M340L-CS, SPRING 2024
Matrices and Matrix Computations

Project: QR decomposition

Theory. We have seen an application of the QR decomposition to least square problems. Later, we will also use it as a part of many other numerical linear algebra algorithms. It is therefore useful to have some *efficient* and *numerically stable* algorithms.

The focus of this project is on the stability part. We will compare three different QR decomposition algorithms: Gram–Schmidt, modified Gram–Schmidt, Householder reflections. If A is an $n \times k$ matrix, and the program outputs an $n \times k$ matrix Q and a $k \times k$ matrix R, then we want

$$Q^T Q = I_k, \quad A = QR,$$

so two good measures of error are

$$\varepsilon_\perp = \|Q^T Q - I_k\|_{\max}, \quad \varepsilon_s = \|A - QR\|_{\max}$$

where $\|X\|_{\max}$ for a matrix X just means the largest absolute value among all of its entries.

Problems.

1. Write a program to perform Gram–Schmidt on an $n \times k$ matrix A, i.e. perform the following procedure for $i = 1, \cdots, k$.

$$\begin{aligned} \mathbf{v}_i &\leftarrow \mathbf{v}_i - (\mathbf{q}_1 \cdot \mathbf{v}_i)\mathbf{q}_1 - \cdots - (\mathbf{q}_{i-1} \cdot \mathbf{v}_i)\mathbf{q}_{i-1} \\ \mathbf{q}_i &\leftarrow \mathbf{v}_i / \|\mathbf{v}_i\| \end{aligned}$$

   The matrix Q is $[\mathbf{q}_1 \ \cdots \ \mathbf{q}_k]$. Compute R using $R = Q^T A$. Output the matrices Q and R, as well as the two errors $\varepsilon_\perp$, ε_s.
   (Make sure you are not accidentally doing the modification described in the following part. This could happen from passing by reference.)

2. In a different function, make the following modification to the iteration step

$$\begin{aligned} &\texttt{for } j = 1, \cdots, i-1: \\ &\quad \mathbf{v}_i \leftarrow \mathbf{v}_i - (\mathbf{q}_j \cdot \mathbf{v}_i)\mathbf{q}_j \\ &\mathbf{q}_i \leftarrow \mathbf{v}_i / \|\mathbf{v}_i\| \end{aligned}$$

   In other words, instead of computing all of the coefficients at once, we compute them one at a time, updating $\mathbf{v}_i$ before each computation. The output should consists of the same quadruple $(Q, R, \varepsilon_\perp, \varepsilon_s)$ as in part 1.

3. Recall the Householder reflection matrix

$$R = I_n - \frac{2\mathbf{w}\mathbf{w}^T}{\mathbf{w}^T\mathbf{w}}$$

   This is not a sparse matrix in general. Despite this, describe a way to compute RA for an arbitrary A with $O(n^2)$ operations (the usual way would take $O(n^3)$ operations).

4. Using this, implement the Householder reflection method for QR factorization. See the supplementary notes for Lecture 21 for details. To compute Q, you might find it helpful to initialize $B = I_n$, and every time you apply a reflection to the input A, also do it to B. The result is Q^T. You are recommended to do the calculations in place, overwriting entries of the input (or a deep copy of the input).

   Its output is the same quadruple as before, so if the input is not a square matrix, you should truncate Q and R as described in Homework 7, Q3.

5. How many operations do the three methods take on an $n \times k$ matrix? Give your answer using the big O notation, ignoring constants.

The following questions will test your three programs on both random matrices and the infamous Hilbert matrices, giving you a sense of their numerical stability.

6. Run the three programs on some 200×200 matrices whose entries are randomly selected between -1 and 1. Write down the average of $\varepsilon_\perp$. How does this average compare?

7. You should have gotten reasonably accurate results, but this is not necessarily a good model for matrices showing up in real life. The Hilbert matrix is the $n \times n$ matrix with entries

$$H_{ij} = \frac{1}{i+j-1}, \quad i, j = 1, \cdots, n$$

This is a famous example of an *ill-conditioned* matrix, meaning it tends to drastically amplify round off errors, which makes numerical algorithms challenging. Since we are normalizing the columns of Q, the maximum possible value of $\varepsilon_\perp$ is 1. In this case, the supposedly orthogonal Q is close to being singular, and it is completely useless for any application.

For each of the three methods, find the smallest n such that $\varepsilon_\perp > 0.9999$ when applied to the $n \times n$ Hilbert matrix. If this is taking too much time, just say you can't find an n.

8. We introduce a regularized version of the Hilbert matrix, which is slightly better behaved. Let $\varepsilon = 0.0001$. Define

$$H_\varepsilon = H + \varepsilon I_n$$

Apply the three methods to H_ε for $n \leq 100$. Make a graph which shows how the errors (both $\varepsilon_\perp$ and ε_s) depend on n. It is useful to put logarithmic scale on the y-axis.

9. Give some plausible reasons for the difference in errors across the three methods.

Remarks/Hints. It might be of interest to look up the *condition number* of a matrix. Roughly speaking, this measures the amount of distortion that happens when A is applied. Large distortion means a small error in the input becomes a noticeable error in the output. By one definition, the condition number of a matrix is the quotient of its largest and its smallest singular value. In many practical settings, the matrices used have a few large singular values, and most of the other singular values are very close to 0, representing noise. This is one reason ill-conditioned matrices show up in actual problems. The regularization of adding εI_n keeps the smallest singular value greater than ε without significantly modifying the larger ones, so its condition number is smaller.

The Hilbert matrix famously has a large condition number. On the other extreme, orthogonal matrices have condition number 1 (in the L^2-norm). Intuitively, orthogonal matrices are just (a sequence of) rotations or reflections, and a small error in any direction stays small, just in a possibly different direction. This is why in ill-conditioned problems, orthogonal matrices are preferred. This further explains why the singular value decomposition is so useful.

In practice, when using Householder reflections, Q is almost never stored as a matrix. It is usually stored as a sequence of reflection matrices, and each one only takes $O(n)$-space to store the vector $\mathbf{w}$. To multiply by Q, one uses the type of methods you found in part 6.