Lecture 15: Linear Algebra II

Instructor: Professor Amos Ron Scribes: Mark Cowlishaw, Nathanael Fillmore

Recall that last time, we were discussing the numerical solution of systems of linear equations.

$$A\vec{x} = \vec{b}$$

We shall discuss four important aspects of numerical solutions to linear systems:

1. **Algorithms** - What methods can we use to find numerical solutions to linear systems.
2. **Complexity/Cost** - What is the cost of each algorithm. This will determine how large a linear system can be solved in a reasonable amount of time.
3. **Conditioning** - How *good* is the system we are trying to solve? Some (ill-conditioned) systems of equations will pose problems for any numerical algorithm.
4. **Stability** - How *reliable* are the algorithms.

As we discussed last time, all of the direct, finite algorithms for solving linear systems use *factorization*, which is decomposing the matrix A into *simple* matrices, then solving the resulting set of linear equations.

1 Simple Matrices

What do we mean by a simple matrix? We shall discuss three kinds of simple matrices:

- Lower Triangular Matrices
- Upper Triangular Matrices
- Orthogonal Matrices

1.1 Lower Triangular Matrices

A *lower triangular* matrix is a matrix in which all the elements above the diagonal are zero ($A = (a_{ij}), \forall_{j>i}\ a_{ij} = 0$).

$$A_{N\times N} = \begin{bmatrix} a_{11} & 0 & 0 & \dots & 0 \\ a_{21} & a_{22} & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{N1} & a_{N2} & \dots & a_{NN-1} & a_{NN} \end{bmatrix}$$

If we are given the system

$$A\vec{x} = \vec{b}$$

with A lower triangular, then the first row corresponds to the equation

$$a_{11} \cdot x_1 = b_1$$

We can then substitute the resulting value for x_1 into the equation that corresponds to the second row of A.

$$a_{21} \cdot x_1 + a_{22} \cdot x_2 = b_2$$

Leaving only one unknown to solve for. Clearly, we can continue this process until we have determined each entry of $\vec{x}$, solving a single equation with a single unknown at each step. This process is known as *forward substitution*. How many computations does this process require? In general, we must perform one multiplication for each non-zero entry in A. There is 1 non-zero entry in row 1, 2 non-zero entries in row 2, and so on, yielding:

$$\sum_{i=1}^{N} i = \frac{N(N-1)}{2} \approxeq \frac{N^2}{2}$$

So, we must perform about $N^2/2$ calculations to solve a linear system containing a lower-triangular matrix. Note that this is essentially the theoretical optimum cost, since we are performing about one calculation per non-zero entry in the matrix. In general, we expect that any algorithm for solving a general linear system must perform at least one calculation per entry in the matrix, or $O(N^2)$ operations in total, so that forward substitution is quite a good method (when it is possible to use it).

For example if we are given the system

$$\begin{cases} 2x_1 = 7 \\ 3x_1 + 5x_2 = 8 \\ 4x_1 - 4x_2 + x3 = -2 \end{cases}$$

we solve

$$2x_1 = 7 \quad \text{so} \quad x_1 = 3.5$$

then

$$3 \cdot 3.5 + 5x_2 = 8 \quad \text{so} \quad x_2 = -0.5$$

and finally

$$4 \cdot 3.5 - 4 \cdot (-0.5) + x3 = -2 \quad \text{so} \quad x_3 = -2 - 4 \cdot 3.5 + 4 \cdot (-0.5) = -18.$$

1.2 Upper Triangular Matrices

Similarly, an *upper triangular* matrix is a matrix in which all the elements below the diagonal are zero ($A = (a_{ij}), \forall_{j<i}\, a_{ij} = 0$).

$$A_{N\times N} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1N-1} & a_{1N} \\ 0 & a_{22} & a_{23} & \dots & a_{2N} \\ 0 & 0 & a_{33} & \dots & a_{3N} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & a_{NN} \end{bmatrix}$$

If we are given the system

$$A\vec{x} = \vec{b}$$

with A upper triangular, then the final row corresponds to the equation

$$a_{NN} \cdot x_N = b_N$$

As before, we can solve this equation and substitute the resulting value for x_N into the equation for the row above. We can continue this process, solving one equation with one unknown at each step, until the entire linear system is solved. We call this process *backward substitution*. As before, we must perform about $N^2/2$ calculations to solve a linear system containing an upper triangular matrix using this method.

1.3 Orthogonal Matrices

An *orthogonal matrix* is a square matrix with inverse equal to its transpose, that is:

$$A \cdot A^T = A^T \cdot A = I$$

For example, consider the matrix Q:

$$Q = \begin{bmatrix} 0.6 & 0.8 \\ -0.8 & 0.6 \end{bmatrix} \qquad Q^T = \begin{bmatrix} 0.6 & -0.8 \\ 0.8 & 0.6 \end{bmatrix}$$

It is easy to verify that $Q \cdot Q^T = Q^T \cdot Q = I$. Note that if we can write a linear system in terms of an orthogonal matrix Q, then we can easily find the inverse by transposing the matrix, and use this to find the solution:

$$\begin{aligned} Q \cdot \vec{x} &= \vec{b} \\ Q^T \cdot (Q \cdot \vec{x}) &= Q^T \cdot \vec{b} \\ (Q^T \cdot Q) \cdot \vec{x} &= Q^T \cdot \vec{b} \\ I \cdot \vec{x} &= Q^T \cdot \vec{b} \\ \vec{x} &= Q^T \cdot \vec{b} \end{aligned}$$

However, the seeming simplicity of obtaining the solution is not enough, we must *quantify* just how costly this operation is. To get the solution, we multiply the vector $\vec{b}$ by the matrix Q^T. Consider that, for $\vec{b} \in \mathbb{C}^N$, to calculate each element $(Q^T \cdot \vec{b})_i$, we must perform N multiplications. Since there are N such elements, there are N^2 multiplications in total. This has the same order of growth as the methods of forward and back substitution ($O(N^2)$), but requires roughly twice as many multiplications.

2 Factorization Algorithms

The idea of factorization is to take a linear system

$$A \cdot \vec{x} = \vec{b} \tag{1}$$

and transform A into the product of simple matrices B and C:

$$A \quad = \quad B \cdot C$$

We can then express our system as

$$\begin{aligned} & Ax = b \\ \iff & (BC)x = b \\ \iff & B(Cx) = b \\ \iff & By = b \quad \text{and} \quad Cx = y \end{aligned}$$

Thus, using the transformation $C \cdot \vec{x} = y$, we can solve equation 1 by solving two linear systems involving only simple matrices:

$$\begin{aligned} B \cdot \vec{y} &= \vec{b} \\ C \cdot \vec{x} &= \vec{y} \end{aligned}$$

As we saw earlier, the cost of solving a system with a $N \times N$ simple matrix of coefficients is only $O(N^2)$ - so once we have B and C, the total cost of solving the system $Ax = b$ is also $O(N^2)$. Unfortunately, the factorization of A needed to get B and C requires $O(N^3)$ operations, so the cost of factorization dominates the cost of the solution.

We will next discuss two prevailing factorization algorithms, LU factorization and QR factorization.

2.1 LU Factorization

As the name suggests, in LU factorization, we factor the matrix A into the product of a lower and upper triangular matrix:

$$A \quad = \quad L \cdot U$$

This process is identical to the familiar process of solving a linear system by elimination. For example, consider the following linear system:

$$\underbrace{\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}}_{A} \cdot \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{\vec{x}} \quad = \quad \underbrace{\begin{bmatrix} 5 \\ 6 \end{bmatrix}}_{\vec{b}}$$

We might solve this system by multiplying the first row of the matrix A by 3 and subtracting it from the third row, and making similar transformations to $\vec{b}$.

$$\begin{bmatrix} 1 & 2 \\ 0 & -2 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad = \quad \begin{bmatrix} 5 \\ -9 \end{bmatrix} \tag{2}$$

This is equivalent to solving the following system (arrived at by LU factorization):

$$\underbrace{\begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix}}_{B}\underbrace{\begin{bmatrix} 1 & 2 \\ 0 & -2 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{y} = \underbrace{\begin{bmatrix} 5 \\ 6 \end{bmatrix}}_{b}$$

Since this yields:

$$\begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$$
$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 5 \\ -9 \end{bmatrix}$$

we then solve the equation:

$$\begin{bmatrix} 1 & 2 \\ 0 & -2 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 5 \\ -9 \end{bmatrix} \tag{3}$$

to obtain our final values for $\vec{x}$. Note that equations 2 and 3 are identical. In general, LU factorization is equivalent to solving a system by elimination.

In Matlab, we can find the LU factorization of a matrix A via the command

```
[L,U,P] = lu(A)
```

This LU factorization command returns the lower and upper triangular matrices L and U, and also a permutation matrix P. This is due to a problem with the stability of the LU factorization method, which we will discuss later.

LU-factorization of an $N \times N$ matrix takes approximately $\frac{N^3}{6}$ operations.

2.2 QR Factorization

In QR factorization, we split the matrix A into an orthogonal matrix Q and an upper triangular matrix R:

$$A = Q \cdot R$$

Transforming the equation $A\vec{x} = \vec{b}$ into:

$$Q \cdot \vec{y} = \vec{b}$$
$$R \cdot \vec{x} = \vec{y}$$

In Matlab, we can find the QR factorization of a matrix A via the command

```
[Q,R] = qr(A)
```

QR factorization on an $N \times N$ matrix takes about $\frac{N^3}{3}$ operations, about twice as many as LR-factorization, however, there are situations in which it is a more appropriate choice.

3 Conditioning

In an ideal world, we would now be done. Unfortunately, in practice we will always need to solve systems on finite precision machines. Can it happen that things go really wrong when we use finite precision? The answer is that for some systems, things can go very wrong.

To find out whether things will go wrong for a particular linear system $Ax = b$, we need to study how susceptible that linear system is to small perturbations in the right-hand side. For example, given two linear systems using matrix A

$$\begin{aligned} A \cdot \vec{x} &= \vec{b} \\ A \cdot \vec{x'} &= \vec{b'}, \end{aligned}$$

we would like to know that small perturbations in the right-hand side of such linear equations produces small changes in solutions, that is:

$$\frac{\|x'\|}{\|x\|} \sim \frac{\|b'\|}{\|b\|}$$

What we would especially like to avoid is small changes in the right-hand side $\vec{b}$ resulting in large changes in the solution $\vec{x}$. The reason is as follows: when solving a linear system numerically, all data values in $\vec{b}$ are represented with finite precision and so will have some error (call it $\vec{b'}$). Thus, any linear system that we solve numerically can be represented as

$$A \cdot \left(\vec{x} + \vec{x'}\right) = \vec{b} + \vec{b'}$$

where $\vec{b'}$ is a measure of the error of $\vec{b}$ resulting from finite-precision. From the distributive law, we know that this is equivalent to solving the following two systems:

$$\begin{aligned} A \cdot \vec{x} &= \vec{b} \\ A \cdot \vec{x'} &= \vec{b'} \end{aligned}$$

If $\left\|\vec{x'}\right\|$ is very large in comparison to $\left\|\vec{b'}\right\|$, then the solution to the linear system is dominated by the error in the right hand side, so that the solution $\vec{x} + \vec{x'}$ returned by our numerical method will be very far from the actual solution $\vec{x}$. Similarly, if small changes in $\left\|\vec{x'}\right\|$ cause very large changes in $\left\|\vec{b'}\right\|$, then we will be unable to distinguish between numerical solutions $\vec{x}$ and $\vec{x} + \vec{x'}$ to very different linear systems.

On the other hand, if the solution $\|x'\|$ to $Ax' = b'$ is always roughly the same size as $\|b'\|$, then we can have confidence that even on a finite precision machine our solutions to $Ax = b$ will be accurate.

The property of a matrix A that we have been discussing above is called *conditioning*, and is captured quantitatively in the condition number of a matrix.

Definition 3.1 (Condition Number). *Given a norm $\|\ \|$ and an $N \times N$ matrix A the condition number of A is defined as:*

$$cond(A) = \frac{\max\{\|A \cdot \vec{v}\| : v \in \mathbb{C}^N, \|v\| = 1\}}{\min\{\|A \cdot \vec{v}\| : v \in \mathbb{C}^N, \|v\| = 1\}}$$

Intuitively, the condition number measures

$$cond(A) = \frac{\text{“how much } A \text{ can stretch a vector”}}{\text{“how much } A \text{ can shrink a vector”}}$$

So the condition number captures how “bad” the matrix is according to our discussion above.

For example, consider the matrix

$$A = \begin{bmatrix} 2 & -2 \\ 1 & -1+\epsilon \end{bmatrix}$$

Say we use the 1-norm in the definition of the condition number. Note that

$$\begin{aligned} \|A\|_1 &= \max\{\|Ax\|_1 : \|x\|_1 = 1\} \\ &\approx 3 \end{aligned}$$

On the other hand,

$$\begin{aligned} \min\{\|Ax\|_1 : \|x\|_1 = 1\} &\leq \left\| A \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix} \right\|_1 \\ &= \left\| \begin{bmatrix} 0 \\ \epsilon/2 \end{bmatrix} \right\|_1 \\ &= \epsilon/2 \end{aligned}$$

Thus

$$cond(A) \geq \frac{3}{\epsilon/2} = \frac{6}{\epsilon} \to \infty$$

as $\epsilon \downarrow 0$.

If the condition number $cond(A)$ is large, then due to the issues discussed above we might be in bad shape, and we should try to avoid any linear system $Ax = b$ involving A. Note that we are not discussing any particular algorithm - a bad condition number affects *any* possible algorithm implemented on a finite-precision machine.

A particular consequence is that even estimating the condition number itself on a finite-precision machine number may run into problems. If Matlab says that the condition number of a matrix is large, then we can be sure that it is large in reality. But if Matlab says the condition number is small, it may *still* truly be large - so we need to be careful.

3.1 Geometric Interpretation of Conditioning

To illustrate the idea of conditioning, consider the case of a 2×2 linear system:

$$\begin{aligned} a \cdot x(1) + b \cdot x(2) &= c \\ d \cdot x(1) + e \cdot x(2) &= f \end{aligned}$$

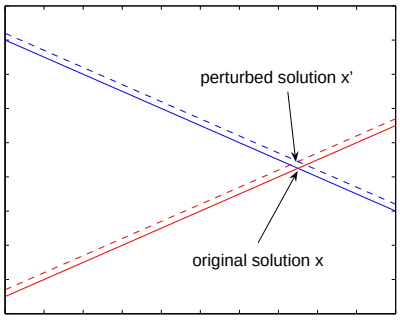

Figure 1: A Well-Conditioned Linear System

Each of these equations can be represented by a line in two-space, and a change to the right hand side will result in shifting the line in a direction perpendicular to its slope (essentially, shifting the line up or down). If this system is well-conditioned, the lines represented by the two equations have slopes that are noticeably different, so that small changes to the right hand sides will result in small changes in the solution, as shown in Figure 1.

Conversely, if the lines representing the equations have nearly identical slope, then small changes in the right hand side will result in enormous changes in the resulting solution, as shown in Figure 2.

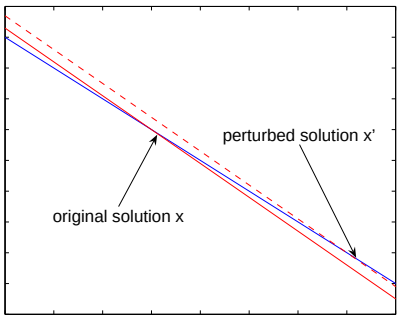

Figure 2: An Ill-Conditioned Linear System

An ill-conditioned linear system might have the form:

$$\underbrace{\begin{bmatrix} 1 & 1 \\ 1 & 1+\varepsilon \end{bmatrix}}_{A} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} ? \\ ? \end{bmatrix}$$

where ε is some very small value. If we examine the effect of matrix A on some normalized vectors

under the ∞-norm, we see that

$$\begin{aligned}
& \left\| \begin{bmatrix} 1 & 1 \\ 1 & 1+\varepsilon \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right\|_\infty \\
= & \left\| \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\| \\
= & 1 \\
& \left\| \begin{bmatrix} 1 & 1 \\ 1 & 1+\varepsilon \end{bmatrix} \cdot \begin{bmatrix} 1 \\ -1 \end{bmatrix} \right\|_\infty \\
= & \left\| \begin{bmatrix} 0 \\ -\varepsilon \end{bmatrix} \right\| \\
= & \varepsilon
\end{aligned}$$

Thus, if ε is very small, the condition number of A will be very large, and any corresponding linear system will be ill-conditioned. Looking at the definition of condition number, we can see that matrices can be ill-conditioned by mapping some vector into a vector with a very large norm, or into a vector with a very small norm. We have already seen that the former idea can be measured as the norm of the matrix.