References for Today’s Lecture

• Required reading
  – CLRS Chapter 28
Systems of Equations

• In some applications, we must determine values for a given set of unknowns, or variables, that satisfy one or more equations.

• Example:
Linear Equations

- A **linear equation** in $n$ variables $x_1, \ldots, x_n$ is an equation of the form

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n = b$$

where $a_1, a_2, \ldots, a_n$ and $b$ are constants.

- A **solution** to the equation is an assignment of values to the variables such that the equation is satisfied.

- Suppose we interpret the constants $a_1, a_2, \ldots, a_n$ as the entries of an $n$-dimensional vector $a$.

- Let’s also make a vector $x$ out of the variables $x_1, x_2, \ldots, x_n$.

- Then we can rewire the above equation as simply $a^T x = b$. 
Systems of Linear Equations

• Suppose we are given a set of \( n \) variables whose values must satisfy more than one equation.

• In this case, we have a \textit{system of equations}, such as

\[
\begin{align*}
    a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\
    a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\
    \vdots & \quad \vdots \\
    a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m
\end{align*}
\]

where \( a_{ij} \) is a constant for all \( 1 \leq i \leq m \) and \( 1 \leq j \leq n \) and \( b_1, \ldots, b_m \) are constants.

• As before, a solution to this system of equations is an assignment of values to the variables such that all equations are satisfied.

• Now we can interpret the constants \( a_{ij} \) as the entries of a \textit{matrix} \( A \) and the constants \( b_1, \ldots, b_m \) as the entries of a \textit{vector} \( b \).

• Interpreting the variables \( x_1, \ldots, x_n \) as a vector, we can again write the system of equation simply as \( Ax = b \).
Solving Systems of Linear Equations

- From linear algebra, we know that the system of equations $Ax = b$ has a unique solution if and only if the matrix $A$ is square and invertible.

- From now on, we will consider only such systems.

- How do we solve a systems of equations?
Special Matrices

• A square matrix $D$ is *diagonal* if $d_{ij} = 0$ whenever $i \neq j$.

• A square matrix $L$ is *lower triangular* if $l_{ij} = 0$ whenever $j > i$.

• A square matrix $U$ is *upper triangular* if $u_{ij} = 0$ whenever $j < i$.

• A square matrix $P$ is a *permutation matrix* if there is a single 1 in each row and column.

• The identity matrix, usually denoted $I$ is a diagonal matrix that is also a permutation matrix.

• What effect does multiplying by a permutation matrix have?
The LUP Decomposition

• Let’s suppose that we are able to find three \( n \times n \) matrices \( L, U, \) and \( P \) such that

\[ PA = LU \]

where

– \( L \) is upper triangular.
– \( U \) is lower triangular with 1’s on the diagonal.
– \( P \) is a permutation matrix.

• This is called an \textit{LUP decomposition} of \( A \).

• How could use such a decomposition to solve the system \( Ax = b \)?
Using the LUP Decomposition

• Once we have an LUP decomposition, we can use it to easily solve the system \( Ax = b \).

• Note that the system \( PAx = Pb \) is equivalent to the original system, which is then equivalent to \( LUx = Pb \).

• We can solve the system in two steps:
  
  – First solve the system \( Ly = Pb \) (forward substitution).
  – Then solve the system \( Ux = y \) (backward substitution).

• Note the similarity to Gaussian elimination.

• What is the running time of this solution method, once we know the factorization?
Finding the LU Decomposition

- Let’s assume for now that $P = I$ and concentrate on finding $L$ and $U$.
- We can find these two matrices using a procedure similar to Gaussian elimination.
- In fact, we will implement the algorithm recursively.
- First we’ll divide the matrix $A$ into four pieces, as follows:

\[
A = \begin{bmatrix}
  a_{11} & a_{12} & \cdots & a_{1n} \\
  a_{21} & a_{22} & \cdots & a_{2n} \\
  \vdots & \vdots & \ddots & \vdots \\
  a_{n1} & a_{n2} & \cdots & a_{nn}
\end{bmatrix}
\]

\[= \begin{bmatrix}
  a_{11} & w^T \\
  v & A'
\end{bmatrix}\]  

- Next, we’ll use use row operations to change $v$ into the zero vector and record the operations in another matrix.
Finding the LU Decomposition (cont.)

• Using the method on the previous slide, we can obtain the following factorization of $A$.

$$A = \begin{bmatrix} a_{11} & w^T \\ v & A' \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 \\ v/a_{11} & I \end{bmatrix} \begin{bmatrix} a_{11} & w^T \\ 0 & A' - vw^T/a_{11} \end{bmatrix}$$

(7)

(8)

• We can show that if $A$ is nonsingular, then so is $A' - vw^T/a_{11}$.

• So we can recursively call the method to factor the $(n - 1) \times (n - 1)$ matrix $A' - vw^T/a_{11}$.

• Applying this recursion $n$ times yields the desired factorization, as explained on the next slide.
Finding the LU Decomposition (cont.)

• To see how to get the factorization from the recursive application of the algorithm, we have the following.

\[
A = \begin{bmatrix}
1 & 0 \\
v/a_{11} & I
\end{bmatrix}
\begin{bmatrix}
a_{11} & w^T \\
0 & A' - vw^T/a_{11}
\end{bmatrix}
\] (9)

\[
= \begin{bmatrix}
1 & 0 \\
v/a_{11} & I
\end{bmatrix}
\begin{bmatrix}
a_{11} & w^T \\
0 & L'U'
\end{bmatrix}
\] (10)

\[
= \begin{bmatrix}
1 & 0 \\
v/a_{11} & L'
\end{bmatrix}
\begin{bmatrix}
a_{11} & w^T \\
0 & U'
\end{bmatrix}
\] (11)

• This shows how to obtain the factorization recursively.

• Notice that this can also be done iteratively and “in place.”
Finding the LUP Decomposition

- The element $a_{11}$ is called the *pivot element*.

- Note that the above decomposition method fails whenever the pivot element is zero.

- In this case, we can permute the rows of $A$ to obtain a new pivot element.

- In fact, for numerical stability, it is desirable to have the pivot element be as large as possible in absolute value.

- If no nonzero pivot is available, $A$ is singular.

- This leads to the following modified factorization.

\[
QA = \begin{bmatrix}
a_{k1} & w^T \\
v & A'
\end{bmatrix}
\]

\[
= \begin{bmatrix}
1 & 0 \\
v/a_{k1} & I
\end{bmatrix} \begin{bmatrix}
a_{k1} & w^T \\
0 & A' - vw^T/a_{k1}
\end{bmatrix}
\]
Finding the LUP Decomposition (cont.)

• Again, we can recursively call the method to factor the \((n - 1) \times (n - 1)\) matrix \(A' - vw^T/a_{11}\).

• As before, we obtain \(L'\), \(U'\), and \(P'\) and we get

\[
PA = \begin{bmatrix} 1 & 0 \\ 0 & P' \end{bmatrix} QA
\]

\[(14)\]

\[
= \begin{bmatrix} 1 & 0 \\ 0 & P' \end{bmatrix} \begin{bmatrix} 1 & 0 \\ v/a_{k1} & I \end{bmatrix} \begin{bmatrix} a_{k1} & w^T \\ 0 & A' - vw^T/a_{k1} \end{bmatrix}
\]

\[(15)\]

\[
= \begin{bmatrix} 1 & 0 \\ P'v/a_{k1} & I \end{bmatrix} \begin{bmatrix} a_{k1} & w^T \\ 0 & P'(A' - vw^T/a_{k1}) \end{bmatrix}
\]

\[(16)\]

\[
= \begin{bmatrix} 1 & 0 \\ P'v/a_{k1} & I \end{bmatrix} \begin{bmatrix} a_{k1} & w^T \\ 0 & L'U' \end{bmatrix}
\]

\[(17)\]

\[
= \begin{bmatrix} 1 & 0 \\ P'v/a_{k1} & L' \end{bmatrix} \begin{bmatrix} a_{k1} & w^T \\ 0 & U' \end{bmatrix}
\]

\[(18)\]

• What is the running time of finding the LUP decomposition?
Using the LUP Decomposition

- Note that finding the decomposition has the same running time as Gaussian elimination.
- The decomposition can be stored in almost the same space as the original matrix.
- Once we have an LUP decomposition, we can solve $Ax = b$ with various right hand sides in time $\Theta(n^2)$. 