Introduction to Numerical Analysis I
Handout 13

1 Numerical Linear Algebra

Consider linear system of equations $A_{n\times n}x_{n,1} = b_{n,1}$, where the matrix A can be dense or, when most of the entries are zeros, sparse. We distinct between direct methods, that provide an exact solution up to round-off error in a finite number of steps, and the iterative methods that can be characterized by a sequence of approximations that tends to the exact solution, that is $x_{n+1} = g(x_n) \to \bar{x}$

1.1 Direct Methods

1.1.1 Cramer's Rule

One of commonly (hand) used direct methods is the Cramer's rule that says $x_i = \frac{det(A_i)}{det(A)}$ where A_i is the matrix formed by replacing the i'th column of A by the column vector b. However this algorithm is very expensive since one need to compute $n+1$ determinants, which costs $n \cdot n!$ each, that is very ineffective - $n \cdot (n+1)!$ operations.

1.1.2 Gauss Elimination

For an upper triangular matrix which has $[A]_{i,i+p} = 0$ for $i < i+p \leqslant n$ the solution to $Ax = b$ is given by backward substitution

$$x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{k=i+1}^{n} a_{ik}x_k\right)$$

for $i = n, n-1, ..., 1$.

In the particular case of diagonal matrix, this gives $x_i = \frac{b_i}{a_{ii}}$ and for unit matrix I even simpler $x_i = b_i$.

For the general matrix which is not triangular or diagonal, one first use elementary operation to make it upper triangular and then do backward substitution.

Algorithm:

Initial Triangulation:

1. for each column $r_j : j = 1, ..., n-1$
2. for each row $r_i : i = j+1 \dots n$
3. $r_i \leftarrow r_i - \frac{a_{ij}}{a_{jj}} r_j$
4. $b_i \leftarrow b_i - \frac{a_{ij}}{a_{jj}} b_j$

Backward Substitution:

5. for each unknown $x_i : i = n, n-1, ..., 1$
6. $x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{k=i+1}^{n} a_{ik}x_k\right)$

In Initial Triangulation for each column we do $n-j$ operations of elimination which gives (as sum of arithmetic progression) $O(n^2)$ operations per row of n elements, that is $O(n^3)$ operation in total. The Backward Substitution is n row operations, thus takes $O(n^2)$.

However, the algorithm it is not defined for the matrices with zeros in diagonal. Furthermore, for the very small entries of the diagonal, $1/a_{ii}$ is huge, therefore future adding/subtracting to this number become negligible which increase the error. The solution to the problems described called a Partial Pivoting, that is, in the second statement of the algorithm the row r_i be choosen to have the largest a_{ij}, $i > j$. This improve the error.

Example 1.1. Let $0 \leq \varepsilon \leq 10^{-15}$. Consider the following system

$$\begin{bmatrix} \varepsilon & 1 \\ 1 & -\varepsilon \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1+\varepsilon \\ 1-\varepsilon \end{bmatrix}$$

Which has the solution $[x, y]^T = [1, 1]^T$.

Without pivoting we get the wrong answer

$$\left[\begin{array}{cc|c} \varepsilon & 1 & 1+\varepsilon \\ 1 & -\varepsilon & 1-\varepsilon \end{array}\right] \xrightarrow{r_2 \leftarrow r_2 - \frac{1}{\varepsilon}r_1} \left[\begin{array}{cc|c} \varepsilon & 1 & 1+\varepsilon \\ 0 & -\varepsilon - \frac{1}{\varepsilon} & 1-\varepsilon-\frac{1+\varepsilon}{\varepsilon} \end{array}\right]$$
$$\approx \left[\begin{array}{cc|c} \varepsilon & 1 & 1 \\ 0 & -\frac{1}{\varepsilon} & -\frac{1}{\varepsilon} \end{array}\right] \to \left[\begin{array}{cc|c} \varepsilon & 1 & 1 \\ 0 & 1 & 1 \end{array}\right] \to \left[\begin{array}{cc|c} 1 & 0 & 0 \\ 0 & 1 & 1 \end{array}\right]$$

With Pivoting we get the correct answer

$$\left[\begin{array}{cc|c} \varepsilon & 1 & 1+\varepsilon \\ 1 & -\varepsilon & 1-\varepsilon \end{array}\right] \approx \left[\begin{array}{cc|c} \varepsilon & 1 & 1 \\ 1 & -\varepsilon & 1 \end{array}\right] \xrightarrow{r_2 \leftrightarrow r_1} \left[\begin{array}{cc|c} 1 & -\varepsilon & 1 \\ \varepsilon & 1 & 1 \end{array}\right]$$
$$\xrightarrow{r_2 \leftarrow r_2 - \varepsilon r_1} \left[\begin{array}{cc|c} 1 & -\varepsilon & 1 \\ 0 & 1+\varepsilon^2 & 1-\varepsilon \end{array}\right] \approx \left[\begin{array}{cc|c} 1 & 0 & 1 \\ 0 & 1 & 1 \end{array}\right]$$

1.1.3 LU Decomposition

One writes Gauss Elimination operations as a a matrix multiplication. Before we get into details of the method, let start with some interesting characteristics. Denote L_n the n'th elimination operation, than the sequence of initial triangulation can be wren as $L_n \cdots L_2 L_1 A = U$, where U denotes the resulting upper triangular matrix. One denotes $L^{-1} = L_n \cdots L_2 L_1$, thus $A = LU = L_1^{-1}L_2^{-1}\dots L_n^{-1}U$. We will see it soon that the matrix L is lower triangular.

The idea of decomposition is as following: since $Ax = LUx = b$, denote $Ux = y$ then $Ly = b$, in other words $Ux = y = L^{-1}b$. One think about this idea as dividing the hard problem of $Ax = b$ into two simple problems (since L and U are triangular, that is forward/backward substitution $O(n^2)$ operations each):

$$\begin{cases} Ux = y \\ Ly = b \end{cases}$$

Still the decomposition itself require $O(n^3)$ operation. The advantage of the method is for the cases of multiple input: $\{A_{n,n}x_k = b_k\}_{k=1}^N$ when $N \cdot O(n^3) > O(n^3) + 2N \cdot O(n^2) \Rightarrow n > \frac{2N}{N-1} > 2$, that is almost every time.

We now define L and L^{-1} as following

$$(L_i)_{jj} = 1, (L_i)_{ij} = -\frac{A_{ij}}{A_{ii}}, \forall j > i$$

$$\text{and } \left(L_i^{-1}\right)_{jj} = 1, \left(L_i^{-1}\right)_{ij} = \frac{A_{ij}}{A_{ii}}, \forall j > i$$

other entries are zeros, for example

$$L_1 = \begin{bmatrix} 1 & & & \\ -\frac{a_{21}}{a_{11}} & 1 & & \\ \vdots & & \ddots & \\ -\frac{a_{n1}}{a_{11}} & & & 1 \end{bmatrix} \quad L_1^{-1} = \begin{bmatrix} 1 & & & \\ \frac{a_{21}}{a_{11}} & 1 & & \\ & & \ddots & \\ \frac{a_{n1}}{a_{11}} & & & 1 \end{bmatrix}$$

Example 1.2.

$$A = \begin{pmatrix} \frac{1}{2} & 1 & \frac{3}{2} \\ 2 & 5 & 5 \\ 7 & 8 & 9 \end{pmatrix} \quad L_1 = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{2}{1/2} & 1 & 0 \\ -\frac{7}{1/2} & 0 & 1 \end{pmatrix} \quad L_1 A = \begin{pmatrix} \frac{1}{2} & 1 & \frac{3}{2} \\ 0 & 1 & -1 \\ 0 & -6 & -12 \end{pmatrix}$$

Note that if we change in previous example $A_{22} = 4$ this won't affect the L_1 but now $(L_1 A)_{2\to} = \begin{pmatrix} 0 & 0 & -1 \end{pmatrix}$ which won't allow us to continue. This is similar problem to what we saw in gaussian elimination and the solution is pivoting. Fortunately, pivoting may also be written in the form of matrix multiplication.

Denote P_n the permutation matrix at n'th step of the algorithm. The permutation matrix that exchange between rows i and j is created by exchanging these rows in the unit matrix I. That is

$$P = P^{i\leftrightarrow j}\left[e_1, \cdots, e_{i-1}, e_j, e_{i+1}, \cdots, e_{j-1}, e_i, e_{j+1}, \cdots, e_n\right]^T$$

where $e_j = (0, ..., 1, ..., 0)^T$ with the 1 at the place j. The nice thing is that $P = P^{-1}$ (why?) For example

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} = I$$

For the general case of LU-decomposition with pivoting one writes $L_n P_n \cdots L_2 P_2 L_1 P_1 A = U$. To get the $PA = LU$ form define

$$\begin{aligned} \tilde{L}_n &= L_n, \\ \tilde{L}_{n-1} &= P_n L_{n-1} P_n^{-1}, \\ &\vdots \\ \tilde{L}_1 &= P_n P_{n-1} \cdots P_2 L_1 P_2^{-1} \cdots P_{n-1}^{-1} P_n^{-1} \end{aligned}$$

and therefore $\tilde{L}_n \cdots \tilde{L}_1 P_n \cdots P_1 = L^{-1} P A = U$. The following two properties holds:

1. $\tilde{L}_j$ is lower triangular since $P_k L_j P_k$ for $k > j$
2. $\tilde{L}_n \cdots \tilde{L}_1 P_n \cdots P_1 A = L_n P_n \cdots L_2 P_2 L_1 P_1 A = U$

Example 1.3. Let $L_3 P_3 L_2 P_2 L_1 P_1 A = U$, then

$$\begin{aligned} \tilde{L}_3 &= L_3, \\ \tilde{L}_2 &= P_3 L_2 P_3^{-1}, \\ \tilde{L}_1 &= P_3 P_2 L_1 P_2^{-1} P_3^{-1} \end{aligned}$$

and therefore

$$\tilde{L}_3 \tilde{L}_2 \tilde{L}_1 P_3 P_2 P_1 = L_3 P_3 L_2 P_3^{-1} P_3 P_2 L_1 P_2^{-1} P_3^{-1} P_3 P_2 P_1 = \\ = L_3 P_3 L_2 P_2 L_1 P_1$$

1.1.4 QR decomposition

Yet another decomposition is into $A_{n\times n} = QR$ where Q is an Orthonormal Matrix (which gives $QQ^T = I$) and R is an Right/Upper Triangular Matrix. One may write it as system of equation

$$\begin{cases} Rx = y \\ Qy = b \end{cases}$$

or evan as $Rx = Q^T b$

The orthogonal matrix $Q = [q_1, \ldots, q_n]$ is a orthogonal basis of the column space (also called a range) of the matrix A. Note that the columns of $A = [a_1, \ldots, a_n]$ is also a basis of it's range, therefore in order to obtain Q one uses the Gram-Schmidt process.

The matrix R is given by $R = Q^T A$. Since R is upper triangular, one obtain it using inner product of columns of Q with columns of A as following: $(R)_{ij} = (q_i, a_j)$ for $i \geq j$, while for $i < j$ set $(R)_{ij} = 0$, that is

$$R = \begin{bmatrix} (q_1, a_1) & (q_1, a_2) & (q_1, a_3) & \cdots \\ 0 & (q_2, a_2) & (q_2, a_3) & \cdots \\ 0 & 0 & (q_3, a_3) & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

There is a problem with the numerical stability of the Gram-Schmidt: due to the round-off error the vectors aren't really orthogonal. In order to improve the stability one uses the Modified-Gram-Schmidt which we describe below. However, there is also two more stable algorithms (which we won't learn), the Householder Transformation and the Givens Rotations, which gives similar results.

Theorem 1.4 (Modified Gram-Schmidt Process). Let $\{b_j\}_{j=0}^n$ be some basis for a vector space V. The orthogonal basis $\{v_j\}_{j=0}^n$ is defined algorithmically by

$v_k = b_k$
for j=1 to k-1
$\quad v_k = v_k - \frac{v_j}{(v_j, v_j)}(v_k, v_j)$
end

Rectangular matrices Let $A_{m\times n}$ be $m \times n$ matrix where $m \geq n$, then $A = QR = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_1 \\ 0 \end{bmatrix} = Q_1 R_1$.

Example 1.5.

$$A = \begin{bmatrix} 3 & -6 \\ 4 & -8 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 3/5 & 0 \\ 4/5 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & -10 \\ 0 & 1 \end{bmatrix}$$