Report on Numerical Linear Algebra Coding Assignment

Chapter 1: Direct Methods for Linear Systems

Hongyi Zhang, 1500017736

October 4, 2017

1 Solving General Linear Systems

1.1 Problem background

Write programs for Gaussian elimination with no pivoting and with partial pivoting, then run them for the following linear system of order 84,

$$\begin{bmatrix} 6 & 1 & & & & \\ 8 & 6 & 1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & 8 & 6 & 1 \\ & & & 8 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{83} \\ x_{84} \end{bmatrix} = \begin{bmatrix} 7 \\ 15 \\ \vdots \\ 15 \\ 14 \end{bmatrix}. \tag{*}$$

Compare the numerical results with the analytic solution, and illustrate your view on Gaussian elimination based on the comparison.

1.2 LU decomposition with Gaussian elimination

The motivation for LU decomposition arises from the computational convenience of triangular systems and the desire for reutilization of these triangular factors. For a linear system $Ly = b$ where $L \in \mathbb{R}^{n\times n}$ lower triangular, the solution can be easily derived as follows, which only takes n^2 flops.

$$\begin{aligned} &y_1 = b_1/\ell_{11}, \\ &y_i = \left(b_i - \sum_{j=1}^{i-1} \ell_{ij} y_j\right) \Big/ \ell_{ii}, \quad (2 \le i \le n), \end{aligned}$$

Similarly, an upper triangular system $Ux = y$ can be solved with n^2 flops as well. So now we would like to factorize $A = LU$ where L lower triangular and U upper triangular to take advantage of these triangular systems.

A Gaussian transformation L_k at step k eliminates $A(k+1:n,k)$ by subtracting a multiple of the k-th row from each row below the k-th. Each L_k can be written as $L_k = I - l_k e_k^{\mathrm{T}}$, where $l_{k,i} = a_{ik}/a_{kk}$ for each $i > k$. By applying $n-1$ such transformations (if possible), we obtain an upper triangular matrix

$$U = L_{n-1}L_{n-2}\cdots L_1 A.$$

Let $L = (L_{n-1}L_{n-2}\cdots L_1)^{-1} = I + l_1 e_1^{\mathrm{T}} + \cdots + l_{n-1}e_{n-1}^{\mathrm{T}}$, which is apparently lower triangular, then $A = LU$ is the desired decomposition.

In practice, it would be desirable for each a_{kk} to have large absolute value when performing elimination, so as to avoid huge absolute error resulted from division by a small number. With such attempt, partial pivoting technique at step k selects p such that

$$|a_{pk}| = \max\{|a_{ik}| : \ k+1 \le i \le n\},$$

then switch the k-th and p-th row and proceed with elimination. This selection step takes (acceptably) extra computational effort, but great improvements on numerical stability can be seen.

1.3 Numerical results

For equation (*), its accurate solution is

$$x = (1, 1, \cdots, 1)^{\mathrm{T}}.$$

Running LU decomposition with no pivoting (LU) and with partial pivoting (PLU) for the equation (*) gives the result in Table 1. LU method displays intolerable absolute error in L^∞ norm (defined as $\|x^{sol} - x\|_\infty$). Further introspection shows that the k-th entry of the LU solution gets only 5 significant digits correct when $k \le 40$, and no digits correct when $k > 50$.

Table 1: Comparison of LU and PLU performance

	L^∞ error	Avg. runtime
LU	5.368e+08	0.00643 sec
PLU	2.797e−06	0.00934 sec

In contrast, PLU solution achieves remarkable accuracy, with all entries at least 6 digits correct. This strongly indicates that partial pivoting helps a great deal in improving numerical stability, despite a bit longer time taken, as is shown in the table, to select the pivot in the columns.

Therefore, being a mathematically perfect algorithm, Gaussian elimination does not always function properly in reality due to various errors from data collected to floating-point representation. Practical implementations must take them into account, try to be invulnerable to errors, and keep a good balance between computational cost and numerical accuracy.

2 Solving Symmetric Positive Definite Linear Systems

2.1 Problem background

Write programs for Cholesky decomposition and $\mathrm{LDL}^{\mathrm{T}}$ decomposition, then run them for the symmetric positive definite system $Ax = b$, where

(1) b is chosen at random, while A is a square matrix of order 100

$$\begin{bmatrix} 10 & 1 & & & \\ 1 & 10 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 10 & 1 \\ & & & 1 & 10 \end{bmatrix};$$

(2) A is the Hilbert matrix of order 40, whose entries are given by

$$a_{ij} = \frac{1}{i+j-1},$$

while b is the row sum of A, namely

$$b_i = \sum_{j=1}^{n} \frac{1}{i+j-1}.$$

2.2 Cholesky decomposition and $\mathbf{LDL^T}$ decomposition

Suppose the coefficient matrix $A \in \mathbb{R}^{n\times n}$ is symmetric positive definite, then there exists a unique lower triangular matrix $L \in \mathbb{R}^{n\times n}$ with positive diagonal entries satisfying

$$A = LL^{\mathrm{T}}.$$

This is known as Cholesky decomposition and may be seen as a special case of LU decomposition, but it can be computed in a faster way, roughly twice as efficient as LU.

Assume $A = LL^{\mathrm{T}}$, where $L = (\ell_{ij})$ lower triangular. Element-wise comparison between two sides of the equality gives the following direct algorithm for computing L.

$$\ell_{kk} = \left(a_{kk} - \sum_{j=1}^{k-1} \ell_{kj}^2 \right)^{1/2},$$

$$\ell_{ik} = \left(a_{ik} - \sum_{j=1}^{k-1} \ell_{ij}\ell_{kj} \right) \Big/ \ell_{kk}, \quad (k+1 \le i \le n),$$

The process is numerically stable since $|\ell_{ik}| \le \sqrt{a_{ii}}$, and it only takes $n^3/3$ flops. But subtle errors

can arise from taking square roots. For this reason, $\mathrm{LDL}^{\mathrm{T}}$ decomposition has shown up, which factorizes the matrix into

$$A = LDL^{\mathrm{T}}$$

where L unit lower triangular and $D = \mathrm{diag}(d_1, d_2, \cdots, d_n)$ positive diagonal. Another element-wise comparison gives the algorithm for computing L and D as follows.

$$\begin{aligned} v_k &= d_k \ell_{jk}, && (1 \le k \le j-1), \\ d_j &= a_{jj} - \sum_{k=1}^{j-1} \ell_{jk} v_k, \\ \ell_{ij} &= \left(a_{ij} - \sum_{k=1}^{j-1} \ell_{ik} v_k \right) \Big/ d_j, && (j+1 \le i \le n). \end{aligned}$$

2.3 Numerical results: well-conditioned case

For the equation in subproblem (1), we first pick the entries of vector b uniformly from the interval $[-100, 100]$, and then compare the result of solving it with Cholesky and $\mathrm{LDL}^{\mathrm{T}}$ decompositions, which is shown in Table 2.

Table 2: Comparison of Cholesky and $\mathrm{LDL}^{\mathrm{T}}$ performance

	L^∞ error	Avg. runtime
Cholesky	2.664e−15	0.02688 sec
$\mathrm{LDL}^{\mathrm{T}}$	3.553e−15	0.00448 sec

No significant difference is found in the absolute error of the numerical solution (assuming the Matlab built-in solver to be accurate). But noticeably, Cholesky runs much longer than $\mathrm{LDL}^{\mathrm{T}}$. A reasonable explanation could be that the operation of taking square roots in Cholesky require a lot many flops, and hence the square-root-free $\mathrm{LDL}^{\mathrm{T}}$ enjoys greater computational efficiency.

2.4 Numerical results: ill-conditioned case

For the equation in subproblem (2), we have made several attempts to solve the equation, including LU, PLU, Cholesky, $\mathrm{LDL}^{\mathrm{T}}$ and the Matlab built-in solver. The results are displayed in Table 3. The analytic solution is apparently

$$x = (1, 1, \cdots, 1)^{\mathrm{T}},$$

but all the numerical methods fail to reach close enough to this solution due to the ill-conditioned coefficient matrix.

Table 3: Comparison of various methods on Hilbert matrix

	L^∞ error	Avg. runtime
LU	195.959	0.00212 sec
PLU	376.105	0.00410 sec
Cholesky	43018000.193	0.01633 sec
LDL^T	67.534	0.00118 sec
Matlab	95.030	0.00051 sec

Cholesky produces unexpectedly terrible results, with the largest absolute error and the longest runtime. Further investigation shows that Cholesky has taken the square root of a negative number during the process, which should mathematically never happen but in reality might be caused by subtle floating point error, and hence leads to a complex matrix complicating the computation afterwards. As a matter of fact, LDL^T suffers from the same problem as well, i.e. negative diagonal entries in D, but is partly resolved by avoiding taking square roots of them.

In terms of L^∞ error, interestingly, LDL^T outperforms all the other methods even the mature Matlab solver, although the solution is still not acceptable. In terms of runtime, the results fit the theoretical predictiosn quite well — PLU takes a bit longer in pivoting, and LDL^T has roughly half the cost of LU decomposition. But anyhow, it still remains a headache for us to handle such ill-conditioned linear systems.