HOMEWORK
David F. Gleich
October 17, 2025

Homework 3b

Please answer the following questions in complete sentences in a clearly prepared manuscript and submit the solution by the due date on Gradescope.

Remember that this is a graduate class. There may be elements of the problem statements that require you to fill in appropriate assumptions. You are also responsible for determining what evidence to include. An answer alone is rarely sufficient, but neither is an overly verbose description required. Use your judgement to focus your discussion on the most interesting pieces. The answer to "should I include 'something' in my solution?" will almost always be: Yes, if you think it helps support your answer.

Problem 0: Homework checklist

- Please identify anyone, whether or not they are in the class, with whom you discussed your homework. This problem is worth 1 point, but on a multiplicative scale.
- Make sure you have included your source-code and prepared your solution according to the most recent Piazza note on homework submissions.

Problem 1: Recursive block elimination

In class, we saw "element-wise" variable elimination. That is, in the first step, we eliminated the the first variable from the problem and built a new problem for every other variable.

In fact, we can do this for multiple variables too. There's only one small catch that has to do with pivoting, but we won't worry about that for this problem. That is, you can assume everything you need to be non-singular, is actually non-singular.

Why this matters A recent innovation are the presence of large "tensor-cores" that support "matrix multiplication" directly on GPUs or other CPU accelerators. These can make matrix-multiplication operations much much faster than expected. We are going to use them to solve a linear system via block-substitution. As a simple example, note that Apple's recent M-series chip includes a dedicated "matrix-multiplication" instruction! As another example, Intel has their own extensions too

Your problem Write down the steps of an algorithm to solve $\boldsymbol{A}\mathbf{x} = \mathbf{b}$ via recursive block-elimination. That is, divide $\boldsymbol{A}$ into blocks:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_1 & \boldsymbol{A}_2 \\ \boldsymbol{A}_3 & \boldsymbol{A}_4 \end{bmatrix}$$

where all blocks have the same size. In this case, assume that $\boldsymbol{A}$ has a power-of-two dimension so that you can do this recursively as well until you get down to a 1×1 system. Then if

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix}$$

are the vectors that conform to the partitioning of $\boldsymbol{A}$, develop an algorithm that uses as a subroutine:

```
# return the matrix after solving k linear systems $A^{-1} B$
function solve(A::Matrix, B::Matrix)
end
```

and matrix-matrix multiplication to solve the linear system. Operations you are allowed:

- submatrix extraction, e.g. `A1 = A[1:div(n,2),1:div(n,2)]`
- addition and subtractions of matrices or vectors, e.g. `A .- B`
- matrix-vector or matrix-matrix multiplication, e.g. `A*b` or `A*B`
- recursive calls to `solve`
- you are allowed to use division only for a `n=1` case.

For testing, you can use the fact that pivoting isn't required for symmetric positive definite matrices. So just compute a random matrix $\boldsymbol{Z}$ and let $\boldsymbol{A} = \boldsymbol{Z}^T\boldsymbol{Z}$.

Notes You may find it helpful to think about doing this with only *two* variables at a time first. Here, I write down a few small steps in the process to get you started. *I have not fully debugged these notes as they are meant as a supplementary guide to this problem as a bridge from the lecture notes to the question*

Let

$$\boldsymbol{A} = \begin{bmatrix} \alpha & \beta & \mathbf{c}^T \\ \gamma & \delta & \mathbf{d}^T \\ \mathbf{f} & \mathbf{g} & \boldsymbol{R} \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} \theta \\ \nu \\ \mathbf{h} \end{bmatrix}$$

And let

$$\mathbf{x} = \begin{bmatrix} \omega \\ \mu \\ \mathbf{y} \end{bmatrix}.$$

Let $\boldsymbol{A}_1 = \begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix}$. Then note that $\boldsymbol{A}_1 \begin{bmatrix} \omega \\ \mu \end{bmatrix} + \begin{bmatrix} \mathbf{c}^T \\ \mathbf{d}^T \end{bmatrix} \mathbf{y} = \begin{bmatrix} \theta \\ \nu \end{bmatrix}$. So like we found in class, we can find ω and μ given $\mathbf{y}$. (Assuming that $\boldsymbol{A}_1$ is non-singular. (Which, to be clear, we are assuming in this problem!) Let $\mathbf{z} = \begin{bmatrix} \omega \\ \mu \end{bmatrix}$. The last expression shows that we can write $\mathbf{z}$ as a function of $\mathbf{y}$. At this point, we can then substitute $\mathbf{z}(\mathbf{y})$ in like we did for the other cases in the class notes.

Problem 2: More on Cholesky

Note, we often use either $\boldsymbol{F}$ or $\boldsymbol{L}$ as the Cholesky factor.

Your professor also boldy asserted that we preserve positive definiteness after one step of the Cholesky factorization. Recall that this was:

$$\boldsymbol{A} = \begin{bmatrix} \alpha & \mathbf{b}^T \\ \mathbf{b} & \boldsymbol{C} \end{bmatrix} = \boldsymbol{F}\boldsymbol{F}^T = \begin{bmatrix} \gamma & 0 \\ \mathbf{f} & \boldsymbol{F}_1 \end{bmatrix} \begin{bmatrix} \gamma & \mathbf{f}^T \\ 0 & \boldsymbol{F}_1^T \end{bmatrix}.$$

So we set $\gamma = \sqrt{\alpha}$ and $\mathbf{f} = \mathbf{b}/\gamma$. Then, we recursively compute the Cholesky factorization of

$$\boldsymbol{C} - \mathbf{b}\mathbf{b}^T/\alpha = \boldsymbol{F}_1\boldsymbol{F}_1^T.$$

This only works if $\boldsymbol{C} - \mathbf{b}\mathbf{b}^T/\alpha$ is positive definite. This is discussed in the notes. (Hint: you can do it from the definition of positive definiteness: $\mathbf{x}^T\boldsymbol{A}\mathbf{x} > 0$ for *all* $\mathbf{x}$.)

1. *Briefly* explain why this step justifies that a Cholesky factorization exists for any positive definite matrix. (This is really like a one or two sentence answer.)

2. One of the most useful aspects of the Cholesky factorization is as a way to *check* if a matrix is positive definite *or* negative definite. A matrix is negative definite if

$$\mathbf{x}^T \boldsymbol{A}\mathbf{x} < 0$$

for all $\mathbf{x}$. **Change** our Julia implementation to report if a matrix is not positive definite. (Hint: this relates to *why* a Cholesky factorization always exists for a positive definite matrix in the previous problem.)

Use this to test if the matrix for Poisson's equation from Homework 1 is positive or negative definite.

Problem 3: Direct Methods for Tridiagonal systems (Even more on Cholesky!)

In class, we said the definition of a sparse matrix was one with enough zeros to take advantage of it. In this problem, we'll show how to take advantage of tridiagonal structure in a matrix.

Make sure you have read through the notes on the Cholesky factorization in our notes on Elimination methods.

Let $\boldsymbol{A}$ be a symmetric, tridiagonal, positive definite matrix:

$$\boldsymbol{A} = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{n-2} & \alpha_{n-1} & \beta_{n-1} \\ & & & \beta_{n-1} & \alpha_n \end{bmatrix}$$

1. Let $\boldsymbol{A}_{n-1}$ is the $n-1 \times n-1$ leading prinicipal minor of $\boldsymbol{A}$. Suppose you are given the Choleksy factorization of $\boldsymbol{A}_{n-1} = \boldsymbol{L}_{n-1}\boldsymbol{L}_{n-1}^T$. Determine the structure of $\boldsymbol{L}_{n-1}$ (hint, $\boldsymbol{L}_{n-1}\boldsymbol{L}_{n-1}^T$ must be tridiagonal!).
2. Now, show how to compute the Cholesky factorization of $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^T$ from $\boldsymbol{L}_{n-1}$ in a constant number of operations.
3. Use the result of the first two parts to write down an algorithm that computes the Cholesky factorization of $\boldsymbol{A}$ in a linear amount of work (starting from scratch!)

Problem 4: Direct methods for eigenvectors.

Suppose you try to compute an eigenvector using the "elimination" techniques from class. This question will ask you to investigate the issues that arise.

`First, we will assume we know the eigenvalue $\lambda$ and that this`

eigenvector is simple. (Don't worry if you don't know what this means. What it means for this problem is that the linear system we build only has the mild issue we discuss now.) Consequently, we wish to find a solution of the linear system:

$$\boldsymbol{A}\mathbf{x} = \lambda\mathbf{x} \qquad \text{or} \qquad (\boldsymbol{A} - \lambda\boldsymbol{I})\mathbf{x} = 0.$$

This linear system is singular because any scalar multiple of an eigenvector is also a solution. However, this might be considered a mild singularity as we simply get to choose the scale of the solution. (If the eigenvalue was not simple, then the solution would be more complicated, that's why we assume simple, which formally means that $\text{rank}(\boldsymbol{A} - \lambda\boldsymbol{I}) = n - 1$)

Explain what happens when we try and use our elimination solver `solve1_pivot2` using this system and also alter this routine so that we are able to produce an eigenvector when given the matrix $\boldsymbol{A}$ and λ.

Problem 5: Givens vs. Householder

(Based on class discussion.) In this question, you'll study some of the questions that came up in class. Given a vector $\mathbf{a}$, we can use a single Householder transform to produce an orthogonal matrix $\boldsymbol{H}$ such that $\boldsymbol{H}\mathbf{a} = \pm\|\mathbf{a}\|\mathbf{e}_1$. We can also use a set of $n-1$ givens rotations:

$$\boldsymbol{G}_{n-1}\boldsymbol{G}_{n-2}\cdot\boldsymbol{G}_1\mathbf{a} = \pm\|\mathbf{a}\|\mathbf{e}_1.$$

1. Compare (in theory) the orthogonal matrices that arise from these two ways of looking at the problem for a length 2 vector.
2. Compare (in theory) the orthogonal matrices that arise from these two ways of looking at the problem for a length 3 vector.