Lab 3: LU Factorization vs. Matrix Inversion

CAS CS 132: *Geometric Algorithms*

Due October 16, 2025 by 8:00PM

This week's lab will be shorter than usual, you'll be benchmarking LU factorization and matrix inversion as implemented in NumPy and SciPy. 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."

Introduction

There are several programming concepts that you'll need for this lab, which we'll cover in turn in the following sections.

Benchmarking Basics

There are several ways to benchmark code in Python. We'll be taking the simplest approach (which is also the approach that we took for Lab 1). There is a module in Python called `time`, and in it a function called `time` which can be used to get the current time. We can use this to determine how much time has been elapsed since a process was started:

```
start = time.time()
process() # thing being timed
total = time.time() - start
```

It's important `start` is determined *just* before the process you want to time. We don't want to include any proprocessing in the time measured.

Randomness

When we benchmark a function, we usually test it on random inputs. We'll use a `numpy.random.generator` to generate these inputs. To create a random number generator, we can use the constructor given in NumPy:

```
rng = np.random.default_rng()
```

We can then use the `random` method to construct random matrices that we'll use for benchmarking:

```
# m by n matrix with random numbers in [0, 1)
a = rng.random((m, n))

# m by n matrix with numbers in [low, high)
a = (hi - low) * rng.random((m, n)) + low

# n elment vector with numbers in [low, high)
a = (hi - low) * rng.random(n) + low
```

Solving Matrix Equations

There are three ways that we've learned to solve matrix equations $A\mathbf{x} = \mathbf{b}$. Each of these ways corresponds to a function in Numpy/Scipy:

- Use Gaussian elimination on the matrix $[A\ \mathbf{b}]$ (`numpy.linalg.solve`)[1]
- Compute the LU Factorization of A and use this to solve $A\mathbf{x} = \mathbf{b}$ (`scipy.linalg.lu_factor` and `scipy.linalg.lu_solve`)
- Determine the inverse A^{-1} and compute $\mathbf{x} = A^{-1}\mathbf{b}$ (`numpy.linalg.inv`)

It's conventional wisdom that you should always use LU factorization if you need to compute the same matrix equation with different right hand sides. This is especially true if the matrix is well-structured (one of the "arts" of solving matrix equations with computers is that different approaches may be more or less efficient depending on the shape of A). We'll be looking into this phenomenon by benchmarking the three approaches above.

Lab Write-Up

The items in **bold** are what must be included in your write-up.

1. Implement the three processes that we want to benchmark. Each one should be of the form

   ```
   process(a, bs)
   ```

   where `a` is matrix and `bs` is a list of vectors. The output should be a list of solutions, one for each right-hand side in `bs`.

   - `solve(a, bs)` which uses `numpy.linalg.solve` to solve each matrix equation.
   - `lu_solve(a, bs)` which uses the function `scipy.linalg.lu_factor` to LU factorize `a`, and then uses `scipy.linalg.lu_solve` to solve each matrix equation. **Important:** This should only factorize once.
   - `inv_solve(a, bs)` which uses `numpy.linalg.inv` to invert `a`, and then uses uses matrix-vector multiplication (`@`) to solve each matrix equation. **Important:** This should only invert once.

   Include your implementations in your write-up.

2. Implement the function `benchmark_random` in the starter code. Look at the docstring to see how this function should work.

   There is nothing to include in your write-up for this part.

3. Read through the function `banded_matrix` in the starter code and try to understand what it is doing.

   Include in the value `banded_matrix(3)` in your write-up.

4. Implement the function `benchmark_banded` in the starter code. Look at the docstring to see how this function should work.

   There is nothing to include in your write-up for this part.

[1] This is not actually true. In the second half of the course we'll see how these functions *actually* work. But for the sake of our benchmarking, they are sufficiently analogous.

5. Using the function `benchmark_random`, create a plot which compares the running time of `solve` and `lu_solve` for between 80 and 100 experiments (depending on what your machine can handle) with step size 10, where each experiment uses the given process to solve 100 equations. Your plot should have a title, labeled axes, and a legend.

   Include the graph in your lab write-up.

6. Using the function `benchmark_random`, create a plot which compares the running time of `inv_solve` and `lu_solve` for between 200 and 300 experiments (depending on what your machine can handle) with step size 10, where each experiment uses the given process to solve 100 equations. Your plot should have a title, labeled axes, and a legend.

   Include the graph in your lab write-up.

7. Using the function `benchmark_banded`, create a plot which compares the running time of `inv_solve` and `lu_solve` for between 8 and 12 experiments (depending on what your machine can handle) with step size 1000, where each experiment uses the given process to solve 100 equations. Your plot should have a title, labeled axes, and a legend.

   Include the graph in your lab write-up.