reshape((1,4))

    # type in the expression after the 'result =' in the next line
    result =
    return(result)

answer_1d = problem_1d(u, v, w, a, b)
print(answer_1d)
```

[ ]:
```
# 1 (e)
def problem_1e(A, B, x):
    # type in the expression after the 'result =' in the next line
    result =
    return result

answer_1e = problem_1e(A, B, x)
print(answer_1e)
```

[ ]:
```
# 1 (f)
def problem_1f(A, b, x):
    # type in the expression after the 'result =' in the next line
    result =
    return result

answer_1f = problem_1f(A, b, x)
print(answer_1f)
```

[ ]:
```
# 1 (g)
def problem_1g(A, B):
    # type in the expression after the 'result =' in the next line
    result_AB =
    result_BA =
    return result_AB, result_BA

answer_1g = problem_1g(A, B)
print(answer_1g)
```

[ ]:
```
# 1 (h)
def problem_1h(A, B):
    # type in the expression after the 'result =' in the next line
    result_LHS =
    result_RHS =
    return result_LHS, result_RHS

answer_1h = problem_1h(A, B)
```

```
print(answer_1h)
```

[ ]:
```
# 1 (i)
def problem_1i(A):
    # type in the expression after the 'result =' in the next line
    result =
    return result

answer_1i = problem_1i(A)
print(answer_1i)
```

Problem 2: Plotting the Lorenz System

What is the Lorenz System? (and some brief computational history):

Edward Lorenz was a meteorologist and mathematician. In 1963, Lorenz derived a simplified atmospheric convection model that consisted of a system of 3 ordinary differential equations. After numerically computing values (on a very early off-the-shelf computer) and plotting some solutions, Lorenz discovered that the system showed some very strange behavior. In fact, Lorenz was the first person to discover deterministic chaos! His work has had a large influence on many fields of science and engineering. You may also have heard of the Butterfly Effect: the idea originates from Lorenz's work.

Problem Instructions:

The goal of this problem is to practice plotting with Matplotlib. In order to do so, you will create a 2D plot of a chaotic solution to the Lorenz System. In the cell below, we have set up, solved, and saved the solution data of the Lorenz System for you. You do not need to worry about the details of how we solved it (but we will learn about numerical integration later in the semester!). Please follow these instructions:

0. Run the cell below without modifying it
1. In the next cell, plot `z` versus `x` with a line color of your choice
2. Add a title to the plot
3. Add x and y axis labels
4. Set the range of the x-axis to [-25, 25]
5. Set the range of the y-axis to [0, 50]

Do not modify the cell below

[ ]:
```
def lorenz(X, t, sigma, beta, rho):
        x, y, z = X
        dxdt = sigma*(y-x)
        dydt = x*(rho - z) - y
        dzdt = x*y - beta*z
        return dxdt, dydt, dzdt

sigma = 10
```

```
beta = 8/3
rho = 28
t = np.linspace(0,50,10000)
solution = scipy.integrate.odeint(lorenz, [0, 1, 0], t, args=(sigma, beta, rho))
x = solution[:,0]
y = solution[:,1]
z = solution[:,2]
```

Make your plot below

[ ]:
```
plt.plot()
plt.xlabel()
plt.ylabel()
plt.title()
plt.xlim()
plt.ylim()

plt.show()
```