Online Lecture Notes

Prof. Boris Houska

April 26, 2022

1 Numerical Methods for Linear Equation Systems

In the last lecture we have learned about Gauss elimination, which proceeds by computing upper- and lower triangular matrices R and L associated to a given invertible square matrix A such that

$$A = LR$$

This method can be refined for sparse matrices A, too, but exploiting the zeros in the coefficient scheme explictly (also we may permute rows and columns of A appropriately). A prototype example for a sparse LR-decomposition algorithm is obtained by having a closer look at tri-diagonal matrices of the form

$$A = \begin{pmatrix} a_1 & b_1 & & \\ c_1 & a_2 & \ddots & \\ & \ddots & \ddots & b_{n-1} \\ & & c_{n-1} & a_n \end{pmatrix} \in \mathbb{R}^{n \times n} .$$

All the "empty spaces" are filled with zeros that do not have to be stored explicitly. The main observation now is that L and R inherit the sparsity of A (in this case directly). In detail, this means that L and R should be matrices of the form

$$L = \begin{pmatrix} 1 & & & \\ \gamma_1 & 1 & & \\ & \ddots & \ddots & \\ & & \gamma_{n-1} & 1 \end{pmatrix} \qquad \text{and} \qquad R = \begin{pmatrix} \alpha_1 & b_1 & & \\ & \alpha_2 & \ddots & \\ & & \ddots & b_{n-1} \\ & & & \alpha_n \end{pmatrix}$$

Notice that we can work out a recursion for computing the coefficients α_i and γ_i. Essentially, this is in complete analogy to the dense Gauss elimination with the only difference being that we exploit the particular sparsity pattern of A.

For instance, in order to see we can write

$$
\begin{aligned}
R = L^{-1}A \quad &= \begin{pmatrix} 1 & & & \\ \gamma_1 & 1 & & \\ & \ddots & \ddots & \\ & & \gamma_{n-1} & 1 \end{pmatrix}^{-1} \begin{pmatrix} a_1 & b_1 & & \\ c_1 & a_2 & \ddots & \\ & \ddots & \ddots & b_{n-1} \\ & & c_{n-1} & a_n \end{pmatrix} \\
&= \begin{pmatrix} 1 & & & \\ -\gamma_1 & 1 & & \\ & \ddots & \ddots & \\ & & -\gamma_{n-1} & 1 \end{pmatrix} \begin{pmatrix} a_1 & b_1 & & \\ c_1 & a_2 & \ddots & \\ & \ddots & \ddots & b_{n-1} \\ & & c_{n-1} & a_n \end{pmatrix} \qquad (1) \\
&= \begin{pmatrix} a_1 & b_1 & & \\ c_1 - \gamma_1 a_1 & a_2 - \gamma_1 b_1 & \ddots & \\ & \ddots & \ddots & b_{n-1} \\ & & c_{n-1} - \gamma_{n-1} a_{n-1} & a_n - \gamma_{n-1} b_{n-1} \end{pmatrix}
\end{aligned}
$$

Since we want to ensure that this matrix is upper triangular, we need to choose the coefficients γ_i such that

$$\forall i \in \{1, \ldots, n-1\}, \qquad c_i - \gamma_i a_i = 0 \qquad \Longrightarrow \qquad \gamma_i = \frac{c_i}{a_i}$$

Additionally, we find that $\alpha_1 = a_1$ and $\alpha_i = a_i - \gamma_{i-1} b_{i-1}$ for $i \in \{2, \ldots, n\}$. With this we have found a way to compute a sparse LR-decmposition of the tri-diagonal matrix A with computational complexity

$$O(n) \ .$$

Notice that this much faster than using a dense LR-decomposition solver, which would have run-time $O(n^3)$.

Summary: in practice it is very important to exploit the sparsity pattern of the matrix A when implementing an LR-decomposition, since exploiting the pattern of A can improve the run-time by orders of magnitude! This is especially important when n is large!

1.1 QR-Decomposition

Recall that if the matrix is given columnwise, $A = (a_1, a_2, a_3, \ldots, a_n) \in \mathbb{R}^{n \times n}$, we can run a Gram-Schmidt algorithm to find an orthogonormal basis if the span of the vectors $a_1, a_2, a_3, \ldots, a_n$. The corresponding orthonormal vectors $q_1, q_2, \ldots, q_n$ are constructed recursively by the Gram-Schmidt algorithm such that

$$
\begin{aligned}
a_1 &= (a_1^\intercal q_1) q_1 && (2) \\
a_2 &= (a_2^\intercal q_1) q_1 + (a_2^\intercal q_2) q_2 && (3) \\
&\vdots && (4) \\
a_n &= (a_2^\intercal q_1) q_1 + (a_2^\intercal q_2) q_2 + \ldots + (a_n^\intercal q_n) q_n && (5)
\end{aligned}
$$

Notice that the coefficients $r_{i,j} = a_i^\intercal q_j$ are computed by the Gram-Schmidt recursion, too. This means that we can write the above equations in the form

$$
\begin{aligned}
a_1 &= r_{1,1} q_1 && (6)\\
a_2 &= r_{1,2} q_1 + r_{2,2} q_2 && (7)\\
&\vdots && (8)\\
a_n &= r_{1,n} q_1 + r_{2,n} q_2 + \ldots + r_{n,n} q_n && (9)
\end{aligned}
$$

Notice that we can write these equation more elegantly in matrix form

$$
\underbrace{(a_1, a_2, \ldots, a_n)}_{=A} = \underbrace{(q_1, q_2, \ldots, q_n)}_{=Q} \underbrace{\begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,n} \\ & r_{2,2} & \cdots & r_{2,n} \\ & & \ddots & \vdots \\ & & & r_{n,n} \end{pmatrix}}_{=R}
$$

Thus, in summary, the Gram-Schmidt algorithm automatically generates a QR-decomposition of the form

$$
A = QR \ ,
$$

where R is an upper triangular matrix and Q is an ortogonal matrix, since

$$
Q^\intercal Q = (q_1, q_2, \ldots, q_n)^\intercal (q_1, q_2, \ldots, q_n) = \begin{pmatrix} q_1^\intercal q_1 & q_1^\intercal q_2 & \cdots & q_1^\intercal q_n \\ q_2^\intercal q_1 & q_2^\intercal q_2 & \cdots & q_2^\intercal q_n \\ \vdots & \vdots & \ddots & \vdots \\ q_n^\intercal q_1 & q_n^\intercal q_2 & \cdots & q_n^\intercal q_n \end{pmatrix} = I \ .
$$

The corresponding linear equation system of the form

$$
Ax = b
$$

can then be solved by first storing the QR-decomposition of A and then using that

$$
Ax = b \quad \Longrightarrow \quad QRx = b \quad \Longrightarrow \quad Q^\intercal QRx = Q^\intercal b \quad \Longrightarrow \quad Rx = Q^\intercal b
$$

where the latter equation can be solved by backward substitution in $O(n^2)$ operations, since R is upper triangular matrix.

Summary: the QR decomposition is in many ways very similar to the LR decomposition, but one difference is that QR decomposition can also be computed for degenerate (as well as non-square) matrices A, since the Gram-Schmidt algorithm is completely generic—we can always detect an orthonognal basis of the span of the columns of A no matter what the dimensional of this span is! This means that in practice QR decompositions are eventually a bit more well-conditioned than LR decompositions, since they can be used on degerate matrices, too.

1.2 Cholesky Decomposition

Recall first that a matrix A is called symmetric positive definite if

$$A = A^\mathsf{T} \qquad \text{and} \qquad \forall v \in \mathbb{R}^n, \quad v^\mathsf{T} A v > 0 \; .$$

Also recall that the eigenvalues of a symmetric matrix are real-valued and that A is positive definite if all eigenvalues of A are strictly positive. The goal of the following considerations is to develop efficient algorithms for finding a Cholesky decomposition of the form

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

with L being a lower triangular matrix with ones on the diagonal and D a diagonal matrix. Notice that such a decomposition exists, since we could also start with a LR-decomposition of A,

$$A = LR$$

and then denote by D the part of R. Then we must have $R = DL^\mathsf{T}$ due to symmetry. But alternatively, we can find L and D by a direct comparison of coefficients, which leads to Cholesky's algorithm. Let us work this out by setting

$$L = \begin{pmatrix} 1 & & & \\ L_{2,1} & 1 & & \\ \vdots & & \ddots & \\ L_{n,1} & \dots & L_{n,n-1} & 1 \end{pmatrix} \quad \text{and} \quad D = \begin{pmatrix} D_1 & & & \\ & D_2 & & \\ & & \ddots & \\ & & & D_n \end{pmatrix} .$$

This yields

$$\begin{aligned} LDL^\mathsf{T} &= \begin{pmatrix} 1 & & & \\ L_{2,1} & 1 & & \\ \vdots & & \ddots & \\ L_{n,1} & \dots & L_{n,n-1} & 1 \end{pmatrix} \begin{pmatrix} D_1 & & & \\ & D_2 & & \\ & & \ddots & \\ & & & D_n \end{pmatrix} \begin{pmatrix} 1 & & & \\ L_{2,1} & 1 & & \\ \vdots & & \ddots & \\ L_{n,1} & \dots & L_{n,n-1} & 1 \end{pmatrix}^\mathsf{T} \\ &= \begin{pmatrix} D_1 & & & \\ D_1 L_{2,1} & D_2 & & \\ \vdots & & \ddots & \\ D_1 L_{n,1} & \dots & D_{n-1} L_{n,n-1} & D_n \end{pmatrix} \begin{pmatrix} 1 & L_{2,1} & \dots & L_{n,1} \\ & 1 & & \\ & & \ddots & L_{n,n-1} \\ & & & 1 \end{pmatrix} \\ &= \begin{pmatrix} D_1 & DL_{2,1} & & \\ D_1 L_{2,1} & D_1 L_{2,1}^2 + D_2 & & \text{sym} \\ \vdots & & \ddots & \\ D_1 L_{n,1} & \dots & \dots & D_1 L_{n,1}^2 + D_2 L_{n,2}^{@} + \dots + D_{n-1} L_{n,n-1}^2 + D_n \end{pmatrix} \end{aligned}$$

We can compare the coefficients of this matrix with the coefficients of A. This leads to the equations

$$\begin{aligned}
A_{1,1} &= D_1 && (10)\\
A_{2,1} &= D_1 L_{1,2} && (11)\\
A_{2,2} &= D_1 L_{2,1}^2 + D_2 && (12)\\
&\vdots && (13)
\end{aligned}$$

Notice that this equation system can be solved recursively with repsect to the coefficients of D and L. This yields

$$\begin{aligned}
D_1 &= A_{1,1} && (14)\\
L_{1,2} &= \frac{A_{2,1}}{D_1} && (15)\\
D_2 &= A_{2,2} - D_1 L_{2,1}^2 && (16)\\
&\vdots && (17)
\end{aligned}$$

This leads to the general recursion

$$D_j = A_{j,j} - \sum_{k=1}^{j-1} D_k L_{j,k}^2 \quad \text{and} \quad L_{i,j} = \frac{1}{D_j}\left(A_{i,j} - \sum_{k=1}^{j-1} L_{i,k} L_{j,k} D_k\right)$$

which can be evaluated in $O(n^3)$ operations, but the interesting aspect is that the symmetry of the matrix A is exploited explicitly by the recursion.

1.3 Summary

In this lecture we have discussed three types of numerical algorithm for solving linear equation systems:

1. LR-decomposition (Gauss elimination): applicable to invertible square matrices
2. QR-decomposition (Gram-Schmidt algorithm): applicable to any matrix
3. $LDL^\intercal$-decomposition (Cholesky algorithm): applicable to symmetric positive definite matrices.