Numerical Algorithms (MU4IN910)

Lecture 2: Matrix computation

Stef Graillat

Sorbonne Université

Summary of the previous lecture

- Introduction to floating-point arithmetic
- Introduction to MATLAB
- Matrix storage
- Efficient tools for matrix manipulations: the BLAS

Goals

We consider computations involving dense matrices (matrices that do no have a large number of zero elements)

We need to

1. efficiently manipulate those matrices on computers
2. choose the right decomposition to solve the problem we consider

Classic problems in linear algebra

Solving linear systems: given a matrix A of size $n \times n$ and a vector b of size n, find x such that

$$Ax = b$$

Solving least-square problems: given a matrix A of size $m \times n$ (with $m > n$) and a vector b of size m, solve

$$\min_x \|b - Ax\|$$

Solving eigenvalue/eigenvector problems: given a matrix A of size $n \times n$, find a vector $x \neq 0$ and a scalar λ such that

$$Ax = \lambda x$$

Outline of the lecture

1. Matrix manipulation
   1. How matrices are stored on computers?
   2. Basic tools for matrix manipulation: the BLAS
2. Matrix decompositions and their uses
   1. LU
   2. QR
   3. Eigendecomposition (diagonalization, etc.)
   4. SVD (singular value decomposition)
3. Software

Bibliography

- **Scientific Computing with Case Studies, D. O'Leary, SIAM, 2009**
- Applied Numerical Linear Algebra, J. Demmel, SIAM, 1997
- Numerical Linear Algebra, L. N. Trefethen and D. Bau, SIAM, 1997
- Matrix Computations, G. Golub and C. Van Loan, Johns Hopkins University Press, 4th edition, 2013
- Matrix Algorithms. Volume I: Basic Decompositions, G. W. Stewart, SIAM, 2001
- Matrix Algorithms. Volume II: Eigensystems, G. W. Stewart, SIAM, 2001

Notation

- All vectors are column vectors
- Matrices are upper case letters; vectors and scalars are lower case
- The element of a matrix A at the (i, j)th entry will be denoted a_{ij} or $A(i, j)$
- I is the identity matrix and e_i is the ith column of I
- $B = A^T$ means that B is the tranpose of A: $b_{ij} = a_{ji}$
- $B = A^*$ means that B is the complex conjugate transpose of A: $b_{ij} = \overline{a}_{ji}$
- We will often use MATLAB notation. For example $A(i : j, k : l)$ denotes the submatrix of A with row entries between i and j and column entries between k and l
- An orthogonal matrix U satisfies $U^T U = I$
- A unitary matrix U satisfies $U^* U = I$
- Two matrices A and B are similar if there exists an invertible matrix X such that $B = XAX^{-1}$

Vector and matrix norms

Definition 1

A ***vector norm*** *is a function* $\|\cdot\| : \mathbb{C}^n \to \mathbb{R}^+$ *satisfying the following conditions:*

1. $\|x\| = 0$ *iff* $x = 0$
2. $\|\alpha x\| = |\alpha|\|x\|$ *for all* $\alpha \in \mathbb{C}$ *and* $x \in \mathbb{C}^n$
3. $\|x + y\| \leq \|x\| + \|y\|$ *for all* $x, y \in \mathbb{C}^n$

Example 1

- $\|x\|_1 = |x_1| + \cdots + |x_n| = \sum_{i=1}^{n} |x_i|$
- $\|x\|_2 = (|x_1|^2 + \cdots + |x_n|^2)^{1/2} = (\sum_{i=1}^{n} |x_i|^2)^{1/2}$
- $\|x\|_\infty = \max_{1 \leq i \leq n} |x_i|$

Vector and matrix norms (cont'd)

Definition 2

A *matrix norm* *is a function* $\|\cdot\| : \mathbb{C}^{m\times n} \to \mathbb{R}^+$ *satisfying the same properties as vector norms.*

Example 2

Subordinate matrix norms *to vector norms:* $\|A\| = \sup_{x\neq 0} \dfrac{\|Ax\|}{\|x\|} = \sup_{\|x\|=1} \|Ax\|$

- $\|A\|_1 = \sup_{x\neq 0} \dfrac{\|Ax\|_1}{\|x\|_1} = \max_{1\leq j\leq n} \sum_{i=1}^{m} |a_{ij}|$
- $\|A\|_\infty = \sup_{x\neq 0} \dfrac{\|Ax\|_\infty}{\|x\|_\infty} = \max_{1\leq i\leq m} \sum_{j=1}^{n} |a_{ij}|$
- $\|A\|_2 = \sup_{x\neq 0} \dfrac{\|Ax\|_2}{\|x\|_2}$

Matrix decompositions and their uses

We are going to study the four following decompositions:

1. LU
2. QR
3. Eigendecomposition (diagonalization, etc.)
4. SVD (singular value decomposition)

Permutation matrix

A permutation matrix is a square matrix that satisfies the following conditions:

- All the coefficients are either 0 or 1
- There is exactly one entry of 1 in each row
- There is exactly one entry of 1 in each column

Multiply on the left by a permutation matrix results in permuting the rows of the matrix A. For example:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \end{pmatrix}$$

LU decomposition

Definition 3

The LU decomposition of an invertible matrix A of size $n \times n$ is defined by $PA = LU$ where

- *P is a permutation matrix*
- *L is a unit lower triangular matrix (Low) (zero above the main diagonal and ones on the main diagonal)*
- *U is an upper triangular matrix (Up)*

$$P \begin{pmatrix} a_{1,1} & \dots & a_{1,n} \\ \vdots & & \vdots \\ a_{n,1} & \dots & a_{n,n} \end{pmatrix} = \begin{pmatrix} 1 & & \\ \vdots & \ddots & \\ \ell_{n,1} & \dots & 1 \end{pmatrix} \begin{pmatrix} u_{1,1} & \dots & u_{1,n} \\ & \ddots & \vdots \\ & & u_{n,n} \end{pmatrix}$$

This corresponds to the Gaussian elimination algorithm

How to compute a LU decomposition

Principle of the algorithm

- The matrix A is reduced to an upper triangular matrix by putting zeros below the main diagonal, column by column, by subtracting a multiple of the current pivot from all rows below it
- Multipliers form the entries of L
- For numerical stability it is necessary to pivot or interchange rows. Changes are stored in P

In MATLAB: `[L,U,P]=lu(A)` or `[PtL,U]=lu(A)` to compute $P^T L$ and U

Cost: $n^3/3$ multiplications

Example of LU decomposition

$$L_2^{-1}A = \begin{pmatrix} 1 & 0 & 0 \\ -1/2 & 1 & 0 \\ -1/4 & 0 & 1 \end{pmatrix} \begin{pmatrix} 4 & 4 & 8 \\ 2 & 8 & 7 \\ 1 & 3 & 6 \end{pmatrix} = \begin{pmatrix} 4 & 4 & 8 \\ 0 & 6 & 3 \\ 0 & 2 & 4 \end{pmatrix}$$

$$L_1^{-1}L_2^{-1}A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1/3 & 1 \end{pmatrix} \begin{pmatrix} 4 & 4 & 8 \\ 0 & 6 & 3 \\ 0 & 2 & 4 \end{pmatrix} = \begin{pmatrix} 4 & 4 & 8 \\ 0 & 6 & 3 \\ 0 & 0 & 3 \end{pmatrix} = U$$

Finally,

$$A = L_2 L_1 U = \begin{pmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 1/4 & 1/3 & 1 \end{pmatrix} \begin{pmatrix} 4 & 4 & 8 \\ 0 & 6 & 3 \\ 0 & 0 & 3 \end{pmatrix} = LU$$

Why a permutation matrix is needed (1)

What about
$$A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}?$$

The pivot is zero, so we factorize the matrix:
$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

Why a permutation matrix is needed (2)

What about

$$A = \begin{pmatrix} 10^{-20} & 1 \\ 1 & 1 \end{pmatrix}?$$

We have $A = LU = \begin{pmatrix} 1 & 0 \\ 10^{20} & 1 \end{pmatrix} \begin{pmatrix} 10^{-20} & 1 \\ 0 & 1 - 10^{20} \end{pmatrix}$

If the arithmetic uses only 16 decimal digits, then $1 - 10^{20}$ is rounded to the number -10^{20}.

The decomposition is then

$$\widetilde{L}\widetilde{U} = \begin{pmatrix} 1 & 0 \\ 10^{20} & 1 \end{pmatrix} \begin{pmatrix} 10^{-20} & 1 \\ 0 & -10^{20} \end{pmatrix} = \begin{pmatrix} 10^{-20} & 1 \\ 1 & 0 \end{pmatrix}$$

Use the permutation matrix to use the largest element in absolute value as the pivot.

Uses of the LU decomposition

One can use the LU decomposition to solve linear system

$$Ax = b,$$

given A and b.

If we factorize A as $PA = LU$ then we have

$$PAx = LUx = Pb$$

Let $y = Ux$, then $Ly = Pb$

To solve $Ax = b$:

1. one solves $Ly = Pb$ by forward substitution
2. one solves $Ux = y$ by backward substitution

In MATLAB, the backslash command `x=A\b` generally uses the LU decompostion to solve a linear system

Uses of the LU decomposition (cont'd)

- To solve $Ax = b$, one first solves $Ly = Pb$ by forward substitution
  For $i = 1 : n$
  $$y_i = (Pb)_i - \sum_{j=1}^{i-1} l_{ij} y_j$$
- and then solve $Ux = y$ by backward substitution
  For $i = n : -1 : 1$,
  $$x_i = \left(y_i - \sum_{j=i+1}^{n} u_{ij} x_j \right) / u_{ii}$$

What about symmetric positive-definite matrices?

A $n \times n$ matrix A is symmetric positive-definite if $A^T = A$ and $x^T A x > 0$ for all $x \neq 0$.

If ω is a vector of size $n-1$ and K a matrix of size $(n-1) \times (n-1)$, one step of Gaussian elimination on $\frac{1}{\alpha}A$ with $a_{1,1} = \alpha^2$ gives:

$$A = \begin{pmatrix} \alpha^2 & \omega^T \\ \omega & K \end{pmatrix} = \begin{pmatrix} \alpha & 0 \\ \omega/\alpha & I \end{pmatrix} \begin{pmatrix} \alpha & \omega^T/\alpha \\ 0 & K - \omega\omega^T/\alpha^2 \end{pmatrix}$$

By factorizing the matrix U as follows:

$$\begin{pmatrix} \alpha & \omega^T/\alpha \\ 0 & K - \omega\omega^T/a_{1,1} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & K - \omega\omega^T/a_{1,1} \end{pmatrix} \begin{pmatrix} \alpha & \omega^T/\alpha \\ 0 & I \end{pmatrix}$$

and combining the two operation, we get:

$$A = \begin{pmatrix} \alpha & 0 \\ \omega/\alpha & I \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & K - \omega\omega^T/a_{1,1} \end{pmatrix} \begin{pmatrix} \alpha & \omega^T/\alpha \\ 0 & I \end{pmatrix} = LDL^T$$

Cholesky decomposition

If A is symmetric positive-definite : $A^T = A$ et $x^T Ax > 0$ for all $x \neq 0$, then it is more convient to use the Choleski decomposition,

$$A = LL^T$$

with L is a lower triangular matrix, or

$$A = LDL^T$$

where L is a unit lower triangular matrix, and D is a diagonal matrix

This gives a decomposition at half the cost of the LU decomposition

In MATLAB, use the command `chol`

Existence, uniqueness of solutions of linear systems

- If A is nonsingular (invertible) then the linear system $Ax = b$ has a unique solution
- If A is singular then x is a solution of $Ax = b$ if b can be written as a linear combinaison of some columns of A. In this case, every vector $x + y$ is a solution if $Ay = 0$.

Sensitivity of the solution of a linear system

Assuming that we perturb the system such that we now need to solve

$$(A + \Delta A)y = b + \Delta b.$$

We want to know to which distance the solution y of the perturbed system is from the solution x of the intial system
Let

$$\begin{aligned}
\varepsilon_A &= \frac{\|\Delta A\|}{\|A\|} \\
\varepsilon_b &= \frac{\|\Delta b\|}{\|b\|} \\
\kappa &= \|A\|\,\|A^{-1}\| \quad \text{condition number of the matrix } A
\end{aligned}$$

If $\kappa\varepsilon_A < 1$ then

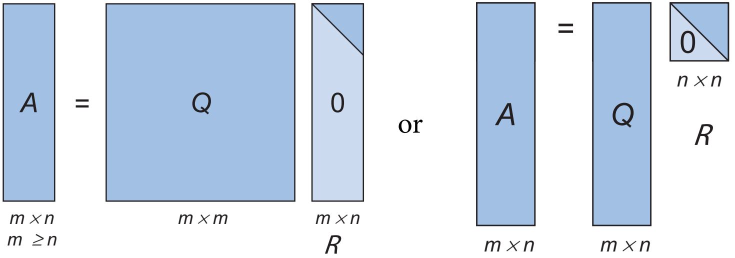
$$\frac{\|x - y\|}{\|x\|} \leq \frac{\kappa}{1 - \kappa\varepsilon_A}(\varepsilon_A + \varepsilon_b)$$

The QR decomposition

Definition 4 (QR decomposition)

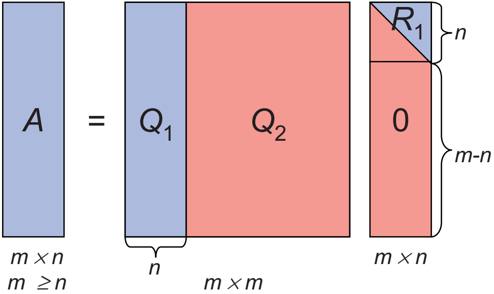
The QR decomposition of a matrix A of size $m \times n$ with $m \geq n$ is defined by $A = QR$ where

- *Q is an $m \times m$ unitary matrix (orthogonal if A is real) and R is an $m \times n$ matrix with zeros below the main diagonal or*
- *Q is an $m \times n$ unitary matrix (orthogonal if A is real) and R is an $n \times n$ upper triangular matrix.*



The QR decomposition (cont'd)

The compact $m \times n$ decomposition arises because part of the $m \times n$ matrix Q is not needed in the decomposition



$$A = Q_1 R_1 + Q_2 0 = Q_1 R_1$$

Algorithms for computing QR decomposition

There are mainly 3 different algorithms for computing the QR decomposition:

1. Givens rotations (good for Q $m \times m$)
2. Gram-Schmidt orthgonalization (good for Q $m \times n$)
3. Householder reflections

We will study methods 1 and 2 in this lecture. We will see method 3 in the tutorial

Givens rotations

We assume all matrices are real ones (we will see the complex case in the tutorial)

A simple orthogonal matrix, a rotation, can be used to introduce one zero at a time into a real matrix.

A Given matrix is written as

$$G = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}$$

where $c^2 + s^2 = 1$ (c and s have the geometric interpretation of the cosine and sine of an angle θ)

A vector multiplied by G is rotated through an angle θ

How to use Givens rotations?

Problem: Given a vector $z \neq 0$ of size 2, find a matrix G such that $Gz = xe_1$ where $x = \|z\|$

Solution:

$$Gz = \begin{pmatrix} cz_1 + sz_2 \\ -sz_1 + cz_2 \end{pmatrix} = xe_1$$

Multiplying the first equation by c, the second by s, and subtracting yields $(c^2 + s^2)z_1 = cx$ and so $c = z_1/x$.

In the same way, we find that $s = z_2/x$

As $c^2 + s^2 = 1$, we can conclude that $z_1^2 + z_2^2 = x^2$, and so

$$\begin{aligned} c &= \frac{z_1}{\sqrt{z_1^2 + z_2^2}} \\ s &= \frac{z_2}{\sqrt{z_1^2 + z_2^2}} \end{aligned}$$

Givens QR decomposition

So we know how to use Givens matrices to zero out single components of a matrix. We will use the notation G_{ij} to denote an $n \times n$ identity matrix with its ith and jth rows modified to include the Givens rotation.

Example if $n = 6$:

$$G_{25} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & c & 0 & 0 & s & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & -s & 0 & 0 & c & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

The multiplication of a vector by this matrix leaves all but rows 2 and 5 of the vector unchanged.

Givens QR decomposition (cont'd)

Algorithm 1 (Givens algorithm)

Initialize Q to be the $m \times m$ identity matrix
Initialize R to be the $m \times n$ matrix A
for $i = 1:n$
  for $j = i+1:m$
   – Choose the matrix G_{ij} to put a zero in position (j,i) of the matrix R, using the current value in position (i,i)
   – $R = G_{ij} R$
   – $Q = Q G_{ij}^T$
  end
end

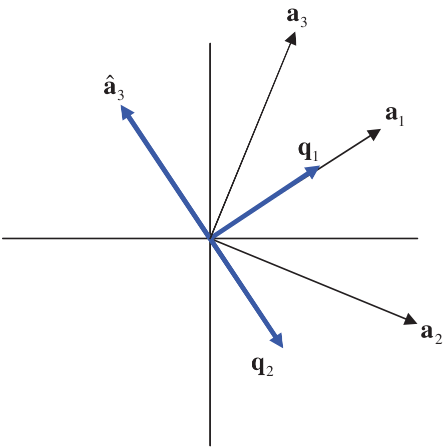
QR by Gram-Schmidt

From the columns $[a_1, \ldots, a_n]$ of the matrix A, we create an orthonormal basis $\{q_1, \ldots, q_n\}$ and save the coefficients that accomplish this goal in an upper triangular matrix R.

Algorithm 2 (Gram-Schmidt orthogonalization)

$r_{1,1} = \|a_1\|$
$q_1 = a_1/r_{1,1}$
for $k = 1:n-1$
    $q_{k+1} = a_{k+1}$
    for $i = 1:k$
        $r_{i,k+1} = q_i^* q_{k+1}$
        $q_{k+1} = q_{k+1} - r_{i,k+1} q_i$
    end
    $r_{k+1,k+1} = \|q_{k+1}\|$
    $q_{k+1} = q_{k+1}/r_{k+1,k+1}$
end

QR by Gram-Schmidt (cont'd)



Cost of QR decomposition

- Givens rotations: $2mn^2 - 2/3n^3$ multiplications
- Gram-Schmidt: mn^2 multiplications
- Householder reflections: $mn^2 - 1/3n^3$ multiplications

Uses of QR decomposition

In MATLAB : `[Q,R] = qr(A)` for A of size $m \times n$ with $m \geq n$

- `qr(A,0)` returns the compact matrix Q (although the full is computed with Householder reflections)
- QR can get the basis for the range of a full-rank matrix A (the first n columns of Q) and the null-space of A^* (the last $m - n$ columns of Q).
- QR can be used to solve linear least squares problems

Solution of a linear system: $Ax = b$
$\rightarrow$ if $A = QR$ then $Rx = Q^* b$

Solving linear least squares problems

Given A of size $m \times n$ (with $m > n$), we want to find

$$\min_x \|Ax - b\|$$

where $\|\cdot\|$ is the Euclidean norm (2-norm).

- Minimizing $\|Ax - b\|$ gives the same solution as minimizing $\|Ax - b\|^2$
- The norm of a vector is invariant under multiplication by Q^*, so $\|y\| = \|Q^* y\|$ for all y
- Suppose we partition the vector y into two pieces:

$$y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

Then $\|y\|^2 = \|y_1\|^2 + \|y_2\|^2$

Solving linear least squares problems (cont'd)

If $A = QR$, we define

$$c = Q^* b = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}$$

with c_1 of size n and c_2 of size $m - n$.
Then

$$\begin{array}{rcl} \|b - Ax\|^2 & = & \|Q^*(b - Ax)\|^2 \\ & = & \|c - Rx\|^2 \\ & = & \|c_1 - R_1 x\|^2 + \|c_2 - 0x\|^2 \\ & = & \|c_1 - R_1 x\|^2 + \|c_2\|^2 \end{array}$$

The minimum is obtained for x solution of $R_1 x = c_1$

In MATLAB, use `x = A\b`

Case study: data fitting

Data (t_i, f_i) represent the amount of a pollutant in a river, measured once a year.

We want to know whether a straight line is a good fit to this data!

We want to solve

$$\min_x \|Ax - b\|$$

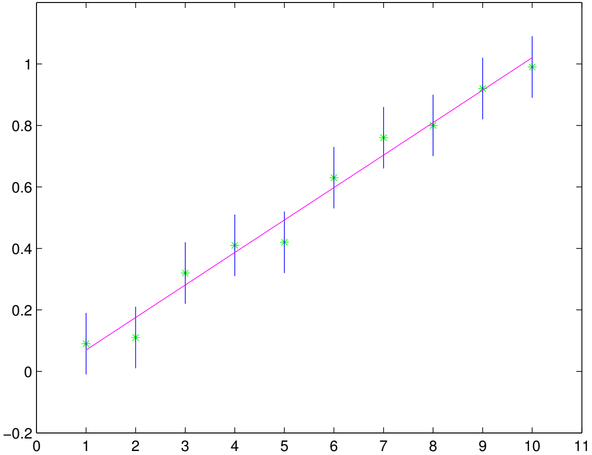
with

$$A = \begin{pmatrix} 1 & t_1 \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ 1 & t_{10} \end{pmatrix}, \qquad b = \begin{pmatrix} f_1 \\ \cdot \\ \cdot \\ \cdot \\ f_{10} \end{pmatrix}$$

Case study: data fitting (cont'd)

```
sigma=.05;
t = [1:10];
b = [0.09 0.11 0.32 0.41 0.42 0.63 0.76 0.8 0.92 0.99];
plot(t,b,'g*');
hold on;
for i=1:10
   plot([t(i),t(i)],[b(i)+2*sigma,b(i)-2*sigma]);
end
axis([0 11 -.2 1.2]);
A = [ones(10,1),t'];
x = A \ b';
plot(t,A*x,'m');
```

Case study: data fitting (cont'd)



Eigendecomposition

Definition 5

A matrix A of size $n \times n$ is diagonalizable if $A = U \Lambda U^{-1}$ where Λ is a diagonal matrix whose diagonal entries are the eigenvalues λ_i. The columns of U are called the eigenvectors:

$$Au_i = \lambda_i u_i.$$

The decomposition is guaranteed to exist if

- A is real symmetric or complex Hermitian, or
- A is normal ($AA^* = A^* A$), or
- the eigenvalues of A are distinct.

Otherwise, the decomposition may fail to exist, although it will exist for a nearby matrix.

How to compute the eigendecomposition

Algorithm with 2 steps:

Step 1: reduce the matrix A to compact form, so that it is easy to manipulate.
Find a unitary matrix V so that

$$V^* A V = H$$

where H is

- tridiagonal if A is Hermitian (or real symmetric)
- upper Hessenberg otherwise

This can be done in $\mathcal{O}(n^3)$ operations.

The matrices A and H being similar, if we find an eigendecomposition of H as

$$H = U \Lambda U^{-1}$$

then we have the eigendecomposition

$$A = (VU)\Lambda(VU)^{-1}$$

for A

How to compute the eigendecomposition (cont'd)

Step 2: Find the eigendecomposition of H by QR iteration:

- Form $H = QR$
- Replace H by RQ

As $Q^*Q = I$ and $H = QR$, we have

$$RQ = (Q^*Q)RQ = Q^*HQ$$

As a consequence, the new H has the same eigenvalues as the old one, and if we have an eigendecomposition of RQ, then we have an eigendecomposition of H

We repeat Step 2 many times (about $5n$, typically), and often some subdiagonal elements of H converge to zero. Once that happens, we can read some eigenvalues off the diagonal.

In MATLAB, `[U,Lambda]=eig(A)`

Cost of eigendecomposition

- The cost of Step 1 is $\mathcal{O}(n^3)$
- The cost of $\mathcal{O}(n)$ iterations of Step 2 is $\mathcal{O}(n^3)$
- The total cost is $\mathcal{O}(n^3)$

Uses of eigendecomposition

- stability analysis in control theory
- convergence of iterative methods
- computation of invariant subspaces
- convergence of A^p when $p \to +\infty$

The Singular Value Decomposition (SVD)

Definition 6 (SVD)

Every matrix A of dimensions $m \times n$ (with $m \geq n$) can be decomposed as

$$A = U \Sigma V^*$$

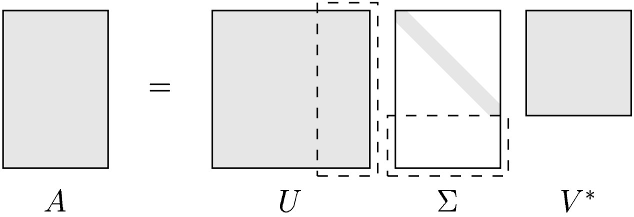
where

- *U has dimension $m \times m$ and $U^* U = I$*
- *Σ has dimension $m \times n$, the only nonzeros are on the main diagonal, and they are nonnegative real numbers $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$*
- *V has dimension $n \times n$ and $V^* V = I$*

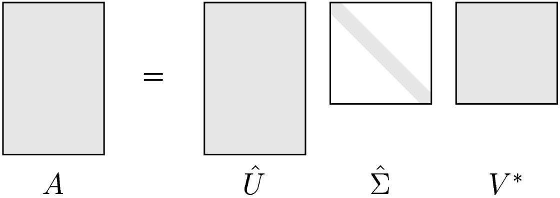
The σ_i are called the singular values of A

The Singular Value Decomposition (SVD)

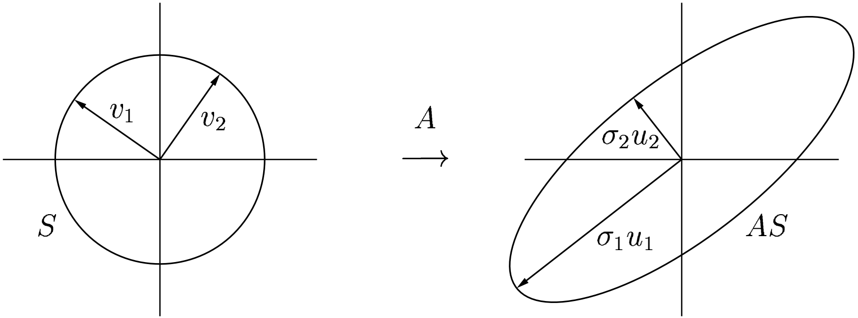
- Full decomposition:



- Compact decomposition:



Geometric interpretation of the SVD



Some useful relations

If $A = U\Sigma V^*$ then

$$A^*A = (U\Sigma V^*)U\Sigma V^* = V\Sigma^* U^* U\Sigma V^* = V\Sigma^*\Sigma V^*$$

Therefore,

- The singular values σ_i of A are the square roots of the eigenvalues of A^*A
- The columns of V are the right singular vectors of A and the eigenvectors of A^*A
- The columns of U are the left singular vectors of A and the eigenvectors of AA^*

How to compute the SVD

We can compute the SVD as follows:

1. Compute A^*A
2. Compute the eigendecomposition of $A^*A = V\Lambda V^*$
3. Let Σ the $m \times n$ matrix whose diagonal entries are the square root of the diagonal entries of Λ
4. Solve $U\Sigma = AV$ with unitary matrix U

This algorithm is unstable! Nevetheless there exits efficient and stable algorithms to compute SVD

In MATLAB, `[U,S,V]=svd(A)`

Cost: $\mathcal{O}(mn^2)$ with constant usually of order 10

Uses of the SVD: Uses of the SVD include solving ill-conditioned least squares problems, representing the range or null space of a matrix, image compression, image deblurring, etc.

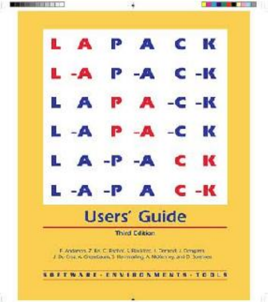
Some tasks to avoid

- matrix inverse
  We can solve $Ax = b$ by multiplying both sides of the equation by A^{-1}:
  $$A^{-1}Ax = x = A^{-1}b$$
  Therefore, we can solve linear systems by multiplying the right-hand side b by A^{-1}.
  This is a bad idea. It is more expensive than the LU decomposition and it generally computes an answer that has larger error.
  Whenever you see a matrix inverse in a formula, think "LU decomposition".
- Jordan canonical form
  Some matrices do not have an eigendecomposition like
  $$A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$
  but every matrix can be decomposed into Jordan canonical form as,
  $$A = WJW^{-1}$$

Software

For computing matrix decompositions and solving matrix problems in Fortran or C, look for LAPACK (`http://www.netlib.org/lapack/`).

- numerically stable algorithms
- uniform interface, making use easy
- row or column oriented implementation, appropriate for the matrix storage scheme used by the language
- built on BLAS and thus efficient (at least efficient when n is large (100 or more). The overhead for small n is quite big)



For Java, we can use JavaNumerics (`http://math.nist.gov/javanumerics/`)

MATLAB is based on LAPACK

A summary of matrix decompositions

Decomposition	Multiplications	Examples of uses
LU	$n^3/3$	• Solving linear systems • Computing determinants
QR	$mn^2 - 1/3n^3$	• Solving well-conditioned linear least squares problems • Representing the range or null space of a matrix
Eigendecomposition	$\mathcal{O}(n^3)$	• Determining eigenvalues or eigenvectors of a matrix • Determining invariant subspaces. • Determining stability of a control system • Determining convergence of A^p when $p \to +\infty$
SVD	$\mathcal{O}(mn^2)$	• Solving ill-conditioned linear least squares problems • Representing the range or null space of a matrix • Computing an approximation to a matrix

Conclusion

- We have not discussed sparse matrices, those with mostly zero entries. There exists some specific algorithms for those matrices that preserve the sparsity.
- There exists some specific algorithms for structured matrices (for example symmetric, tridiagonal, etc)