Linear equations

A system of linear equations is a set of linear algebraic equations generally written in the form

$$\sum_{j=1}^{n} A_{ij} x_j = b_i \ , \ i = 1 \dots m \ , \tag{1}$$

where $x_1, x_2, \dots, x_n$ are the unknown variables, $A_{11}, A_{12}, \dots, A_{mn}$ are the (constant) coefficients of the system, and $b_1, b_2, \dots, b_m$ are the (constant) right-hand side terms.

The system can be written in matrix form as

$$A\mathbf{x} = \mathbf{b} \ , \tag{2}$$

where A is the $n \times m$ matrix of the coefficients, $\mathbf{x}$ is the size-n column-vector of the unknown variables, and and $\mathbf{b}$ is a size-m column-vector of right-hand side terms.

Systems of linear equations occur regularly in applied mathematics and therefore computational algorithms for finding solutions of linear systems are an important part of numerical analysis. Such algorithms play a prominent role in engineering, physics, chemistry, computer science, and economics.

A system of non-linear equations can often be approximated by a linear system, a helpful technique (called *linearization*) in creating a mathematical model or a computer simulation of a relatively complex system.

If $m = n$, the matrix A is called *square*. A square system has a unique solution if A is nonsingular, that is, has a matrix inverse.

Triangular systems and back-substitution

An efficient algorithm to solve a square system of linear equations numerically is to transform the original system into an equivalent *triangular system*,

$$T\mathbf{y} = \mathbf{c} \ , \tag{3}$$

where T is a *triangular matrix*: a special kind of square matrix where the matrix elements either below or above the main diagonal are zero.

An upper triangular system can be readily solved by *back substitution*:

$$y_i = \frac{1}{T_{ii}} \left(c_i - \sum_{k=i+1}^{n} T_{ik} y_k \right) , \ i = n, \dots, 1 \ . \tag{4}$$

For the lower triangular system the equivalent procedure is called *forward substitution*.

Note that a diagonal matrix, that is a square matrix in which the elements outside the main diagonal are all zero, is also a triangular matrix.

Reduction to triangular form

Popular algorithms for transforming a square system to triangular form are *LU decomposition* and *QR decomposition*.

LU decomposition

LU decomposition is a factorization of a square matrix into a product of a lower triangular matrix L and an upper triangular matrix U,

$$A = LU \ . \tag{5}$$

The linear system $A\mathbf{x} = \mathbf{b}$ after LU-decomposition of the matrix A becomes $LU\mathbf{x} = \mathbf{b}$ and can be solved by first solving $L\mathbf{y} = \mathbf{b}$ for $\mathbf{y}$ and then $U\mathbf{x} = \mathbf{y}$ for $\mathbf{x}$ with two runs of forward and backward substitutions.

If A is a $n \times n$ matrix, the condition (5) is a set of n^2 equations,

$$\sum_{k=1}^{n} L_{ik} U_{kj} = A_{ij} \ , \tag{6}$$

for $n^2 + n$ unknown elements of the triangular matrices L and U. The decomposition is thus not unique.

Usually the decomposition is made unique by providing extra n conditions e.g. by the requirement that the elements of the main diagonal of the matrix L are equal one, $L_{ii} = 1$, $i = 1 \ldots n$. The system (6) can then be easily solved row after row using e.g. the *Doolittle algorithm*,

```
for i = 1 to n :
  L_ii = 1
  for j = 1 to i - 1 :
    L_ij = ( A_ij - sum_{k<j} L_ik U_kj ) / U_jj
  for j = i to n :
    U_ij = A_ij - sum_{k<i} L_ik U_kj
```

QR decomposition

QR decomposition is a factorization of a matrix into a product of an orthogonal matrix Q, such that $Q^T Q = 1$ (where ^{T} denotes transposition), and a right triangular matrix R,

$$A = QR \ . \tag{7}$$

QR-decomposition can be used to convert the linear system $A\mathbf{x} = \mathbf{b}$ into the triangular form

$$R\mathbf{x} = Q^T \mathbf{b}, \tag{8}$$

which can be solved directly by back-substitution.

QR-decomposition can also be performed on non-square matrices with few long columns. Generally speaking a rectangular $n \times m$ matrix A can be represented as a product, $A = QR$, of an orthogonal $n \times m$ matrix Q, $Q^T Q = 1$, and a right-triangular $m \times m$ matrix R.

QR decomposition of a matrix can be computed using several methods, such as Gram-Schmidt orthogonalization, Householder transformations, or Givens rotations.

Gram-Schmidt orthogonalization *Gram-Schmidt orthogonalization* is an algorithm for orthogonalization of a set of vectors in a given inner product space. It takes a linearly independent set of vectors $A = \{\mathbf{a}_1, \ldots, \mathbf{a}_m\}$ and generates an orthogonal set $Q = \{\mathbf{q}_1, \ldots, \mathbf{q}_m\}$ which spans the same subspace as A. The algorithm is given as

```
for i = 1 to m
  q_i <- a_i / ||a_i||  (normalization)
  for j = i + 1 to m
    a_j <- a_j - <a_j, q_i> q_i  (orthogonalization)
```

where $\langle \mathbf{a}, \mathbf{b} \rangle$ is the inner product of two vectors, and $\|\mathbf{a}\| = \sqrt{\langle \mathbf{a}, \mathbf{a} \rangle}$ is the vector's norm. This variant of the algorithm, where all remaining vectors $\mathbf{a}_j$ are made orthogonal to $\mathbf{q}_i$ as soon as the latter is calculated, is considered to be numerically stable and is referred to as *stabilized* or *modified*.

Stabilized Gram-Schmidt orthogonalization can be used to compute QR decomposition of a matrix A by orthogonalization of its column-vectors $\mathbf{a}_i$ with the inner product

$$\langle \mathbf{a}, \mathbf{b} \rangle = \mathbf{a}^T \mathbf{b} \equiv \sum_{k=1}^{n} (\mathbf{a})_k (\mathbf{b})_k \ , \tag{9}$$

where n is the length of column-vectors $\mathbf{a}$ and $\mathbf{b}$, and $(\mathbf{a})_k$ is the kth element of the column-vector.

```
input: matrix A = {a_1, ..., a_m}  (destroyed)
output: matrices R, Q = {q_1, ..., q_m}: A = QR
for i = 1 ... m
  R_ii = (a_i^T a_i)^{1/2}
```

$\mathbf{q}_i = \mathbf{a}_i / R_{ii}$
for $j = i+1 \ldots m$
$R_{ij} = \mathbf{q}_i^T \mathbf{a}_j$
$\mathbf{a}_j = \mathbf{a}_j - \mathbf{q}_i R_{ij}$

The factorization is unique under requirement that the diagonal elements of R are positive. For a $n \times m$ matrix the complexity of the algorithm is $O(mn^2)$.

Determinant of a matrix

LU- and QR-decompositions allow $O(n^3)$ calculations of the of the determinant of a square matrix. Indeed, for the LU-decomposition,

$$\det A = \det LU = \det L \det U = \det U = \prod_{i=1}^{n} U_{ii} \; . \tag{10}$$

For the QR-decomposition

$$\det A = \det QR = \det Q \det R \; . \tag{11}$$

Since Q is an orthogonal matrix $(\det Q)^2 = 1$ and therefore

$$|\det A| = |\det R| = \left| \prod_{i=1}^{n} R_{ii} \right| \; . \tag{12}$$

Matrix inverse

The inverse A^{-1} of a square $n \times n$ matrix A can be calculated by solving n linear equations $A\mathbf{x}_i = \mathbf{z}_i$, $i = 1 \ldots n$, where $\mathbf{z}_i$ is a column where all elements are equal zero except for the element number i, which is equal one. The matrix made of columns $\mathbf{x}_i$ is apparently the inverse of A.

JavaScript implementations

```
function qrdec(A){ // QR-decomposition A=QR of matrix A
  var m=A.length, dot = function(a,b){
    var s=0; for(var i in a) s+=a[i]*b[i]; return s;}
  var R=[[0 for (i in A)] for (j in A)];
  var Q=[[A[i][j] for (j in A[0])] for(i in A)]; //Q is a copy of A
  for(var i=0;i<m;i++){
    var e=Q[i], r=Math.sqrt(dot(e,e));
    if(r==0) throw "qrdec: singular matrix"
    R[i][i]=r;
    for(var k in e) e[k]/=r; //normalization
    for(var j=i+1;j<m;j++){
      var q=Q[j], s=dot(e,q);
      for(var k in q) q[k]-=s*e[k]; //orthogonalization
      R[j][i]=s; } }
  return [Q,R]; } //end qrdec
```

```
function qrback(Q,R,b){ // QR-backsubstitution
// input: matrices Q,R, array b; output: array x such that QRx=b
  var m = Q.length, c = new Array(m), x = new Array(m);
  for(var i in Q){ // c = Q^T b
    c[i]=0; for(var k in b) c[i]+=Q[i][k]*b[k]; }
  for(var i=m-1;i>=0;i--){ // backsubstitution
    for(var s=0, k=i+1;k<m;k++) s+=R[k][i]*x[k];
    x[i]=(c[i]-s)/R[i][i]; }
  return x; } // end qrback
```

```
function inverse(A){ // calculates inverse of matrix A
  var [Q,R]=qrdec(A);
  return [qrback(Q,R,[(k == i?1:0) for(k in A)]) for(i in A)]; } // end inverse
```