Scientific Computing

Direct solution methods

Martin van Gijzen

Delft University of Technology

October 3, 2018

Program October 3

- Matrix norms
- LU decomposition
  - Basic algorithm
  - Cost
  - Stability
  - Pivoting
- Pivoting
- Choleski decomposition
- Sparse matrices and reorderings

Matrix norms (1)

The analysis of matrix algorithms frequently requires use of matrix norms.

For example, the quality of a linear system solver may be poor if the matrix of coefficients is "nearly singular".

To quantify the notion of near-singularity we need a measure of distance on the space of matrices. Matrix norms provide that measure.

Matrix norms (2)

A matrix norm on $\mathbb{R}^{m\times n}$ is a function $\|.\| : \mathbb{R}^{m\times n} \to \mathbb{R}$ that satisfies the following properties:

i) $\|A\| \geq 0 \qquad A \in \mathbb{R}^{m\times n}$,
and $\qquad \|A\| = 0 \iff A = 0$,

ii) $\|A + B\| \leq \|A\| + \|B\| \qquad A, B \in \mathbb{R}^{m\times n}$,

iii) $\|\alpha A\| = |\alpha| \; \|A\| \qquad \alpha \in \mathbb{R} \; , \; x \in \mathbb{R}^{m\times n}$.

The most commonly used matrix norms are the p-norms induced by the vector p-norms.

$$\|A\|_p = \sup_{x\neq 0} \frac{\|Ax\|_p}{\|x\|_p} = \max_{\|x\|_p=1} \quad \|Ax\|_p \quad p \geq 1.$$

Matrix norms (3)

Below we list some properties of vector and matrix p-norms

- $\|AB\|_p \leq \|A\|_p \|B\|_p \qquad A \in \mathbb{R}^{m\times n} \;\; , \;\; B \in \mathbb{R}^{n\times q}$
- $\|A\|_1 = \max\limits_{1\leq j\leq n} \sum\limits_{i=1}^{m} |a_{ij}| \qquad A \in \mathbb{R}^{m\times n}$
- $\|A\|_\infty = \max\limits_{1\leq i\leq m} \sum\limits_{j=1}^{n} |a_{ij}| \qquad A \in \mathbb{R}^{m\times n}$
- $\|A\|_2$ is equal to the square root of the largest eigenvalue of $A^T A$.
- All norms are equivalent, meaning that there are $m, M > 0$ such that $m\|A\|_p \leq \|A\|_q \leq M\|A\|_p$.

Matrix norms (4)

Matrix norms that are not induced by a vector norm also exist. One of the best known is the Frobenius norm. The Frobenius norm of an $m \times n$ matrix A is given by

$$\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{i=1}^{n} |a_{ij}|^2}$$

This is equal to

$$\|A\|_F = \sqrt{Tr(AA^T)}$$

In which $Tr(A)$ is the trace of A, which is the sum of the main diagonal elements.

Condition number (1)

The condition number plays an important role in numerical linear algebra since it gives a measure of how perturbations in A and b affect the solution x.

The condition number $K_p(A)$, for a nonsingular matrix A, is defined by

$$K_p(A) = \|A\|_p \|A^{-1}\|_p .$$

A small condition number means that small perturbations in the matrix or right-hand side give small changes in the solution. A large condition number means that a small perturbation in the problem may give a large change in the solution.

Condition number (2)

Suppose $Ax = b,\ A \in \mathbb{R}^{n \times n}$ and A is nonsingular, $0 \neq b \in \mathbb{R}^n$, and $A(x + \Delta x) = b + \Delta b$, then

$$\frac{\|\Delta x\|_p}{\|x\|_p} \leq K_p(A) \frac{\|\Delta b\|_p}{\|b\|_p} \ .$$

Condition number (2)

Suppose $Ax = b,\ A \in \mathbb{R}^{n\times n}$ and A is nonsingular, $0 \neq b \in \mathbb{R}^n$, and $A(x + \Delta x) = b + \Delta b$, then

$$\frac{\|\Delta x\|_p}{\|x\|_p} \leq K_p(A)\frac{\|\Delta b\|_p}{\|b\|_p}\ .$$

Proof: From the properties of the norms it follows that

$$\|b\|_p = \|Ax\|_p \leq \|A\|_p\|x\|_p,\ \text{so}\ \frac{1}{\|x\|_p} \leq \|A\|_p\frac{1}{\|b\|_p}.$$

We know that $A\Delta x = \Delta b$, so $\Delta x = A^{-1}\Delta b$. Furthermore

$$\|\Delta x\|_p = \|A^{-1}\Delta b\|_p \leq \|A^{-1}\|_p\|\Delta b\|_p.$$

Combination of these inequalities proves the theorem.

Condition number (3)

Suppose you want the solution x of

$$Ax = b,\ A \in \mathbb{R}^{n\times n}\text{nonsingular},\ 0 \neq b \in \mathbb{R}^n$$

You actually solve the perturbed system

$$(A + \Delta A)(x + \Delta x) = b + \Delta b,\ \Delta A \in \mathbb{R}^{n\times n},\ \Delta b \in \mathbb{R}^n$$

with $\|\Delta A\|_p \leq \delta \|A\|_p$ and $\|\Delta b\|_p \leq \delta \|b\|_p$.

When has this system a (unique) solution?

Condition number (3)

Suppose you want the solution x of

$$Ax = b,\ A \in \mathbb{R}^{n\times n}\text{nonsingular},\ 0 \neq b \in \mathbb{R}^n$$

You actually solve the perturbed system

$$(A + \Delta A)(x + \Delta x) = b + \Delta b,\ \Delta A \in \mathbb{R}^{n\times n},\ \Delta b \in \mathbb{R}^n$$

with $\|\Delta A\|_p \leq \delta \|A\|_p$ and $\|\Delta b\|_p \leq \delta \|b\|_p$.

When has this system a (unique) solution?

If $K_p(A)\delta = r < 1$ then $A + \Delta A$ is nonsingular and

$$\frac{\|\Delta x\|_p}{\|x\|_p} \leq \frac{2\delta}{1-r} K_p(A).$$

<u>Proof</u>: Lecture notes, p.44.

Gaussian elimination

Consider the system

$$Ax = \begin{bmatrix} 2 & 1 & 1 \\ 4 & 1 & 0 \\ -2 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \\ 7 \end{bmatrix}$$

Then we can reduce this system to upper triangular form

- (i) Subtract two times equation one from equation two
- (ii) Subtract -1 times equation one from equation three
- (iii) Subtract -3 times the second equation from the third

Gaussian elimination (2)

The resulting equivalent system is

$$Ux = \begin{bmatrix} 2 & 1 & 1 \\ 0 & -1 & -2 \\ 0 & 0 & -4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ -4 \\ -4 \end{bmatrix}$$

This system can be solved by back substitution.

If we have a different right-hand side, do we have to do the same operations? Or can we save what we did?

Gaussian elimination (3)

The first reduction step was 'Subtract two times equation one from equation two'. The so-called elementary matrix $E_{2,1}$ that performs this operations is

$$E_{2,1} = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Multiplying with this matrix yields

$$E_{2,1}A = \begin{bmatrix} 2 & 1 & 1 \\ 0 & -1 & -2 \\ -2 & 2 & 1 \end{bmatrix}.$$

Gaussian elimination (4)

The second reduction step 'Subtract -1 times equation one from equation three' is equivalent with multiplying with the matrix

$$E_{3,1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

and 'Subtract -3 times the second equation from the third' is equivalent with multiplying with the matrix

$$E_{3,2} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 3 & 1 \end{bmatrix}$$

Gaussian elimination (4)

The second reduction step 'Subtract -1 times equation one from equation three' is equivalent with multiplying with the matrix

$$E_{3,1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

and 'Subtract -3 times the second equation from the third' is equivalent with multiplying with the matrix

$$E_{3,2} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 3 & 1 \end{bmatrix}$$

Gaussian elimination (5)

Hence we can write

$$E_{3,2}E_{3,1}E_{2,1}A = U$$

Notice that the matrix $E_{3,2}E_{3,1}E_{2,1}$ is a product of lower triangular matrices and therefore lower triangular.
The above equation can also be written as

$$A = E_{2,1}^{-1}E_{3,1}^{-1}E_{3,2}^{-1}U$$

Since the inverse of a lower triangular matrix is lower triangular, the matrix $E_{2,1}^{-1}E_{3,1}^{-1}E_{3,2}^{-1}$ must be lower triangular:

$$L = E_{2,1}^{-1}E_{3,1}^{-1}E_{3,2}^{-1} \qquad A = LU$$

Gaussian elimination (6)

It is easy to check that

$$E_{2,1}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad E_{3,1}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} E_{3,2}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -3 & 1 \end{bmatrix}$$

and that

$$L = E_{2,1}^{-1} E_{3,1}^{-1} E_{3,2}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -3 & 1 \end{bmatrix}$$

Gaussian elimination (7)

Clearly L is lower triangular, with 1's on the main diagonal. The special thing is that the entries below the diagonal are exactly the multipliers 2, -1, -3 used in the elimination steps.

The example shows that an LU-decomposition can be made by

- Reducing A to upper triangular form by elementary row operations, this gives U,
- Storing all the multipliers in a lower triangular matrix L that has ones on the main diagonal.

Gaussian elimination algorithm

Given $A \in \mathbb{R}^{n\times n}$ the following algorithm computes the factorization $A = LU$.

```
for $k = 1, \ldots, n-1$ do
    for $i = k+1, \ldots, n$ do
        $\eta := a_{ik}/a_{kk}$
        $a_{ik} = \eta$
        for $j = k+1, \ldots, n$
            $a_{ij} := a_{ij} - \eta\, a_{kj}$
        end for
    end for
end for
```

The element a_{kk} is called pivot

Some remarks (1)

- The algorithm overwrites the matrix A with the matrix U (upper triangular part) and the matrix L (strictly lower triangular part). The main diagonal of L is not stored.
- The value a_{kk} is called the pivot. The numerical stability of the algorithm depends on the size of the pivots. If a pivot is zero the algorithm breaks down. If a pivot is close to zero large numerical errors may occur.
- The number of floating point operations to compute the LU-decomposition is $2n^3/3$.

Some remarks (2)

- A more 'symmetrical' variant of the LU-decomposition is the LDU-decomposition. Here D is a diagonal that scales the main diagonal elements of U to one.
- A system of the form $Ax = LUx = b$ can be solved by a forward substitution $Ux = L^{-1}b$, and a back substitution $x = U^{-1}L^{-1}b$. Once the LU-decomposition of A is known, a system with A can simply be solved by forward and back substitution with the LU factors.
- Both the back and forward substitution (algorithms given in the next two slides) require n^2 flops.

Forward substitution

Given an $n \times n$ nonsingular lower triangular matrix L and $b \in \mathbb{R}^n$, the following algorithm finds $y \in \mathbb{R}^n$ such that $Ly = b$.

Forward substitution algorithm

```
for $i = 1, \dots, n$ do
    $y_i := b_i$
    for $j = 1, \dots, i-1$ do
        $y_i := y_i - \ell_{ij}\, y_j$
    end for
    $y_i := y_i / \ell_{ii}$
end for
```

Backward substitution

Given an $n \times n$ nonsingular upper triangular matrix U and $y \in \mathbb{R}^n$, the following algorithm finds $x \in \mathbb{R}^n$ such that $Ux = y$.

Back substitution algorithm

```
for $i = n, \ldots, 1$ do
    $x_i := y_i$
    for $j = i + 1, \ldots, n$ do
        $x_i := x_i - u_{ij}\, x_j$
    end for
    $x_i := x_i / u_{ii}$.
end for
```

Round off errors, a bound

We use the following conventions, if A and B are in $\mathbb{R}^{m\times n}$ then

- $B = |A|$ means
  $b_{ij} = |a_{ij}| \quad , \quad i = 1, \ldots, m \quad , \quad j = 1, \ldots, n.$
- $B \leq A$ means $b_{ij} \leq a_{ij} \quad , \quad i = 1, \ldots, m \quad , \quad j = 1, \ldots, n.$

Theorem Let $\hat{L}$ and $\hat{U}$ be the computed LU factors of the $n \times n$ floating point matrix A. Suppose that $\hat{y}$ is the computed solution of $\hat{L}y = b$ and $\hat{x}$ is the computed solution of $\hat{U}x = \hat{y}$. Then $(A + \triangle A)\hat{x} = b$ with

$$|\Delta A| \leq n(3|A| + 5|\hat{L}||\hat{U}|)u + O(u^2).$$

Proof: see Golub and van Loan p.107.

Round off errors, discussion

The term $|\hat{L}||\hat{U}|$ can be large if a small pivot is encountered during the elimination process.
Small pivots do not necessarily indicate an ill-conditioned problem. Consider for example

$$A = \begin{bmatrix} \epsilon & 1 \\ 1 & 0 \end{bmatrix}$$

Gaussian elimination (without further precautions) can give arbitrarily poor results, even for well-conditioned problems. The method may be unstable, depending on the matrix.
Assignment: Compute the LU factorisation of A and bound $|\Delta A|$.

Permutations

The problem in the previous example can of course be solved by interchanging the rows (or columns). This is a row permutation.

Suppose that the i-th column of the identity matrix I is denoted by e_i. A row permuted version of A is given by PA,

where $P = \begin{pmatrix} e_{s_1}^T \\ \vdots \\ e_{s_n}^T \end{pmatrix}$.

The matrix AP^T is a column permuted version of A.

Exchanging rows (and/or columns) for selecting large pivots is called pivoting.

Partial pivoting

A simple strategy to avoid small pivots is partial pivoting:

- Determine the in absolute value largest element below the pivot,
- Interchange the corresponding row with the pivot row.
- Eliminate all nonzeros elements below the pivot.

Remark: partial pivoting only works well if the elements of the matrix are scaled in some way.

Partial pivoting (2)

Partial pivoting leads to the decomposition

$$PA = LU.$$

The permutation matrix P corresponds to all the row exchanges.

Using partial pivoting we can show that no multiplier is greater than one in absolute value.

Symmetric positive definite systems

In many applications the matrix A, used in the linear system $Ax = b$, is symmetric and positive definite. So matrix A satisfies the following rules:

- $A = A^T$,
- $x^T Ax > 0 \quad , \quad x \in \mathbb{R}^n \quad , \quad x \neq 0.$

For this type of matrices, memory and CPU time can be saved. Since A is symmetric only the elements $a_{ij},\ i = j, \ldots, n \quad ; \quad j = 1, \ldots, n$ should be stored in memory. Moreover, a cheap LU-like decomposition exists.

The Choleski decomposition

Algorithm

Given a symmetric positive definite $A \in \mathbb{R}^{n\times n}$, the following algorithm computes $A = CC^T$, with C lower triangular. The entry a_{ij} is overwritten by $c_{ij} (i \geq j)$.

```
for   $k = 1, 2, \ldots, n$ do
      $a_{kk} := (a_{kk} - \sum_{p=1}^{k-1} a_{kp}^2)^{1/2}$
      for  $i = k+1, \ldots, n$ do
           $a_{ik} := (a_{ik} - \sum_{p=1}^{k-1} a_{ip} a_{kp}) / a_{kk}$
      end for
end   for
```

The Choleski decomposition (2)

The Choleski decomposition has many favourable properties:

- The number of flops for the algorithm is $n^3/3$; both memory and operations are half that of the LU decomposition for general matrices.
- The inequality

$$c_{ij}^2 \leq \sum_{p=1}^{i} c_{ip}^2 = a_{ii} \ ,$$

  shows that the elements in C are bounded: pivoting is not necessary.

Banded systems

In many applications the matrix is banded. This is the case whenever the equations can be ordered so that each unknown x_i appears in only a few equations in a 'neighborhood' of the ith equation.

The matrix A has upper bandwidth q where $q \geq 0$ is the smallest number such that $a_{ij} = 0$ whenever $j > i + q$ and lower bandwidth p where $p \geq 0$ is the smallest number such that $a_{ij} = 0$ whenever $i > j + p$.

Typical examples are obtained after finite element or finite difference discretizations.

Banded systems (2)

Substantial reduction of work and memory can be realized for these systems, since L and U inherit the lower and upper bandwidth of A.

This is easily checked by writing down some elimination steps for a banded system of equations.

The LU decomposition can now be obtained using $2npq$ flops if $n \gg p$ and $n \gg q$.

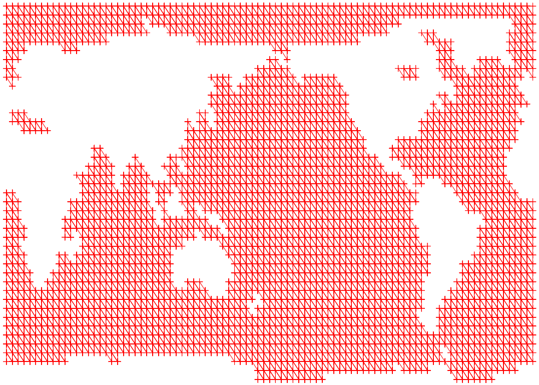
However, the band structure is to a large extend destroyed if partial pivoting is applied.

General sparse systems (1)

Large matrices with general sparsity patterns arise for example in unstructured finite element calculation.

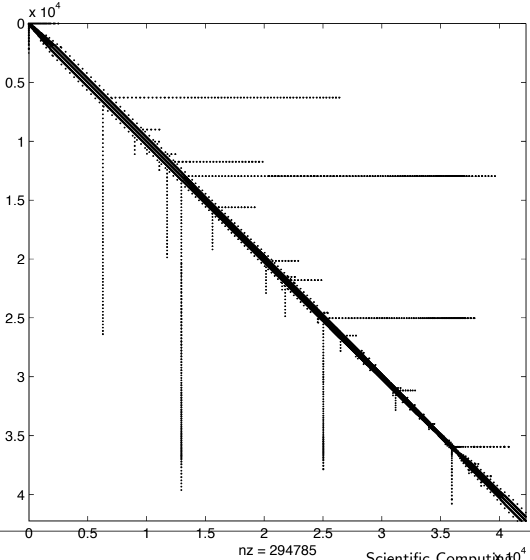
In order to limit the amount of fill in the matrix is usually reordered before it is decomposed.

Example: ocean circulation



Numerical model: discretisation with FEM

Example: Nonzero pattern



General sparse systems (2)

Solving a sparse system $Ax = b$ with a direct method normally consists of three phases:

- An analysis phase: a symbolic decomposition is made during which a suitable ordering is determined.
- A decomposition phase: the permuted matrix is decomposed.
- A solution phase: the solutions for one or more right-hand sides are determined using back and forward substitution, and back permutation to the original ordering.

Ordering algorithms

Quite sophisticated ordering algorithms exist to minimise the fill in. They are based on graph theory.
Some general principles that are used are:

- Try to minimise the bandwidth in every elimination step. An example is the Reverse Cuthill-McKee algorithm.
- Select rows with the fewest nonzeros. An example is the minimum degree ordering.

Concluding remarks

- Direct solution methods are the preferred choice for dense systems.
- Also for sparse methods they are widely used, in particular for positive definite banded systems.
- A few of the state-of-the-art solvers are: MUMPS, Super-LU, Pardiso, UMFpack (matlab \)
- For very large sparse systems iterative methods are usually preferred. However, most of the time a combination of the two is used. Iterative methods can be used to improve the solution computed with the direct method, or an approximate factorisation is used to accelerate the convergence of an iterative method. These issues will be discussed in the next lessons.