Lab 1: NumPy, SciPy, and Linear Algebra

CAS CS 132: Geometric Algorithms

Due September 24, 2026 by 8:00PM

Outline. You'll be introduced to the programming tools we'll use throughout this course. We'll begin 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 below.

Note. Please make sure that you've followed the Installation Guide[1] on the course webpage.

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[2] up to and including the section called "Shape manipulation."
2. Read the supplementary Numpy Tutorial in the course notes.[3]

Lab Write-Up

This section contains the tasks you're required to complete for this lab. Some questions require you to write code, others require you to reason about code. If a question is labeled with a "(*)", it's the kind of question that could appear on a quiz or on an exam.

1. (*) As we'll see later in the course, we can use the following expression to compute an echelon form of a NumPy array `A`:

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

[1] https://nmmull.github.io/cs132-f26/install.html
[2] https://numpy.org/devdocs/user/quickstart.html
[3] https://nmmull.github.io/lecture-notes/linear-algebra/numpy/notes.html

This is used to implement the function `is_inconsistent` 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.

2. (*) SciPy has a function called `scipy.linalg.solve` that solves (square) systems of linear equations.[4] 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, implement the function `split_aug` according to the docstring in `lab1.py`.

   Include your implementation (just your code, not the docstring) in your write-up.

3. Implement the function `mk_aug` according to the docstring in `lab1.py`. This function is the inverse of `split_aug` in the previous problem.

   Include your implementation (just your code, not the docstring) in your write-up.

4. Look through the starter code given for this lab. The purpose of this code is to benchmark consistency checking.

   Running `python3 lab1.py` should produce a graph that shows the running time of `is_inconsistent`. Depending on your machine, you may need to update the input to `benchmark` so that the code finishes in a reasonable amount of time (you should use `n` of at least 100).

   It will also show the running time of a function called `getrf`, a FORTRAN function that *actually* computes the echelon form.[5] The "point" is to demonstrate that there is little overhead from working with NumPy versus working with the FORTRAN function directly. NumPy is what's sometimes called a "thin" wrapper.

   Include the graph produced by the starter code in your write-up. Update the graph so that the running times of `is_inconsistent` are displayed using green triangles.

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

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

NumPy Cheatsheet

After this lab, you should be familiar with the following NumPy concepts.

- general numpy array splicing
- `numpy.isclose` and `numpy.allclose`
- the `shape` property and `numpy.reshape`
- stacking via `numpy.vstack` and `numpy.hstack`
- you should be aware of (but not necessarily fully familiar with) solving methods like `scipy.linalg.lu`, `numpy.linalg.solve`, and `scipy.linalg.solve`