Homework 1 - Solving Linear Systems

Handout date:	October 11, 2024
Submission deadline:	October 29, 2024, 11:59pm

1 LU Decompositions and Triangular Matrices

Problem 1 (10 pts). Construct by hand an LU decomposition of the following matrix, showing your work:

$$\begin{bmatrix} 5 & 4 & 3 \\ 15 & 19 & 15 \\ 20 & 51 & 44 \end{bmatrix}.$$

Problem 2 (15 pts). Given a *unit* lower triangular matrix L (like in the LU decomposition),

$$L = \begin{bmatrix} 1 & & & & \\ l_{21} & 1 & & & \\ l_{31} & l_{32} & 1 & & \\ \vdots & & & \ddots & \\ l_{m1} & l_{m2} & l_{m3} & \cdots & 1 \end{bmatrix},$$

derive a formula for entries x_{ij} of $X = L^{-1}$. Explain why your formula proves that the inverse of a unit lower-triangular matrix is unit lower-triangular. Warning: the answer is *not* just $x_{ij} = -l_{ij}$!

*Hint: column j is the solution to $L\mathbf{x}_j = \mathbf{e}_j$. You can give your formula for entry (i, j) in "cases" notation (*e.g.*, giving separate formulas for the cases when $i < j$, $i = j$, and $i > j$). Your formula may be recursive.*

Problem 3 (5 pts). Explain what changes in your formula when L is a general invertible lower-triangular matrix and how it reveals that the inverse of a lower-triangular matrix is lower-triangular.

Problem 4 (5 pts). Use the result of the previous problem and what you know about matrix transposition to give a *short* proof that the inverse of an upper triangular matrix is upper triangular.

Problem 5 (10 pts). Suppose $A \in \mathbb{R}^{n\times n}$ is nonsingular and admits a factorization $A = LU$ with ones on the diagonal of L. Prove that such a decomposition is unique. *Hint: consider two factorizations $A = L_1U_1 = L_2U_2$, isolate the L factors on one side of this equation, the U factors on the other, and reason about the structure of each side.*

2 Block Matrices

As we saw when vectorizing the LU decomposition and deriving the Cholesky factorization, it can be useful to group the entries of a matrix into blocks. A generic matrix $A \in \mathbb{R}^{m\times n}$ can be decomposed into a block matrix with $q \times r$ blocks:

$$A = \begin{bmatrix} A_{11} & \dots & A_{1r} \\ \vdots & & \vdots \\ A_{q1} & \dots & A_{qr} \end{bmatrix} \begin{matrix} m_1 \\ \\ m_q \end{matrix}, \qquad \begin{matrix} n_1 & & n_r \end{matrix}$$

where each block is a submatrix $A_{ij} \in \mathbb{R}^{m_i \times n_j}$ with $m_1 + \cdots + m_q = m$ and $n_1 + \cdots + n_r = n$.

Many of the operations we perform on matrices respect this block structure:

- Linear combinations

$$\alpha \begin{bmatrix} A_{11} & A_{12} & \dots \\ A_{21} & A_{22} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} + \beta \begin{bmatrix} B_{11} & B_{12} & \dots \\ B_{21} & B_{22} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} = \begin{bmatrix} \alpha A_{11} + \beta B_{11} & \alpha A_{12} + \beta B_{12} & \dots \\ \alpha A_{21} + \beta B_{21} & \alpha A_{22} + \beta B_{22} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

- Transposition

$$\begin{bmatrix} A_{11} & A_{12} & \dots \\ A_{21} & A_{22} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}^\top = \begin{bmatrix} {A_{11}}^\top & {A_{21}}^\top & \dots \\ {A_{12}}^\top & {A_{22}}^\top & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

- Matrix multiplication $C = AB$ for $A \in \mathbb{R}^{p\times m}$ and $B \in \mathbb{R}^{m\times n}$ decomposed into blocks:

$$A = \begin{bmatrix} A_{11} & \dots & A_{1q} \\ \vdots & & \vdots \\ A_{s1} & \dots & A_{sq} \end{bmatrix} \begin{matrix} p_1 \\ \\ p_s \end{matrix}, \qquad \begin{matrix} m_1 & & m_q \end{matrix} \qquad B = \begin{bmatrix} B_{11} & \dots & B_{1r} \\ \vdots & & \vdots \\ B_{q1} & \dots & B_{qr} \end{bmatrix} \begin{matrix} m_1 \\ \\ m_q \end{matrix} \qquad \begin{matrix} n_1 & \dots & n_r \end{matrix}$$

$$\implies \quad C = \begin{bmatrix} C_{11} & \dots & C_{1r} \\ \vdots & & \vdots \\ C_{s1} & \dots & C_{sr} \end{bmatrix} \begin{matrix} p_1 \\ \\ p_s \end{matrix} \qquad \begin{matrix} n_1 & \dots & n_r \end{matrix} \qquad \text{with} \quad C_{\alpha\beta} = \sum_{\gamma=1}^{q} A_{\alpha\gamma} B_{\gamma\beta}.$$

This block matrix multiplication rule is identical to the ordinary matrix multiplication formula, except scalar operations (multiplication and addition) are replaced with operations on the matrix blocks. Of course, for this to work, each block size must be compatible as indicated by the column block sizes of A $(m_1, \cdots, m_q)$ equaling the row block sizes of B. A simple example:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}.$$

Problem 6 (20 pts). Suppose we have a matrix $M \in \mathbb{R}^{n\times n}$ decomposed into blocks:

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix},$$

where $A \in \mathbb{R}^{k\times k}$ is nonsingular.

a) Verify that the following "block LU decomposition" formula holds:

$$M = \begin{bmatrix} I & 0 \\ CA^{-1} & I \end{bmatrix} \begin{bmatrix} A & B \\ 0 & D - CA^{-1}B \end{bmatrix}.$$

The matrix $D - CA^{-1}B$ is called the Schur complement of A and is the matrix we get after eliminating the first block of unknowns $\mathbf{x}_1$ in the system $\begin{bmatrix} A & B \\ C & D \end{bmatrix}\begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix}$ via the formula $\mathbf{x}_1 = A^{-1}(\mathbf{b}_1 - B\mathbf{x}_2)$; plugging this formula into the second block row of the equation yields:

$$\mathbf{b}_2 = C\mathbf{x}_1 + D\mathbf{x}_2 = CA^{-1}(\mathbf{b}_1 - B\mathbf{x}_2) + D\mathbf{x}_2 \quad \Longrightarrow \quad (D - CA^{-1}B)\mathbf{x}_2 = \mathbf{b}_2 - CA^{-1}\mathbf{b}_1.$$

b) Describe how you can construct the full LU decomposition of M by computing an LU decomposition of A, evaluating and factorizing $D - CA^{-1}B$, and finally manipulating the result using efficient operations (and never explicitly constructing A^{-1})!

3 Python Implementation and Benchmarking

The following problems will give you some practice implementing matrix algorithms in Python; you should write all your code in the file `linear_systems.py`

First, make sure that you have a working version of Python and the `numpy`, `scipy`, and `matplotlib` libraries installed. I recommend that you use a virtual environment when installing Python packages. If you have not worked with virtual environments before, you can create one with the following commands on Linux or macOS:

```
python3 -m venv $HOME/sc_env
. $HOME/sc_env/bin/activate
```

This creates and activates a Python virtual environment in the directory `$HOME/sc_env`; feel free to use a different location on your system. When the virtual environment is successfully activated, your shell prompt should change to include the string `(sc_env)`.

With your environment activated, you can install the necessary packages with:

```
pip install numpy scipy matplotlib
```

I also recommend installing JupyterLab into your environment (`pip install jupyterlab`) to get a nice web-based interface for running Python code interactively.

Problem 7 (10 pts). Translate the LU decomposition algorithm pseudocode into Python. Because our pseudocode notation is very similar to Python, this is largely just an exercise in copying and pasting. However, you will need to be careful about the loop bounds since Python uses 0-based indexing. After completing your implementation, the first two cases of the provided unit tests should pass when you run `python3 linear_systems.py`.

Problem 8 (10 pts). Implement the forward substitution algorithm to efficiently solve $L\mathbf{x} = \mathbf{b}$ with lower triangular matrix L.

Problem 9 (5 pts). Implement the back substitution algorithm to efficiently solve $U\mathbf{x} = \mathbf{b}$ with upper triangular matrix U.

Problem 10 (5 pts). Use the code from the previous two problems to solve $A\mathbf{x} = \mathbf{b}$ using the LU decomposition approach.

After completing all problems above correctly, all unit tests should pass.

Problem 11 (5 pts). If you run `python3 linear_systems.py -b`, the provided code will benchmark your implementation of `forwardsub` on a collection of progressively larger examples, plotting runtimes in `forwardsub_benchmarks.pdf`. It also compares your timings against a high-performance implementation from `scipy`. Include this plot in your solution PDF and comment on whether it demonstrates the scaling law you expect (and why).

4 Optional: Vectorization

Eliminate the inner for loop in your `forwardsub` implementation by replacing it with a single vector operation. Register your new implementation in the `forwardsub_implementations` dictionary so that it gets tested and benchmarked. Include the expanded timing plot in the report.

A note on our grading policy: we generally won't assign partial credit to programming problems like these for two reasons (1) the pseudocode in the slides typically gives away most of the solution; and (2) we don't have the TA support to debug each incorrect implementation to identify what went wrong. We will of course still read through your code to check for violations of the collaboration policy or other improprieties (like LLM use or calling `numpy`/`scipy` routines for the parts you were asked to implement). Please use the unit tests to check if your solution is correct and seek help if you're stuck with failed test cases. If you miss all the points on a programming task, we will give you an opportunity to fix your bugs, explain to us what was wrong, and, depending on the problem, potentially recover some of the points.

5 What to Turn In

Please upload a *single zip file* to Canvas containing:

- a PDF of your solutions (typesetting preferred but not required as long as solutions are legible);
- your completed code file `linear_systems.py`.

6 Late Work and Collaboration Policy

Late work will be accepted with a deduction of 10pts per day up until graded homework is returned. The deadline is strict: homework submitted after 11:59pm will be considered one day late.

Discussing the problems with classmates is allowed, but you may not collaborate as you write up your solutions, and you must not at any point look at another student's written solutions. You also must not search for solutions online: if you are stuck on a problem, please message me or post a question to the Canvas discussion board.