CHAPTER 1: LINEAR SYSTEMS

1. Linear systems in statistics

Suppose that $X \in \mathbb{R}^{m\times m}$ is non-singular and write $A = X^{-1}$. The expression $X^{-1}b$ for some $b \in \mathbb{R}^m$, can be viewed as the solution of the linear system $Xu = b$. In other words, $X^{-1}b$ is the vector of coefficients that allows us to write b as linear combination of the columns of X. This interpretation prevails in computational linear algebra. That is, computing $X^{-1}b$ should always be viewed as solving the linear system $Xu = b$, not as computing X^{-1} and then multiplying it by b.

More generally the expression $X^{-1}B$ (for some $B \in \mathbb{R}^{m\times q}$) should be viewed as solving the linear system $XU = B$ ($Xu_j = b_j$, for $j = 1, \ldots, q$). In particular, and since $X^{-1} = X^{-1}I_m$, computing X^{-1} is equivalent to solving the linear system $XU = I_m$). Hence in general, computing X^{-1} itself is more expensive that computing $X^{-1}b$ (that is solving the linear system $Xu = b$).

There are several problems in statistics where these computations are needed.

1.1. **Gaussian density.** Let $\Sigma \in \mathbb{R}^{p\times p}$ be a spd matrix. Consider the problem of computing the lof-density of $\mathsf{N}_p(0, \Sigma)$. We have

$$\log f_\Sigma(x) = -\frac{p}{2}\log(2\pi) - \frac{1}{2}x'\Sigma^{-1}x - \frac{1}{2}\log\det\Sigma.$$

The computation of $x'\Sigma^{-1}x$ entails solving the linear system $\Sigma z = x$, and returning $z'x$.

1.2. **Graphical Lasso.** Let $X_1, \ldots, X_n \overset{i.i.d.}{\sim} \mathbf{N}(0, \theta^{-1})$ for some precision matrix $\theta \in \mathbb{R}^{p\times p}$ that we wish to estimate. The log-likelihood function is given by

$$\ell(\theta) = -\frac{np}{2}\log(2\pi) + \frac{n}{2}\log\det(\theta) - \frac{1}{2}\sum_{i=1}^n X_i'\theta X_i = -\frac{np}{2}\log(2\pi) + \frac{n}{2}\log\det(\theta) - \frac{n}{2}\mathsf{Tr}(\theta S),$$

where $S = \frac{1}{n}\sum_{i=1}^n X_iX_i' = \frac{1}{n}(XX')$ is the sample covariance matrix, where $X = [X_1, \ldots, X_n] \in \mathbb{R}^{p\times n}$. The maximum likelihood estimate of θ is

$$\hat\theta_{\mathsf{mle}} = \mathsf{Argmin}_{\theta\succ 0}\ \left[-\log\det(\theta) + \mathsf{Tr}(\theta S)\right].$$

If S is spd, it can be shown that $\hat\theta_{\mathsf{mle}} = S^{-1}$. But if S is not spd $\hat\theta_{\mathsf{mle}}$ does not exist in general. In fact when $p > n$, S is typically not spd, and one typically turn to regularized estimators:

$$\hat\theta_{\mathsf{reg}} = \mathsf{Argmin}_{\theta\succ 0}\ \left[-\log\det(\theta) + \mathsf{Tr}(\theta S) + \lambda\mathsf{Reg}(\theta)\right],$$

for a regularization function $\mathsf{Reg}(\theta)$. If θ is assumed sparse the graphical lasso regularization is commonly used where

$$\mathsf{Reg}(\theta) = \sum_{i,j}\left[\alpha|\theta_{ij}| + (1-\alpha)\frac{\theta_{ij}^2}{2}\right].$$

For this choice of regularization, one can show that $\hat\theta_{\mathsf{reg}}$ exists and can be computed using the following iterations (for some step-size $\gamma > 0$)

$$\theta_{k+1} = \mathsf{Prox}_{\gamma\lambda}\left(\theta_k - \gamma\left(S - \theta_k^{-1}\right)\right), \qquad (1)$$

where for a matrix $M \in \mathbb{R}^{p\times p}$, and $c > 0$, $\mathsf{Prox}_c(M) \in \mathbb{R}^{p\times p}$ is the matrix obtained obtained as

$$(\mathsf{Prox}_c(M))_{ij} = \begin{cases} \frac{M_{ij}-\alpha c}{1+(1-\alpha)c} & \text{if } M_{ij} > \alpha c \\ \frac{M_{ij}+\alpha c}{1+(1-\alpha)c} & \text{if } M_{ij} < -\alpha c \\ 0 & \text{otherwise} \end{cases}$$

For large values of p, the gradient algorithm described in (1) is virtually the most efficient algorithm for computing $\hat{\theta}_{\mathsf{reg}}$. Hence the ability to solve the graphical lasso problem for large p hinges on the ability to invert large spd matrices.

1.3. **Spatial regression.** Suppose we have data $y = (y_1, \ldots, y_n)$, where y_i is supposed to be observed at location $s_i \in \mathbb{R}^2$. We assume that $y \sim \mathsf{N}(0, C_\theta)$, where $C_\theta = \sigma^2 I_n + K_\theta$, where

$$(K_\theta)_{ij} = \exp\left(-\frac{\|s_i - s_j\|}{\theta}\right).$$

This is a simplified version of regression models commonly used in spatial data analysis. The log-likelihood of θ is given by

$$\ell(\theta) = \mathsf{const} - \frac{1}{2}\log\det C_\theta - \frac{1}{2}y' C_\theta^{-1} y, \quad \theta > 0.$$

Suppose that we want to compute the maximum likelihood estimate of θ. Getting the derivative of ℓ is key in this task. We have

$$\ell'(\theta) = -\frac{1}{2}\mathsf{Tr}\left[C_\theta^{-1}\left(\frac{\mathrm{d}C_\theta}{\mathrm{d}\theta}\right)\right] + \frac{1}{2}y' C_\theta^{-1}\left(\frac{\mathrm{d}C_\theta}{\mathrm{d}\theta}\right) C_\theta^{-1} y, \tag{2}$$

where $(\frac{\mathrm{d}C_\theta}{\mathrm{d}\theta})_{ij} = \frac{\mathrm{d}(K_\theta)_{ij}}{\mathrm{d}\theta}$.

Remark 1. This differentiation of ℓ can be derived as follows. If C_0, C are spd, and $\Delta = C - C_0$, then

$$\log\det C = \log\det(C_0 + \Delta) = \log\det(C_0(I_n + C_0^{-1}\Delta)) = \log\det(C_0) + \log\det(I_n + C_0^{-1}\Delta).$$

If $\lambda_1, \ldots, \lambda_n$ denote the eigenvalues of $C_0^{-1}\Delta$, then

$$\log\det(I_n + C_0^{-1}\Delta) = \sum_{j=1}^n \log(1+\lambda_j) = \sum_{j=1}^n \lambda_j + O(\sum_{j=1}^n \lambda_j^2) = \mathsf{Tr}(C_0^{-1}\Delta) + O(\|\Delta\|_{\mathsf{F}}^2).$$

Hence

$$\log\det(C_0 + \Delta) = \log\det C_0 + \left\langle C_0^{-1}, \Delta\right\rangle + O(\|\Delta\|_{\mathsf{F}}^2),$$

where $\langle\cdot,\cdot\rangle$ is the Frobenius inner product. We use this to conclude that

$$\begin{aligned} \log\det C_{\theta+\delta} &= \log\det C_\theta + \mathsf{Tr}(C_\theta^{-1}(C_{\theta+\delta} - C_\theta)) + O(\delta^2) \\ &= \log\det C_\theta + \mathsf{Tr}\left[C_\theta^{-1}\left(\frac{\mathrm{d}C_\theta}{\mathrm{d}\theta}\right)\right]\delta + O(\delta^2). \end{aligned}$$

For the second term, we proceed similarly. Let C_0, C invertible matrices and $\Delta = C - C_0$. Then $(C_0+\Delta)^{-1} = (I_n + C_0^{-1}\Delta)^{-1}C_0^{-1}$. Suppose Δ is small enough so that $\|C_0^{-1}\Delta\|_{\mathsf{F}} < 1$. Then

$$(I_n + C_0^{-1}\Delta)^{-1} = \sum_{j=\geq 0} (-C_0^{-1}\Delta)^j, \quad (\text{ prove this! }).$$

Hence

$$(C_0+\Delta)^{-1} = \left[\sum_{j\geq 0}(-C_0^{-1}\Delta)^j\right] C_0^{-1} = C_0^{-1} - C_0^{-1}\Delta C_0^{-1} + O(\|\Delta\|_{\mathsf{F}}^2).$$

Then proceed as above to show that

$$y'C_{\theta+\delta}^{-1}y = y'C_\theta^{-1}y - \left(y'C_\theta^{-1}\left(\frac{\mathrm{d}C_\theta}{\mathrm{d}\theta}\right)C_\theta^{-1}y\right)\delta + O(\delta^2).$$

The above derivations also shows that the differential of the map $C \mapsto \log\det C$ (at C spd) is C^{-1}, and the differential of the map $C \mapsto C^{-1}$ (at C invertible) is the linear application $M \mapsto -C^{-1}MC^{-1}$. □

We see that given θ, computing $\ell'(\theta)$ requires solving the system

$$C_\theta Z = [y, \frac{\mathrm{d}C_\theta}{\mathrm{d}\theta}].$$

Remark 2. It is possible to reduce the computation burden of $\ell'(\theta)$ if one is willing to accept some error. The idea goes as follows. Notice that for any matrix $A \in \mathbb{R}^{p\times p}$, if $U \sim \mathsf{N}_p(0, I_p)$, then $\mathbb{E}(U'AU) = \mathsf{Tr}(A)$. Hence we can form a Monte Carlo approximation of $\mathsf{Tr}(A)$ by generating $U_{1:M} \overset{i.i.d.}{\sim} \mathbf{N}(0, I_p)$, and computing

$$\frac{1}{M}\sum_{j=1}^{M} U_j'AU_j.$$

This is known as the **Hutchinson estimator** of the trace. Applied to the above we get the following approximation

$$\widehat{\ell'_M}(\theta) = -\frac{1}{2}\frac{1}{M}\sum_{j=1}^{M} U_j'C_\theta^{-1}\left(\frac{\partial C_\theta}{\partial\theta}\right)U_j + \frac{1}{2}y'C_\theta^{-1}\left(\frac{\mathrm{d}C_\theta}{\mathrm{d}\theta}\right)C_\theta^{-1}y. \tag{3}$$

Clearly $\mathbb{E}(\widehat{\ell'_M}(\theta)) = \ell'(\theta)$. Notice that computing $\widehat{\ell'_M}(\theta)$ now boils down to solving the system

$$C_\theta Z = [y, U_1, \ldots, U_M].$$

If n is very large, significant reduction in computing time is possible by taking M small without too much lost. □

2. Gaussian elimination

We consider our main problem: Let $A \in \mathbb{R}^{m\times m}$ invertible, and $B \in \mathbb{R}^{m\times p}$. We want to solve the linear system $AX = B$. This corresponds to solving p "standard" linear systems $Ax_j = b_j$, $j = 1, \ldots, p$.

2.1. **Upper triangular systems.** We first consider the case where A is upper-triangular:

$$\begin{pmatrix} A_{11} & A_{12} & \cdots A_{1m} \\ 0 & A_{22} & \cdots A_{2m} \\ \vdots & & \vdots \\ 0 & \cdots & A_{mm} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_m \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}.$$

The solution of $Ax = b$ is then readily available by back-substitution:

$$x_j = \frac{1}{A_{jj}} \left(b_j - \sum_{k=j+1}^{m} A_{jk} x_k \right), \quad j = m, \ldots, 1.$$

The number of basic operations involved in computing this solution is

$$O\left(\sum_{j=m}^{1} 2(m-j) \right) = O(m^2).$$

Solving $AX = B$ involves solving $Ax_j = b_j$, for $j = 1, \ldots, p$, where b_j is the j-th column of B. Hence the cost of solving $AX = B$ is $O(pm^2)$.

2.2. **The general case.** Consider now the general case where $A \in \mathbb{R}^{m \times m}$ is some arbitrary invertible matrix. The basic algorithm to solve $AX = B$ is called Gaussian elimination, and consists in successively transforming the equation $AX = B$ into an upper triangular system. This is achieved by finding lower triangular matrices $L_1, L_2, \ldots, L_{m-1}$ such that

$$L_{m-1} \cdots L_1 A = U,$$

where U is upper triangular. These matrices transforms the initial system into $UX = L_{m-1} \cdots L_1 B$ which can be solved by back-substitution.

The matrices L_k are called Gauss transforms and are built as follows. For $1 \leq k \leq m-1$, assume $A_{k,k} \neq 0$ and let $\ell_k(A) \in \mathbb{R}^m$ defined as

$$\ell_k(A) = \frac{1}{A_{k,k}} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ A_{k+1,k} \\ \vdots \\ A_{m,k} \end{bmatrix}.$$

Then define $L_k(A) = I_m - \ell_k(A) e_k'$, where e_j is the j-th Cartesian unit vector of $\mathbb{R}^m$. The matrices $L_k(A)$ are the building blocks of Gaussian elimination. The following result shows that one can efficiently multiply L_k by a vector.

Lemma 2.1.

$$L_k(A)A_{\cdot,k} = \begin{bmatrix} A_{1,k} \\ \vdots \\ A_{k,k} \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

And for any vector $u \in \mathbb{R}^m$ such that $u_k = 0$, we have $L_k(A)u = u$. More generally, for $u \in \mathbb{R}^m$, $L_k(A)u = u - u_k \ell_k(A)$. Hence $(L_k(A)u)_j = u_j$ for $j = 1, \ldots, k$.

The lemma shows that $L_1(A)A$ is a matrix where the first column is $(A_{1,1}, 0\ldots, 0)'$. Setting $A_1 \overset{\text{def}}{=} L_1(A)A$, $L_2(A_1)A_1$ has the same first column as A_1 but the second column is zero below the diagonal. Setting $A_2 \overset{\text{def}}{=} L_2(A_1)A_1$, $L_3(A_2)A_2$ has the same first two columns as A_2, but its third column is zero below the diagonal. We continue this process until we obtain $A_{m-1} = L_{m-1}(A_{m-2})A_{m-2}$ which is upper triangular. The next result shows that we can efficiently multiply and invert the Gauss transforms L_j.

Lemma 2.2. *(1) If $a, b \in \mathbb{R}^m$ are such that $a'b = 0$, then $(I - ab')^{-1} = (I + ab')$.*

(2) If $u \in \mathsf{span}(e_{k+1}, \ldots, e_m)$, and $v \in \mathsf{span}(e_{n+1}, \ldots, e_m)$, for $1 \leq k \leq n \leq m-1$, then

$$(I_m + ue_k')(I_m + ve_n') = I_m + ue_k' + ve_n'.$$

Proof. By simple verification. □

We are led to the following algorithm.

Algorithm 2.1 (Solving $AX = B$ by Gaussian elimination without pivot)**.**

```
(1) Given $A$ and $B$, set $U = A$. For $j = 1$ to $m-1$:
    (a) Compute $\ell_j = (0\ldots, 0, U_{j+1,j}/U_{j,j}, \ldots, U_{m,j}/U_{j,j})'$, and the matrix $L_j = I_m - \ell_j e_j'$.
    (b) Compute $U = L_j U$, and $B = L_j B$.
(2) If Step (1) succeeds, solve the upper-triangular system $UX = B$ by back-substitution.
```

The above algorithm does not fully utilize the properties of the matrices L_j. In particular, we have seen in Lemma 2.1 that $L_j u$ has same first j components as u, and $L_j(A_j)$ does not modify the first $j-1$ columns of A_j. Utilizing these tricks leads to the following algorithm.

Algorithm 2.2 (Solving $AX = B$ by Gaussian elimination without pivot)**.**

```
(1) Given $A$ and $B$, set $U = A$. For $j = 1$ to $m-1$:
    (a) For $k = j+1$ to $m$, do the following. Set $U_{j+1:m,k} = U_{j+1:m,k} - \frac{U_{jk}}{U_{jj}} U_{j+1:m,j}$.
    (b) For $k = 1$ to $p$, set $B_{j+1:m,k} = B_{j+1:m,k} - \frac{B_{jk}}{U_{jj}} U_{j+1:m,j}$.
(2) If Step (1) succeeds, solve the upper-triangular system $UX = B$ by back-substitution.
```

Note that when the Gaussian elimination algorithm succeeds it also gives a factorization of A known as a LU decomposition of A

$$A = LU, \quad L = L_1^{-1} \cdots L_{m-1}^{-1},$$

where L is lower triangular and A upper triangular. Lemma 2.2 tells us how to efficiently multiply the Gauss transforms L_j. Hence if we are only interested in the LU factorization of A itself (as it is sometimes the case) the following algorithm shows how to do it efficiently.

Algorithm 2.3 (*LU* factorization of A)**.** *(1) Given A set $L = I_m$. For $j = 1$ to $m - 1$:*

```
    (a) $L_{(j+1):m,j} = \frac{1}{A_{jj}} A_{(j+1):m,j}$.
    (b) For $k = j+1$ to $m$, do the following. Set $A_{j+1:m,k} = A_{j+1:m,k} - \frac{A_{jk}}{A_{jj}} A_{j+1:m,j}$.
(2) Let $U$ be the upper-triangular part of $A$.
```

Another issue with the Gaussian elimination algorithm as described above is that it will often fail even when A is nonsingular. For example:

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}.$$

The answer to this problem is to swap the rows of A. For example, in the example above, $A = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ or $A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ will work (is already good). So we see the idea. When in the Gaussian elimination, we encounter a 0, we swap rows (partial pivoting). It turns out that even if the current pivot is not zero, for better stability of the algorithm, it can still be beneficial to swap rows. We get the following result.

This leads to the following practical algorithm.

Algorithm 2.4 (Solving $AX = B$ by Gaussian elimination with pivot)**.** *(1) Given A and B, set $U = A$. For $j = 1$ to $m - 1$:*

```
    (a) Find $i$ such that $|U_{ij}| = \max_{j \le q \le m} |U_{q,j}|$.
    (b) Swap $U_{j,j:m}$ and $U_{i,j:m}$. Swap $B_{j,\cdot}$ and $B_{i,\cdot}$.
    (c) For $k = j+1$ to $m$, do the following. Set $U_{j+1:m,k} = U_{j+1:m,k} - \frac{U_{jk}}{U_{jj}} U_{j+1:m,j}$.
    (d) For $k = 1$ to $p$, set $B_{j+1:m,k} = B_{j+1:m,k} - \frac{B_{jk}}{U_{jj}} U_{j+1:m,j}$.
(2) If Step (1) succeeds, solve the upper-triangular system $UX = B$ by back-substitution.
```

The workload of the algorithm is dominated by the calculations of $U_{j+1:m,k} = U_{j+1:m,k} - \frac{U_{jk}}{U_{jj}} U_{j+1:m,j}$ in step (1.d), and the calculation of $B_{j+1:m,k} = B_{j+1:m,k} - \frac{B_{jk}}{U_{jj}} U_{j+1:m}$ in Step (1.e). Thus the number of operations in solving the equation $AX = B$ by Gaussian elimination is a big O of

$$\sum_{j=1}^{m-1} \sum_{i=j+1}^{m} 2(m-j) + \sum_{j=1}^{m-1} \sum_{i=1}^{p} 2(m-j) + pm^2 = 2 \sum_{j=1}^{m-1} (m-j)^2 + 2pm^2 = O\left(\frac{2m^3}{3} + 2pm^2 \right).$$

The second term pm^2 accounts for the computation of the solution of the upper-triangular system.

2.3. **Matlab implementation.** Here is non-optimized Matlab implementation of the Gaussian elimination method.

```
function b= GaussianElim(X,y)
%Solve Xb=y by  Gaussian elimination
%Following Algorithm 3.3 above

m=length(X(:,1));
b=zeros(m,1);
U=X;
for j=1:(m-1)
    [val,k]=max(abs(U(j:m,j))); k=k+j-1;
    %k=k+j-1 return the position of the max in the original set 1 to m
    %pivoting
    Z=U(k,j:m);
    U(k,j:m)=U(j,j:m);U(j,j:m)=Z;
    z=y(k,1);y(k,1)=y(j,1);y(j,1)=z;
    %Action of Lj
    for k=(j+1):m
        ll=U(j,k)/U(j,j);
        U((j+1):m,k)=U((j+1):m,k)-ll*U((j+1):m,j);
    end
    y((j+1):m)=y((j+1):m)-(y(j)/U(j,j))*U((j+1):m,j);
end
%solve for $b
b(m,1)=y(m,1)/U(m,m);
for k=(m-1) : -1 : 1
    b(k,1)=(y(k,1) - U(k,(k+1):m)*b((k+1):m) )/U(k,k);
end
```

Gaussian elimination is commonly performed with pivoting. It no longer leads to the LU decomposition of the matrix A. Rather we get a decomposition $PA = LU$. In fact the term LU decomposition in the literature typically refers to the factorization $PA = LU$.

Theorem 2.3. *Any non-singular matrix A admits a decomposition $PA = LU$, where P is a permutation matrix, L is lower-triang. and U is upper-triang.*

Proof. Gaussian elimination gives $L_{m-1}P_{m-1}\cdots L_1P_1A = U$. Define $\hat{L}_{m-1} = L_{m-1}$, and for $1 \leq j \leq m-2$,

$$\hat{L}_j = P_{m-1}\cdots P_{j+1}L_jP_{j+1}^{-1}\cdots P_{m-1}^{-1}.$$

Because of the special form of the matrix L_j it is not hard to see that $\hat{L}_j$ has the same form $\hat{L}_j = I - \hat{\ell}_j e_j'$ as L_j (recall that $L_j = I - \ell_j e_j'$) with $\hat{\ell}_j = P_{m-1} \cdots P_{j+1} \ell_j$. Hence $\hat{L}_j$ is lower-triang. It is also easy to check that

$$\hat{L}_{m-1} \cdots \hat{L}_1 = L_{m-1} P_{m-1} \cdots L_1 P_1 (P_{m-1} \cdots P_1)^{-1}.$$

Hence, with $P = P_{m-1} \cdots P_1$, and $L = (\hat{L}_{m-1} \cdots \hat{L}_1)^{-1}$, we have

$$PA = LU.$$

□

It turns out that the $\hat{L}_j$ are very easy to compute. Indeed, since $\hat{L}_j = I_m - \hat{\ell}_j e_j'$, $\hat{L}_j^{-1} = I_m + \hat{\ell}_j e_j'$, and $(\hat{L}_{m-1} \cdots \hat{L}_1)^{-1} = I_m + \sum_{j=1}^{m-1} \hat{\ell}_j e_j'$. (Check these!). Given this, it should be easy to see that the following algorithm (a slight modification of Algorithm 2.4) produces the matrices L, P, U such that $PA = LU$.

Algorithm 2.5 (LU decomposition of A). *(1) Given A, set $U = A$, $L = I_m$, $P = I_m$.*

```
(2) For $j = 1$ to $m - 1$:
    (a) Find $i$ such that $|U_{ij}| = \max_{j \le q \le m} |U_{q,j}|$.
    (b) Swap $U_{j,j:m}$ and $U_{i,j:m}$. Swap $L_{j,1:j-1}$ and $L_{i,1:j-1}$. Swap $P_{j,\cdot}$ and $P_{i,\cdot}$.
    (c) Set $L_{(j+1):m,j} = \frac{1}{U_{jj}} U_{(j+1):m,j}$.
    (d) For $k = j + 1$ to $m$, do the following.
        (i) Set $U_{j+1:m,k} = U_{j+1:m,k} - U_{jk} L_{(j+1):m,j}$.
```

3. The Cholesky decomposition

The Cholesky factorization is one of the most useful in statistics as we deal a lot with symmetric matrices. $A \in \mathbb{R}^{m \times m}$ is called symmetrix if $A' = A$. A symmetric matrix A is called semi-definite positive if $u'Au \ge 0$ for any column vector u. If in addition $u'Au = 0$ if and only if $u = 0$, we call A symmetric positive definite (spd).

Lemma 3.1. *If A is a symmetric positive definite matrix and $C \in \mathbb{R}^{m \times n}$, $m \ge n$ is a matrix of full rank, then $C'AC$ is also symmetric positive definite. In particular, $X_{ii} > 0$ for all $i \in \{1, \dots, m\}$ and any sub-matrix $(X_{ij})_{i,j \in I}$, $I \subseteq \{1, \dots, m\}$ is also symmetric and positive definite.*

Theorem 3.2. *Any symmetric positive definite matrix admits a decomposition*

$$A = RR',$$

where R is a lower triangular matrix. The decomposition is unique if we require $R_{ii} > 0$. Such decomposition is called a Cholesky decomposition of A.

Proof. By Lemma 3.1, $A_{11} > 0$. Starting with Gaussian elimination on A, and with a slight modification of the matrix L_1, we get:

$$\begin{pmatrix} 1/\sqrt{A_{11}} & 0 \\ -\frac{A_{2:m,1}}{A_{11}} & I_{m-1} \end{pmatrix} A = \begin{pmatrix} \sqrt{A_{11}} & A'_{2:m,1}/\sqrt{A_{11}} \\ 0 & A_{2:m,2:m} - A_{2:m,1}A'_{2:m,1}/A_{11} \end{pmatrix}$$

Set $R_1 = \begin{pmatrix} 1/\sqrt{A_{11}} & 0 \\ -\frac{A_{2:m,1}}{A_{11}} & I_{m-1} \end{pmatrix}$. Now, since A is symmetric, we can continue the Gaussian elimination to the right and get the block matrix:

$$R_1 A R_1' = \begin{pmatrix} 1 & 0 \\ 0 & A_{2:m,2:m} - A_{2:m,1}A'_{2:m,1}/A_{11} \end{pmatrix}.$$

Set $A^{(1)} = A_{2:m,2:m} - A_{2:m,1}A'_{2:m,1}/A_{11}$. By Lemma 3.1, R_1AR_1' is positive definite and so is $A^{(1)}$. Therefore the same decomposition can be carried over: with $R_{(2)} = \begin{pmatrix} 1/\sqrt{A^{(1)}_{11}} & 0 \\ -\frac{A^{(1)}_{2:(m-1),1}}{A^{(1)}_{11}} & I_{m-2} \end{pmatrix}$, we have

$$R_{(2)}A^{(1)}R'_{(2)} = \begin{pmatrix} 1 & 0 \\ 0 & A^{(2)} \end{pmatrix}.$$

Therefore

$$\begin{pmatrix} 1 & 0 \\ 0 & R_{(2)} \end{pmatrix} R_1 A R_1' \begin{pmatrix} 1 & 0 \\ 0 & R_{(2)} \end{pmatrix}' = \begin{pmatrix} 1 & 0 \\ 0 & R_{(2)}A^{(1)}R'_{(2)} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & A^{(2)} \end{pmatrix}.$$

Continuing this way, we conclude that there exists a lower triangular matrix R such that $RAR' = I_m$. Thus

$$A = R^{-1}(R^{-1})'.$$

□

In practice, we find the Cholesky decomposition of A by solving directly the equation $RR' = A$. Indeed this equation implies that for any $1 \le j \le m$,

$$A_{\cdot,j} = \sum_{k=1}^{j} R_{j,k}R_{\cdot,k}.$$

Thus, for $1 \le j \le m$,

$$R_{j,j} = \sqrt{A_{j,j} - \sum_{k=1}^{j-1} R_{j,k}^2}, \quad \text{and for} \quad i > j, \;\; R_{i,j} = \left(A_{i,j} - \sum_{k=1}^{j-1} R_{j,k}R_{ik} \right) / R_{j,j},$$

where it is understood that $\sum_a^b \cdot = 0$ if $a > b$.

Algorithm 3.1 (Cholesky factorization)**.** *Given A.*
For $j = 1, \dots, m$:

(1) $R_{j,j} = \sqrt{A_{j,j} - \sum_{k=1}^{j-1} R_{jk}^2}$,

(a) For $i = j+1, \ldots, m$*,* $R_{i,j} = \left(A_{i,j} - \sum_{k=1}^{j-1} R_{j,k}R_{ik}\right)/R_{j,j}$*.*

The number of operations is dominated by the inner loop, each iteration of which takes $2j$ operations. Thus the computing time of Cholesky decomposition is big O of

$$\sum_{j=1}^{m}\sum_{i=j+1}^{m} 2j = 2\sum_{j=1}^{m} j(m-j) = O\left(2\left(\frac{m^3}{2} - \frac{m^3}{3}\right)\right) = O\left(\frac{m^3}{3}\right).$$

Thus Cholesky decomposition is 2 times faster than Gaussian elimination.

Remark 3. The discussion above implies that when A is spd, we solve $AX = B$ by first taking the Cholesky decomposition of A: $A = RR'$. Then we solve the upper-triangular system $RZ = B$ by back-substitution, and we solve the lower-triangular system $R'X = Z$ by back-substitution. The computation cost is $O\left(\frac{m^3}{3} + 2pm^2\right)$. Compared to $O\left(\frac{2m^3}{3} + 2pm^2\right)$ if A is arbitrary. $\square$

4. Notes

The material in these notes is very classic and is well covered in the textbooks listed in the syllabus (Trefethen and Bau (1997); Golub and Van Loan (2013)). For more details on the application to graphical models see for instance Ravikumar et al. (2011), and for more details on the application to spatial statistics, see Stein et al. (2013) and the references therein.

References

Golub, G. and Van Loan, C. F. (2013). *Matrix Computations, 4th Ed.* John Hopkins University Press, Baltimore.

Ravikumar, P., Wainwright, M. J., Raskutti, G. and Yu, B. (2011). High-dimensional covariance estimation by minimizing ℓ_1-penalized log-determinant divergence. *Electron. J. Stat.* **5** 935–980.

Stein, M. L., Chen, J. and Anitescu, M. (2013). Stochastic approximation of score functions for Gaussian processes. *Ann. Appl. Stat.* **7** 1162–1191.

Trefethen, L. N. and Bau, D. (1997). *Numerical linear algebra.* SIAM, Philadelphia.