Scientific Computing

Decompositions, introduction iterative methods

Martin van Gijzen

Delft University of Technology

October 18, 2017

Program October 18

- Choleski decomposition
- Sparse matrices and reorderings
- The Power method
- Basic iterative methods for linear systems
  - Richardson's method
  - Jacobi, Gauss-Seidel and SOR
  - Iterative refinement

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 $n \gg q$.

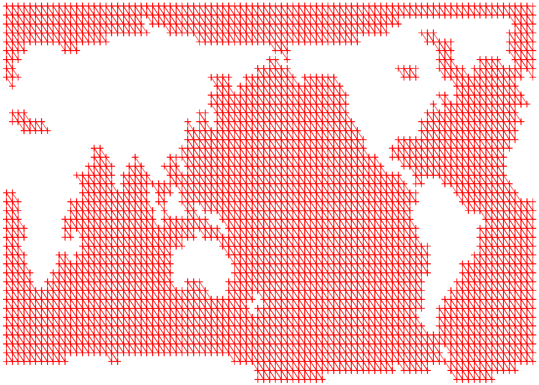
However, the band structure is to a large extend destroyed if partial pivoting is applied.

General sparse systems (1)

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

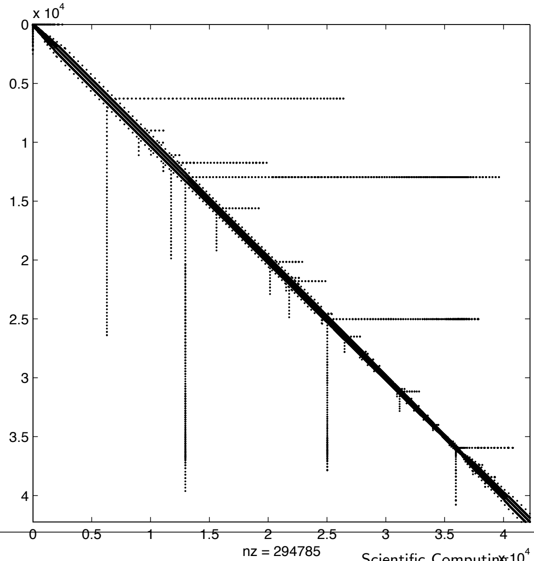
In order to limit the amount of fill in the matrix is usually reordered before it is decomposed.

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.

Some 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.

The Power method

The Power method is the classical method to compute largest eigenvalue and eigenvector of a matrix.

Multiplying with a matrix amplifies strongest the eigendirection corresponding to the (in modulus) largest eigenvalues.

Successively multiplying and scaling (to avoid overflow or underflow) yields a vector in which the direction of the largest eigenvector becomes more and more dominant.

Algorithm

The Power method

```
$q_0 \in \mathbb{C}^n$ is given
for $k = 1, 2, \ldots$
    $z_k = Aq_{k-1}$
    $q_k = z_k / \|z_k\|_2$
    $\lambda^{(k)} = q_{k-1}^H z_k$
endfor
```

It can easily be seen that if q_{k-1} is an eigenvector corresponding to λ_j then

$$\lambda^{(k)} = q_{k-1}^H A q_{k-1} = \lambda_j q_{k-1}^H q_{k-1} = \lambda_j \|q_{k-1}\|_2^2 = \lambda_j .$$

Convergence (1)

Assume that the n eigenvalues are ordered such that $|\lambda_1| > |\lambda_2| \geq ... \geq |\lambda_n|$ and the eigenvectors by $x_1, ..., x_n$ so $Ax_i = \lambda_i x_i$. Each arbitrary starting vector q_0 can be written as:

$$q_0 = a_1 x_1 + a_2 x_2 + ... + a_n x_n$$

and if $a_1 \neq 0$ it follows that

$$A^k q_0 = a_1 \lambda_1^k (x_1 + \sum_{j=2}^{n} \frac{a_j}{a_1} \left(\frac{\lambda_j}{\lambda_1} \right)^k x_j) \ .$$

Convergence (2)

Using this equality we conclude that

$$|\lambda_1 - \lambda^{(k)}| = O\left(\left|\frac{\lambda_2}{\lambda_1}\right|^k\right) \,, \quad \text{and also that}$$

the angle between q_k and x_1 is of order $|\frac{\lambda_2}{\lambda_1}|^k$.

Convergence (3)

Note that there is a problem if $|\lambda_1| = |\lambda_2|$, which is the case for instance if $\lambda_1 = \bar{\lambda}_2$. A vector q_0 which has a nonzero component in x_1 and x_2 can be written as

$$q_0 = a_1 x_1 + a_2 x_2 + \sum_{j=3}^{n} a_j x_j \ .$$

The component in the direction of $x_3, ..., x_n$ will vanish in the Power method, but q_k will not tend to a limit.

Iterative methods for linear systems

Iterative methods construct successive approximations x_k to the solution of the linear systems $Ax = b$. Here k is the iteration number, and the approximation x_k is also called the *iterate*. The vector $r_k = b - Ax_k$ is the *residual*.

The iterative methods are composed of only a few different basic operations:

- Products with the matrix A
- Vector operations (updates and inner product operations)
- *Preconditioning operations*

Preconditioning

Usually iterative methods are applied not to the original system

$$Ax = b$$

but to the preconditioned system

$$M^{-1}Ax = M^{-1}b$$

where the preconditioner is chosen such that:

- Preconditioning operations (operations with M^{-1}) are cheap;
- The iterative method converges much faster for the preconditioned system.

Basic iterative methods

The first iterative methods we will discuss are the *basic iterative methods*. Basic iterative methods only use information of the previous iteration.

Until the 70's they were quite popular. Some are still used but as preconditioners in combination with an acceleration technique. They also still play a role in multigrid techniques where they are used as smoothers.

Basic iterative methods (2)

Basic iterative methods are usually constructed using a splitting of A:

$$A = M - N.$$

Successive approximations are then computed using the iterative process

$$Mx_{k+1} = Nx_k + b$$

which is equivalent too

$$x_{k+1} = x_k + M^{-1}(b - Ax_k)$$

The vector $r_k = b - Ax_k$ is called the *residual*, and the matrix M is a *preconditioner*. The next few frames we look at $M = I$.

Richardson's method

The choice $M = I$, $N = I - A$ gives Richardson's method, which is the most simple iterative method possible.

The iterative process becomes

$$x_{k+1} = x_k + (b - Ax_k) = b + (I - A)x_k$$

Richardson's method (2)

This process yields the following iterates:
Initial guess $x_0 = 0$

$$x_1 = b$$

$$x_2 = b + (I - A)x_1 = b + (I - A)b$$

$$x_3 = b + (I - A)x_2 = b + (I - A)b + (I - A)^2 b$$

Repeating this gives

$$x_{k+1} = \sum_{i=0}^{k} (I - A)^i b$$

Richardson's method (3)

So Richardson's method generates the series expansion for $\frac{1}{1-z}$ with $z = I - A$. If this series converges we have

$$\sum_{i=0}^{\infty} (I - A)^i = A^{-1}$$

The series expansion for $\frac{1}{1-z}$ converges if $|z| < 1$. If A is diagonalizable then the series $\sum_{i=0}^{\infty} (I - A)^i$ converges if

$$|1 - \lambda| < 1$$

with λ any eigenvalue of A. For λ real this means that

$$0 < \lambda < 2$$

Richardson's method (4)

In order to increase the radius of convergence and to speed up the convergence, one can introduce a parameter α:

$$x_{k+1} = x_k + \alpha(b - Ax_k) = \alpha b + (I - \alpha A)x_k$$

It is easy to verify that if all eigenvalues are real and positive the optimal α is given by

$$\alpha_{opt} = \frac{2}{\lambda_{max} + \lambda_{min}}.$$

Richardson's method (5)

Before, we assumed for the initial guess $x_0 = 0$.
Starting with another initial guess x_0 only
means that we have to solve a shifted system

$$A(y + x_0) = b \Leftrightarrow Ay = b - Ax_0 = r_0$$

So the results obtained before remain valid, irrespective of the initial guess.

Richardson's method (6)

We want to stop once the error $\|x_k - x\| < \epsilon$, with ϵ some prescribed tolerance. Unfortunately we do not know x, so this criterion does not work in practice.
Alternatives are:

- $\|r_k\| = \|b - Ax_k\| = \|Ax - Ax_k\| < \epsilon$
  Disadvantage: criterion not scaling invariant
- $\frac{\|r_k\|}{\|r_0\|} < \epsilon$
  Disadvantage: good initial guess does not reduce the number of iterations
- $\frac{\|r_k\|}{\|b\|} < \epsilon$
  Seems best

Convergence of Basic Iterative Methods

To investigate the convergence of Basic Iterative Methods in general, we look again at the formula

$$Mx_{k+1} = Nx_k + b.$$

Remember that $A = M - N$. If we subtract $Mx = Nx + b$ from this equation we get a recursion for the error $e = x_k - x$:

$$Me_{k+1} = Ne_k$$

Convergence of Basic Iterative Methods (2)

We can also write this as

$$e_{k+1} = M^{-1} N e_k$$

This is a power iteration and hence the error will ultimately point in the direction of the largest eigenvector of $M^{-1}N$. The rate of convergence is determined by the *spectral radius* $\rho(M^{-1}N)$ of $M^{-1}N$:

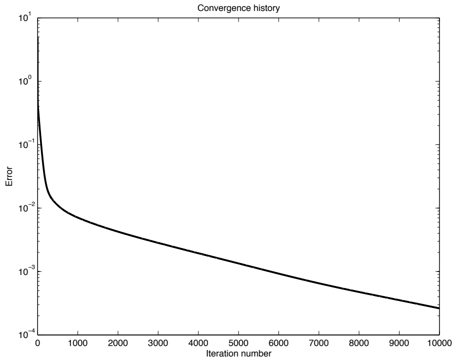
$$\rho(M^{-1}N) = |\lambda_{max}(M^{-1}N)| \ .$$

For convergence we must have that

$$\rho(M^{-1}N) < 1 \ .$$

Linear convergence

Ultimately, we have $\|e_{k+1}\| \approx \rho(M^{-1}N)\|e_k\|$, which means that we have linear convergence



Classical Basic Iterative Methods

We will now briefly discuss the three best known basic iterative methods

- Jacobi's method
- The method of Gauss-Seidel
- Successive overrelaxation

These methods can be seen as Richardson's method applied to the preconditioned system

$$M^{-1}Ax = M^{-1}b \ .$$

Jacobi's method

We first write $A = L + D + U$, with L the strictly lower triangular part of A, D the main diagonal and U the strictly upper triangular part. Jacobi's method is now defined by the choice $M = D$, $N = -L - U$. The process is given by

$$Dx_{k+1} = (-L - U)x_k + b$$

or equivalently by

$$x_{k+1} = x_k + D^{-1}(b - Ax_k)$$

The Gauss-Seidel method

We write again $A = L + D + U$. The Gauss-Seidel method is now defined by the choice $M = L + D$, $N = -U$. The process is given by

$$(L + D)x_{k+1} = -Ux_k + b$$

or equivalently by

$$x_{k+1} = x_k + (L + D)^{-1}(b - Ax_k)$$

Successive overrelaxation (SOR)

We write again $A = L + D + U$. The SOR method is now defined by the choice $M = D + \omega L$, $N = (1 - \omega)D - \omega U$. The parameter ω is called the relaxation parameter. The process is given by

$$(D + \omega L)x_{k+1} = ((1 - \omega)D - \omega U)x_k + \omega b$$

or as

$$x_{k+1} = x_k + \omega(D + \omega L)^{-1}(b - Ax_k)$$

With $\omega = 1$ we get the method of Gauss-Seidel back. In general the optimal value of ω is not known.

Iterative refinement

Before the break we discussed direct methods. For numerical stability it is necessary to perform partial pivoting. However, this goes at the expense of the efficiency.
If the LU-factors are inaccurate, such that $A = LU - N$, they can still be used as *preconditioner* for the process

$$x_{i+1} = x_i + (LU)^{-1}(b - Ax_i)$$

This is called *iterative refinement* and is used to improve the accuracy of the direct solution.

Concluding remarks

During this lesson we discussed so called Basic Iterative Methods, also called Stationary Iterative Methods.

They only use information from the previous iteration.

In the next lessons we will see methods that use information form all previous iterations to find optimal solutions.