Chapter 5

Linear system of equations

In this chapter we study methods for solving linear systems of equations. That is, we seek a solution in terms of a vector x that satisfies a set of linear equations that can be formulated as a matrix equation $Ax = b$.

For a square non-singular matrix A, we can construct direct solution methods based on factorization of the matrix A into a product of matrices that are easy to invert. In the case of a rectangular matrix A we formulate a least squares problem, where we seek a solution x that minimizes the norm of the residual $b - Ax$.

5.1 Linear system of equations

A linear system of equations can be expressed as the matrix equation

$$Ax = b, \tag{5.1}$$

with A a given matrix and b a given vector, for which x is the unknown solution vector. Given our previous discussion, b can be interpreted as the image of x under the linear transformation A, or alternatively, x can be interpreted as the coefficients of b expressed in the column space of A.

For a square non-singular matrix A the solution x can be expressed in terms of the inverse matrix as $x = A^{-1}b$. For some matrices the inverse matrix A^{-1} is easy to construct, such as in the case of a *diagonal matrix* $D = (d_{ij})$, for which $d_{ij} = 0$ for all $i \neq j$. Here the inverse is directly given as $D^{-1} = (d_{ij}^{-1})$. Similarly, for an orthogonal matrix Q the inverse is given by the transpose $Q^{-1} = Q^T$. On the other hand, for a general matrix A, computation of the inverse is not straight forward. Instead we seek to transform the general matrix into a product of matrices that are easy to invert.

We will introduce two factorizations that can be used for solving $Ax = b$, in the case of A being a general square non-singular matrix; QR factorization and LU factorization. Factorization followed by inversion of the factored matrix is an example of a *direct method* for solving $Ax = b$. We note that to solve the equation we do not have to construct the inverse matrix explicitly, instead we only need to compute the action of matrices on a vector, which is important in terms of the memory footprint of the algorithms.

Apart from diagonal and orthogonal matrices, triangular matrices are easy to invert, by backward and forward substitution.

Triangular matrices

We distinguish between two classes of triangular matrices: a *lower triangular matrix* $L = (l_{ij})$, with $l_{ij} = 0$ for $i < j$, and an *upper triangular matrix* $U = (u_{ij})$, with $u_{ij} = 0$ for $i > j$. The product of lower triangular matrices is lower triangular, and the product of upper triangular matrices is upper triangular. Similarly, the inverse of a lower triangular matrix is lower triangular, and the inverse of an upper triangular matrix is upper triangular.

The equations $Lx = b$ and $Ux = b$, take the form

$$\begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{12} & l_{22} & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ l_{1n} & l_{2n} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}, \quad \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix},$$

solved by *forward substitution* and *backward substitution*, respectively,

$$\begin{aligned} x_1 &= \frac{b_1}{l_{11}} & x_n &= \frac{b_n}{u_{nn}} \\ x_2 &= \frac{b_2 - l_{21}x_1}{l_{22}} & x_{n-1} &= \frac{b_{n-1} - u_{n-1n}x_n}{u_{n-1n-1}} \\ &\cdots & &\cdots \\ x_n &= \frac{b_n - \sum_{i=1}^{n-1} l_{ni}x_i}{l_{nn}} & x_1 &= \frac{b_1 - \sum_{i=2}^{n} u_{1i}x_i}{u_{11}} \end{aligned}$$

where both algorithms correspond to $\mathcal{O}(n^2)$ operations.

5.2 QR factorization

Classical Gram-Schmidt orthogonalization

For a square matrix $A \in \mathbb{R}^{n\times n}$ we denote the successive vector spaces spanned by its column vectors a_j as

$$\langle a_1\rangle \subseteq \langle a_1, a_2\rangle \subseteq \langle a_1, a_2, a_3\rangle \subseteq ... \subseteq \langle a_1, ..., a_m\rangle. \tag{5.2}$$

Assuming that A has full rank, for each such vector space we construct an orthonormal basis q_j, such that $\langle q_1, ..., q_j\rangle = \langle a_1, ..., a_j\rangle$, for all $j \leq n$.

Given a_j, we can successively construct vectors v_j that are orthogonal to the spaces $\langle q_1, ..., q_{j-1}\rangle$, since by (2.25) we have that

$$v_j = a_j - \sum_{i=1}^{j-1}(a_j, q_i)q_i, \tag{5.3}$$

for all $j = 1, ..., n$, where each vector is then normalized to get $q_j = v_j/\|v_j\|$. This is the *classical Gram-Schmidt iteration*.

Modified Gram-Schmidt orthogonalization

If we let $\hat{Q}_{j-1}$ be an $n \times (j-1)$ matrix with the column vectors q_i, for $i \leq j-1$, we can rewrite (5.3) in terms of an orthogonal projector P_j,

$$v_j = a_j - \sum_{i=1}^{j-1}(a_j, q_i)q_i = a_j - \sum_{i=1}^{j-1} q_i q_i^T a_j = (I - \hat{Q}_{j-1}\hat{Q}_{j-1}^T)a_j = P_j a_j,$$

with $\hat{Q}_{j-1}\hat{Q}_{j-1}^T$ an orthogonal projector onto range($\hat{Q}_{j-1}$), the column space of $\hat{Q}_{j-1}$. The matrix $P_j = I - \hat{Q}_{j-1}\hat{Q}_{j-1}^T$ is thus an orthogonal projector onto the space orthogonal to range($\hat{Q}_{j-1}$), with $P_1 = I$. Thus the Gram-Schmidt iteration can be expressed in terms of the projector P_j as $q_j = P_j a_j/\|P_j a_j\|$, for $j = 1, ..., n$.

Alternatively, P_j can be constructed by successive multiplication of projectors $P^{\perp q_i} = I - q_i q_i^T$, orthogonal to each individual vector q_i, such that

$$P_j = P^{\perp q_{j-1}} \cdots P^{\perp q_2} P^{\perp q_1}. \tag{5.4}$$

The *modified Gram-Schmidt iteration* corresponds to instead using this formula to construct P_j, which leads to a more robust algorithm than the classical Gram-Schmidt iteration.

Algorithm 1: Modified Gram-Schmidt iteration

```
for i = 1 to n do
    v_i = a_i
end
for i = 1 to n do
    r_ii = ||v_i||
    q_i = v_i / r_ii
    for j = 1 to i + 1 do
        r_ij = q_i^T v_j
        v_j = v_j - r_ij q_i
    end
end
```

QR factorization

By introducing the notation $r_{ij} = (a_j, q_i)$ and $r_{ii} = \|a_j - \sum_{i=1}^{j-1}(a_j, q_i)q_i\|$, we can rewrite the classical Gram-Schmidt iteration (5.3) as

$$\begin{aligned} a_1 &= r_{11}q_1 \\ a_2 &= r_{12}q_1 + r_{22}q_2 \\ &\vdots \\ a_n &= r_{1n}q_1 + ... + r_{2n}q_n \end{aligned} \tag{5.5}$$

which corresponds to the *QR factorization* $A = QR$, with a_j the column vectors of the matrix A, Q an orthogonal matrix and R an upper triangular matrix, that is

$$\left[\begin{array}{c|c|c|c} a_1 & a_2 & \cdots & a_n \end{array}\right] = \left[\begin{array}{c|c|c|c} q_1 & q_2 & \cdots & q_n \end{array}\right] \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ & r_{22} & & \\ & & \ddots & \vdots \\ & & & r_{nn} \end{bmatrix}.$$

Existence and uniqueness of the QR factorization of a non-singular matrix A follows by construction from Algorithm 1.

The modified Gram-Schmidt iteration of Algorithm 1 corresponds to successive multiplication of upper triangular matrices R_k on the right of the matrix A, such that the resulting matrix Q is an orthogonal matrix,

$$AR_1R_2 \cdots R_n = Q, \tag{5.6}$$

and with the notation $R^{-1} = R_1R_2 \cdots R_n$, the matrix $R = (R^{-1})^{-1}$ is also an upper triangular matrix.

Householder QR factorization

Whereas Gram-Schmidt iteration amounts to *triangular orthogonalization* of the matrix A, we may alternatively formulate an algorithm for *orthogonal triangularization*, where entries below the diagonal of A are zeroed out by successive application of orthogonal matrices Q_k, so that

$$Q_n ... Q_2 Q_1 A = R, \tag{5.7}$$

where we note that the matrix product $Q = Q_n ... Q_2 Q_1$ also is orthogonal.

In the *Householder algorithm*, orthogonal matrices are chosen of the form

$$Q_k = \begin{bmatrix} I & 0 \\ 0 & F \end{bmatrix}, \tag{5.8}$$

with I the $(k-1) \times (k-1)$ identity matrix, and with F an $(n-k+1) \times (n-k+1)$ orthogonal matrix. Q_k is constructed to successively introduce $n-k$ zeros below the diagonal of the kth column of A, while leaving the upper $k-1$ rows untouched, thus taking the form

$$Q_k \hat{A}_{k-1} = \begin{bmatrix} I & 0 \\ 0 & F \end{bmatrix} \begin{bmatrix} \hat{A}_{11} & \hat{A}_{12} \\ 0 & \hat{A}_{22} \end{bmatrix} = \begin{bmatrix} \hat{A}_{11} & \hat{A}_{12} \\ 0 & F\hat{A}_{22} \end{bmatrix}, \tag{5.9}$$

with $\hat{A}_{k-1} = Q_{k-1} \cdots Q_2 Q_1 A$, and with $\hat{A}_{ij}$ representing the *sub-matrices*, or *blocks*, of $\hat{A}_{k-1}$ with corresponding block structure as Q_k.

To obtain a triangular matrix, F should introduce zeros in all subdiagonal entries of the matrix. We want to construct F such that for x an $n-k+1$ column vector, we get

$$Fx = \begin{bmatrix} \pm\|x\| \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \pm\|x\| e_1, \tag{5.10}$$

with $e_1 = (1, 0, ..., 0)^T$ a standard basis vector.

Further, we need F to be an orthogonal matrix, which we achieve by formulating F in the form of a reflector, so that Fx is the reflection of x in a hyperplane orthogonal to the vector $v = \pm\|x\| e_1 - x$, that is

$$F = I - 2\frac{vv^T}{v^T v}. \tag{5.11}$$

We now formulate the full algorithm for QR factorization based on this *Householder reflector*, where we use the notation $A_{i:j,k:l}$ for a sub-matrix

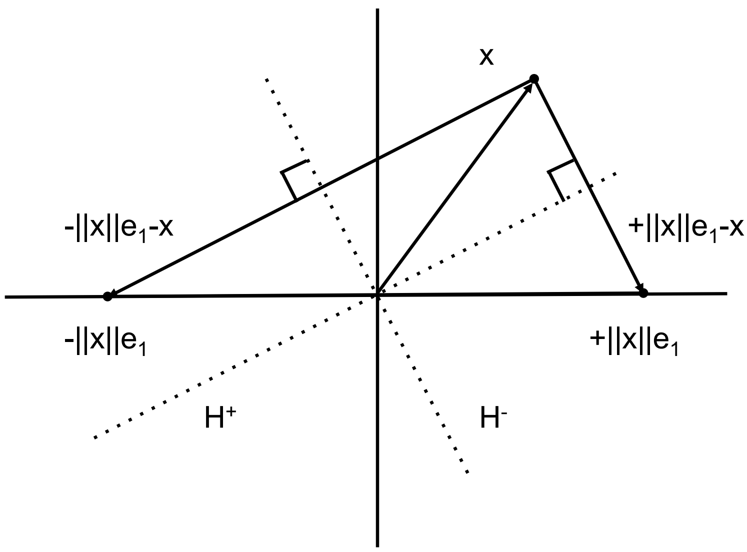

Figure 5.1: Householder reflectors across the two hyperplanes H^+ and H^-.

Algorithm 2: Householder QR factorization

for $k = 1$ **to** n **do**
  $x = A_{k:n,k}$
  $v_k = \text{sign}(x_1)\|x\|_2 e_1 + x$
  $v_k = v_k/\|v_k\|$
  $A_{k:n,k:n} = A_{k:n,k:n} - 2v_k(v_k^T A_{k:n,k:n})$
end

with a row index in the range $i, ..., j$, and column index in the range $k, ..., l$.

We note that Algorithm 2 does not explicitly construct the matrix Q, although from the vectors v_k we can compute the matrix-vector product with $Q = Q_1 Q_2 \cdots Q_n$ or $Q^T = Q_n \cdots Q_2 Q_1$.

5.3 LU factorization

Similar to Householder triangulation, *Gaussian elimination* transforms a square $n \times n$ matrix A into an upper triangular matrix U, by successively inserting zeros below the diagonal. In the case of Gaussian elimination, this is done by adding multiples of each row to the other rows, which corresponds to multiplication by a sequence of triangular matrices L_k from the left, so that

$$L_{n-1} \cdots L_2 L_1 A = U. \tag{5.12}$$

By setting $L^{-1} = L_{n-1} \cdots L_2 L_1$, we obtain the factorization $A = LU$, with $L = L_1^{-1} L_2^{-1} \cdots L_{n-1}^{-1}$.

The kth step in the Gaussian elimination algorithm involves division by the diagonal element u_{kk}, and thus for stability of the algorithm in finite precision arithmetics it is necessary to avoid a small number in that position, which is achieved by reordering the rows, or *pivoting*. With a permutation matrix P, the LU factorization with pivoting may be expressed as $PA = LU$.

Algorithm 3: Gaussian elimination with pivoting

Starting from the matrices $U = A$, $L = I$, $P = I$
for $k = 1$ **to** $n - 1$ **do**
  Select $i \geq k$ to maximize $|u_{ik}|$
  Interchange the rows k and i in the matrices U, L, P
  for $j = k + 1$ **to** n **do**
    $l_{jk} = u_{jk}/u_{kk}$
    $u_{j,k:n} = u_{j,k:n} - l_{jk} u_{k,k:n}$
  end
end

Cholesky factorization

For the case of a symmetric positive definite matrix, A can be decomposed into a product of a lower triangular matrix L and its transpose L^T, which is referred to as the *Cholesky factorization*,

$$A = LL^T. \tag{5.13}$$

In the Cholesky factorization algorithm, symmetry is exploited to perform Gaussian elimination from both the left and right of the matrix A at the same time, which results in an algorithm at half the computational cost of LU factorization.

5.4 Least squares problems

We now consider a system of linear of equations $Ax = b$, for which we have n unknowns but $m > n$ equations, that is $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$.

There exists no inverse matrix A^{-1}, and if the vector $b \notin \text{range}(A)$ we say that the system is *overdetermined*, and thus no exact solution x exists to the equation $Ax = b$. Instead we seek the solution $x \in \mathbb{R}^n$ that minimizes the l_2-norm of the *residual* $b - Ax \in \mathbb{R}^m$, which is referred to as the *least squares problem*

$$\min_{x\in\mathbb{R}^n} \|b - Ax\|_2. \tag{5.14}$$

A geometric interpretation is that we seek the vector $x \in \mathbb{R}^n$ such that the Euclidian distance between Ax and b is minimal, which corresponds to

$$Ax = Pb, \tag{5.15}$$

where $P \in \mathbb{R}^{m\times m}$ is the orthogonal projector onto range(A).

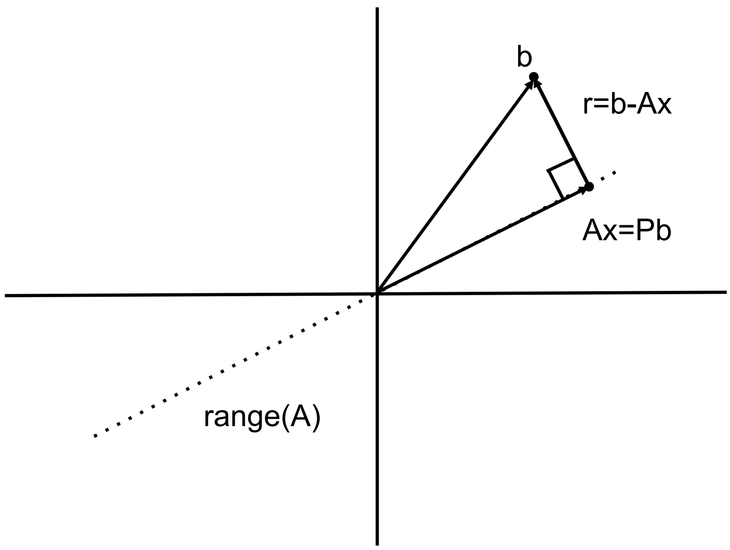

Figure 5.2: Geometric interpretation of the least squares problem.

Thus the residual $r = b - Ax$ is orthogonal to range(A), that is $(Ay, r) = (y, A^T r) = 0$, for all $y \in \mathbb{R}^n$, so that (5.14) is equivalent to

$$A^T r = 0, \tag{5.16}$$

which corresponds to the $n \times n$ system

$$A^T A x = A^T b, \tag{5.17}$$

referred to as the *normal equations*.

The normal equations thus provide a way to solve the $m \times n$ least squares problem by solving instead a square $n \times n$ system. The square matrix $A^T A$ is nonsingular if and only if A has full rank, for which the solution is given as $x = (A^T A)^{-1} A^T b$, where the matrix $(A^T A)^{-1} A^T$ is known as the *pseudoinverse* of A.

5.5 Exercises

Problem 18. *Prove that the product of lower triangular matrices is lower triangular, and the product of upper triangular matrices is upper triangular.*

Problem 19. *Try out the algorithms for QR and LU factorization for a* 3×3 *matrix* A*.*

Problem 20. *Implement the algorithms for QR and LU factorization, and test the computer program for* $n \times n$ *matrices with* n *large.*

Problem 21. *Derive the normal equations for the system*

$$\begin{bmatrix} -2 & 3 \\ -1 & 4 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ -3 \end{bmatrix}. \tag{5.18}$$