**ECE 1111: Engineering Computation I**

**Laboratory No. 14: Application Programming**

|  |  |  |
| --- | --- | --- |
| Problem | Points | Score |
| 0 | 50 | Simply show up… |
| 1 | 60 |  |
| 2 | 70 |  |
| 3 | 80 |  |
| 4 | 90 |  |
| 5 | 100 |  |
| Total | 100 |  |

Notes:

1. For this exam you are allowed to open a terminal window on your computer, you are allowed to web surf with Google, but you cannot use online chat or other interactive services. You can use software tools like ChatGPT to solve particular problems, but your code must be your own original work, and should use concepts discussed in this course.
2. Your code and results should be placed in directories p01, p02, …, p05.
3. You must work the problems in order – you cannot skip a problem. For example, you cannot complete p05 until you have finished p04.
4. Most importantly, only turn in code that you are 100% sure works and is fully debugged. If you turn in a program that doesn’t work, you get a 50.

As usual, place your work here with the proper permissions:

/data/courses/ece\_1111/current/labs/lab\_14/lastname\_firstname

You must use Python 3 for this exam (the version of Python on the AWS server).

This exam is structured in a tiered manner. Start with problem 1. When done, copy your code to the next problem and continue editing it. Only turn in code that is completely working. I will grade the highest level you submit. If that level doesn’t pass my test cases, you will get a 50 for this exam (we call this the “you didn’t debug your code and wasted my time” penalty). Therefore, what you submit must work and meet the stated requirements.

If you write good clean code for problem no. 1, you can reuse that code for the remaining problems and quickly solve those problems.

Also, please resist the temptation to use code generation tools such as ChaptGPT. It is easy to spot code generated from these tools, and the solutions these tools produce often aren’t within the scope of this introductory class.

**P01:** Write a program, *p01.py*, that reads a directory tree, locates all the files ending in (exactly) \*.txt, and stores them in a sorted list. You can use the data here for testing:

/data/courses/ece\_1111/resources/data/matrices

Print your list to stdout. It should look something like this:

ece-000\_[1]: find $PWD/matrices -name "\*.txt" | sort

/data/courses/ece\_1111/resources/data/matrices/d01/d02/f03.txt

/data/courses/ece\_1111/resources/data/matrices/d01/f01.txt

/data/courses/ece\_1111/resources/data/matrices/f00.txt

**P02:** Write a program, *p02.py*, that reads the matrix in each file into a list of matrices. Print the list to *stdout*. For example, in the above example there are three $3x3$ matrices in your list. Your code should work for a matrix of any dimension (e.g., $5x5$) and for an arbitrary number of files.

**P03:** Write a program, *p03.py*, that multiplies each matrix by another matrix in the list and computes the determinant of the result. Print a matrix of determinants – what we call a confusion matrix. Print the confusion matrix to *stdout*. For example, in the above example there are three $3x3$ matrices in your list, so your confusion matrix would be a $3x3$ matrix. The first element of this matrix would be the determinant of the product of the matrix with itself.

To make things easy, assume all the matrices have the same dimension (e.g., they are all 3x3 matrices).

Though we have not covered matrix multiplication and computation of a determinant, part of this exercise is for you to Google search how to do these in Python and implement it. It is fairly easy (one-liners in Python). Those of you taking ENGR 2011 presumably know how to do this by now. You do not need to write original code for this. Python has built-in support for matrix multiplication.

**P04:** Write a program, *p04.py*, that sorts the rows of each matrix based on their values, and rewrites the matrix in sorted order. This means you would convert your list of matrices into a list of new matrices which contains the new matrices in sorted order. The order of the matrix in the list remains the same, but the rows in each matrix are sorted. Do a numeric sort in ascending order using the first value of the row, the second value, ..., until you exhaust all values in a row.

Print your list of sorted matrices to *stdout*.

**P05:** Write a program that does an eigenvalue and eigenvector decomposition of each matrix in your list. Build on your code developed in *p04.py* and add eigenvalue analysis to it. Iterate over your list, print the sorted matrix, and then print the eigenvalues and eigenvectors for that matrix to *stdout*. For a $3x3$ matrix there will be a vector of eigenvalues (3) and a $3x3$ matrix of eigenvectors for each input matrix.

Again, we have not covered the math for this in class. You have or will presumably study this in ENGR 2011. But that never stops us. Google search and you will find a nice description of how to do this in Python. It is another one‑liner. We are testing your ability to digest information in real-time.