1 Systems of linear equations

A system of linear equations is a set of linear algebraic equations,

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

where $x_1, x_2, \dots, x_n$ are the unknown variables, $A_{11}, A_{12}, \dots, A_{mn}$ are the coefficients of the system, and $b_1, b_2, \dots, b_m$ are the constant right-hand terms. The system can be represented 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 unknowns, and and $\mathbf{b}$ is a size-m column-vector of right-hand terms.

Computational algorithms for finding the solutions of systems of linear equations are an important part of numerical analysis, and such methods 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*) when making 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, i.e. has a matrix inverse.

1.1 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.

1.2 Reduction of a linear system to triangular form

Popular algorithms for transforming a square system to triangular form are LU-decomposition and QR-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 equation $A\mathbf{x} = \mathbf{b}$, i.e. $LU\mathbf{x} = \mathbf{b}$, can then 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.

QR-decomposition is a factorization of a matrix into a product of an orthogonal matrix Q, where $Q^TQ = 1$, and a right triangular matrix R,

$$A = QR \ . \tag{6}$$

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

$$R\mathbf{x} = Q^T\mathbf{b}, \tag{7}$$

which can be solved directly by back-substitution.

QR-decomposition can be performed on non-square matrices, where $m < n$, and can be used for linear least-squares problems.

1.3 QR decomposition

A rectangular $n \times m$ matrix A can be represented as a product, $A = QR$, of an orthogonal $n \times m$ matrix Q, $Q^TQ = 1$, and a right-triangular $m \times m$ matrix R.

QR decomposition of a matrix can be computed by means of *modified Gram-Schmidt orthogonalization.*

1.3.1 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 $A = \{\mathbf{a}_1, \dots \mathbf{a}_m\}$ and generates an orthogonal set $Q = \{\mathbf{q}_1, \dots \mathbf{q}_m\}$ which spans the same subspace as A. The algorithm is given as

```
input: set A = {a_1,...,a_m} (destroyed)
output: orthogonal set Q = {q_1,...,q_m}
for i = 1 to m
  q_i ← a_i/||a_i|| (normalization)
  for j = m+1 to m
    a_j ← a_j − ⟨a_j · q_i⟩q_i (orthogonalization)
```

Here $\langle \mathbf{a} \cdot \mathbf{b} \rangle$ is the inner product of two vectors, and $\|\mathbf{a}\| = \sqrt{\langle \mathbf{a} \cdot \mathbf{a} \rangle}$ 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, the algorithm 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

$$\langle \mathbf{a} \cdot \mathbf{b} \rangle = \mathbf{a}^T \mathbf{b} \equiv \sum_{k=1}^{n} (\mathbf{a})_k (\mathbf{b})_k \ , \qquad (8)$$

where superscript ^{T} denotes transposition, n is the length of vectors $\mathbf{a}$ and $\mathbf{b}$, and $(\mathbf{a})_k$ is the kth element of the vector. The algorithm is given as

```
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)
  q_i = a_i / R_ii
  for j = i+1...m
    R_ij = q_i^T a_j
    a_j = a_j - 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 $O(mn^2)$.

1.3.2 Determinant of a matrix

QR-decomposition allows an $O(n^3)$ calculation of the absolute value of the determinant of a square matrix. Indeed,

$$\det(A) = \det(QR) = \det(Q)\det(R) \ . \qquad (9)$$

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

$$|\det(A)| = |\det(R)| \ , \qquad (10)$$

where the determinant $\det(R)$ of a triangular matrix R is simply a product of its diagonal elements.

1.4 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.

1.5 JavaScript implementation

1.5.1 QR-decomposition

```
function qrdec(A){
// QR-decomposition A=QR of matrix A
// input: matrix A
// output: matrices Q,R
  var dot = function(a,b){
    var s=0; for(let 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)];
  for(let i=0;i<Q.length;i++){
    var e=Q[i], r=Math.sqrt(dot(e,e));
    if(r==0){print("singular matrix");
      return undefined}
    R[i][i]=r;
    for(let k in e) e[k]/=r;
    for(let j=i+1;j<Q.length;j++){
      var q=Q[j], s=dot(e,q);
      for(let k in q) q[k]-=s*e[k];
      R[j][i]=s;
      }
    }
  return [Q,R];
}
```

1.5.2 QR-backsubstitution

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

1.5.3 QR-inverse

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