Lab 1: NumPy, SciPy and Linear Algebra

CAS CS 132: *Geometric Algorithms*

Due February 5, 2026 by 8:00PM

In this lab, you'll be introduced to the programming tools we'll use throughout this course. We begin with a short installation guide and a review of the basics of NumPy. We'll then benchmark some SciPy functions for solving linear systems. You're required to submit a write-up for this lab as a pdf on Gradescope. The details of what you're required to submit are given in the last section called "Lab Write-up."

Installation Guide

We'll assume access to a unix-style command line. If you are a Windows user, we recommend setting up a WSL environment.[1] Otherwise, we cannot guarantee we'll be able to troubleshoot issues you might run into.

Getting Set Up

1. Install the latest release of Python from the Python webpage.[2] Follow the "Downloads" tab to an installer for your operating system. After doing this, open a terminal and run the command `python3` on the command line to open an interactive Python REPL (Read-Eval-Print-Loop). This will verify that everything was set up correctly. You can use the key combination CTL-D to exit the REPL.

2. The above installation includes a program called `pip3`, a package manager for Python. Run the following command, which will install all the libraries you need for this course.

   ```
   pip3 install numpy scipy matplotlib networkx scikit-learn
   ```

3. (Optional) Install VSCode.[3]

Working in Python

1. Open a file `file_name.py` in your code editor and make the necessary changes.

2. In a terminal, navigate to the directory where your Python file is, e.g.,

   ```
   cd path/to/directory/where/file/is
   ```

3. Run the following command to evaluate the code in that file.

   ```
   python3 file_name.py
   ```

[1] https://learn.microsoft.com/en-us/windows/wsl/install
[2] https://www.python.org
[3] https://visualstudio.microsoft.com/downloads/

If you want to evaluate the file and then go into an interactive Python REPL, use the the flag `-i`, as in:

```
python3 -i file_name.py
```

Tips

You may want to review basic command line usage.

- `ls` shows whats in your current directory
- `pwd` shows the path of the directory you're currently in
- `cd path/to/directory_name` goes to the directory given by the path
- `rm file_name` deletes the file `file_name`[4]
- `rm -r directory_name` deletes the directory named `directory_name` and everything in it
- `touch file_name` creates an empty file called `file_name`
- `mkdir dir_name` creates an empty directory called `dir_name`

I would also generally recommend putting all your code for this course in a single directory, e.g., in your "Documents" folder. You can run the command

```
mkdir -p  /Documents/CS132
```

to create this directory. This is equivalent to going to the "Documents" folder (in Finder for MacOS) and creating a new folder.

Familiarizing yourself with NumPy

The primary objective of this lab is for you to familiarize yourself with NumPy. The course staff will cover the basics of NumPy during your discussion section. On your own you should look through the following resources.

1. Read the NumPy quickstart[5] up to and including the section called "Shape manipulation."
2. Read the supplementary Numpy Tutorial on the course webpage.[6]

Lab Write-Up

1. For reasons that will become clear later in the course, we can use the following to compute an echelon form of a NumPy array `A`:

   ```
   scipy.linalg.lu(A)[2]
   ```

   This is used to implement the function `is_consistent` in the starter code. Read the implementation of this function, and the NumPy docs for each NumPy function used.

   Describe in 1-2 sentences what this code is doing.

[4] This cannot be undone. It's not the same as putting a file in the Trash folder.
[5] `https://numpy.org/devdocs/user/quickstart.html`
[6] `https://nmmull.github.io/CS132-Notes/NumPy/notes.html`

2. SciPy has a function that solves square systems of linear equations called `scipy.linalg.solve`.[7] However, it doesn't take as an argument an augmented matrix, but rather a coefficient matrix and a NumPy array representing the right-hand-sides of the equations of the linear system. Using NumPy slicing, write a wrapper for this function called `solve` which takes as an argument an augmented matrix and calls `scipy.linalg.solve` on the appropriate inputs.

   Note that `scipy.linalg.solve` only works if given a *consistent* linear system with the same number of equations as variables. You should put your call of `scipy.linalg.solve` is a try-except block and return `None` if it fails.

   Include your implementation in your write-up.

3. Look through the starter code given for this lab. The purpose of this code is to benchmark the functions defined above. According to what we've said in class, determining that a system is inconsistent should be faster than determining a solution.

   Put your two functions in the given starter code and run it. Doing so should produce a graph which shows the running time of `is_consistent` and `solve`. Depending on your machine, you may need to update the input to `benchmark` so that the code finishes in a reasonable amount of time.

   It will also show the running time of a function called `getrf`, a FORTRAN function that *actually* computes the echelon form.[8] The point here is to demonstrate that, even though it's technically faster to determine consistency, the overhead from working in Python makes solving about as fast as determining consistency using an echelon form.

   Include the graph produced by the starter code in your write-up. Make sure that the graph demonstrates the separation of the running time of `getrf` from that of the other functions.

4. You'll also notice that the graph includes an approximation of the form an^3 for the running time of `getrf`, and that the starter code prints the coefficient a of this approximation.[9] According to what we've said in class, the number of FLOPs performed by `getrf` should be approximately $(2/3)n^3$. Based on this fact, you should be able to estimate how many FLOPs per second are performed by your CPU.

   Include the coefficient a printed by the starter code in your write-up. Use this number to estimate the number of FLOPs per second performed by your CPU. Justify your calculation in 1-2 sentences.

[7] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.solve.html`

[8] Much of NumPy and SciPy calls FORTRAN and C functions to do the computationally expensive parts of linear algebraic computations.

[9] We will eventually learn how to do this approximation.