These are the lecture notes for CSC349A Numerical Analysis taught by Rich Little in the Spring of 2018. They roughly correspond to the material covered in each lecture in the classroom but the actual classroom presentation might deviate significantly from them depending on the flow of the course delivery. They are provide as a reference to the instructor as well as supporting material for students who miss the lectures. They are simply notes to support the lecture so the text is not detailed and they are not thoroughly checked. Use at your own risk. They are complimentary to the handouts. Many thanks to all the guidance and materials I received from Dale Olesky who has taught this course for many years and George Tzanetakis.

1 Matrix Inverses

- This topic is discussed in the textbook in Section 10.2 in terms of an LU decomposition (which is just another way of interpreting Gaussian elimination).
- We are omitting Chapter 10.
- The following material is similar to that in Section 10.2 but is not described in terms of an LU decomposition.
- This lecture also corresponds to Handout 17.

If a matrix A is square and nonsingular, then there exists another matrix A^{-1}, called the *inverse* of A, such that

$$AA^{-1} = A^{-1}A = I$$

where I is called the *identity* matrix,

$$I = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

Somtimes it is useful or necessary to calculate A^{-1}. For example, consider a system of linear equations

$$Ax = b$$

If we knew A^{-1}, then we could calculate x directly since

$$\begin{aligned} Ax &= b \\ A^{-1}Ax &= A^{-1}b \\ Ix &= A^{-1}b \\ x &= A^{-1}b \end{aligned}$$

What does it get us?

- Calculate the inverse A^{-1} once, then multiply it by many different b's to calculate various x's.
- No need to go through the entire elimination process repeatedly for the same A.
- Remember that solving $Ax = b$ costs $\frac{2n^3}{3} + O(n^2)$ each time whereas calculating $A^{-1}b$ each time is just the cost of multiplying matrices which is approximately $2n^2$.
- Section 10.2.2 discusses the importance of this in engineering, where typically A contains equations describing some interaction model and b contains a series of constants representing different states of stimulus to the system.

How do we calculate A^{-1}?

Solve n systems of n equations. Let A be an $n \times n$ matrix and let the n unknown column vectors of A^{-1} be $x^{(1)}, x^{(2)}, ..., x^{(n)}$, then solve

$$Ax^{(1)} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, Ax^{(2)} = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \ldots, Ax^{(n)} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

Then, $A^{-1} = [x^{(1)}|x^{(2)}|\ldots|x^{(n)}]$.

The Algorithm

- Apply Gaussian elimination (with partial pivoting) to the $n \times 2n$ augmented matrix $[A|I]$.

- This will have a higher cost than solving some $Ax = b$ one time.
- In the inner most loop j will go to $2n$ each time instead of n.
- There will be n back substitutions instead of 1.
- This comes out to roughly $2(\frac{4n^3}{3})$ flops but can be reduced to $2n^3$.
- But, it only needs to be done once

Example:
Let

$$A = \begin{bmatrix} 4 & 3 \\ 3 & 2 \end{bmatrix}$$

and calculate A^{-1}.
Here, we are solving

$$Ax^{(1)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ and } Ax^{(2)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

by solving,

$$\left[\begin{array}{cc|cc} 4 & 3 & 1 & 0 \\ 3 & 2 & 0 & 1 \end{array}\right]$$

Important Comment
That said, we usually want to avoid calculating A^{-1} for one-time use. For example, if you are given A and b and are asked to compute $A^{-1}b$.

- you would compute A^{-1} as above for a cost of $2n^3$ flops.
- then compute $A^{-1}b$ for a cost of $2n^2$ flops
- giving a total of $2n^3 + O(n^2)$ flops.

Instead, note that given A and b we can solve $Ax = b$ for x, which happens to be equal to $A^{-1}b$, all for the cost $\frac{2n^3}{3} + O(n^2)$ flops.

- Similarly, if $A^{-1}B$ is needed, you can solve for $AX = B$ to get $X = A^{-1}B$.

2 Stability and Condition of Systems of Linear Equations

2.1 Stability of Algorithms for Solving $Ax = b$

- Given a nonsingular matrix A, a vector b and some algorithm for computing the solution of $Ax = b$, let $\hat{x}$ denote the computed solution using this algorithm.
- The computation is said to be stable if there exist small perturbations E and e of A and b, respectively, such that $\hat{x}$ is close to the exact solution y of the perturbed linear system

$$(A + E)y = b + e$$

- That is, the computed solution $\hat{x}$ is very close to the exact solution of some small perturbation of the given problem.

Known Results

- Gaussian elimination without pivoting may be unstable.
- In practice, Gaussian elimination with partial pivoting is almost always stable.
- A much more stable version of Gaussian elimination uses complete pivoting, which uses both row and column interchanges.
- However, as this algorithm is much more expensive to implement and since partial pivoting is almost always stable, complete pivoting is seldom used.

2.2 Condition of $Ax = b$

- A given problem $Ax = b$ is ill-conditioned if its exact solution is very sensitive to small changes in the data $[A|b]$.

- That is, if there exist small perturbations E and e of A and b, respectively, such that $x = A^{-1}b$ is not close to the exact solution y of the perturbed linear system

$$(A + E)y = b + e,$$

  then the linear system $Ax = b$ is ill-conditioned.

- If such perturbations E and e do not exist, then $Ax = b$ is well conditioned.

- **Example:** $n \times n$ Hilbert matrices are ill-conditioned.