# 11. Methods for Sparse Matrices.

NUMERICAL ANALYSIS. Prof. Y. Nishidate (323-B, nisidate@u-aizu.ac.jp)
http://web-int.u-aizu.ac.jp/~nisidate/na/

## Introduction

Numerical methods for partial differential equations (such as the finite element method and the finite difference method) often lead to the equation systems which matrices can be characterized as *symmetric*, *positive definite* and *sparse*.

A square matrix $\mathbf{A}$ is **symmetric** if it is symmetric about the main diagonal, i.e. $a_{ij} = a_{ji}$ for all $i$ and $j$. A symmteric matrix is equal to its own transpose $\mathbf{A}^T = \mathbf{A}$. For symmetric matrix only symmetric part of the matrix plus coefficients of the main diagonal can be stored in computer memory.

A matrix is called **sparse** if only a small proportion of its elements are non-zero. In some cases only non-zero elements of the sparse matrix can be stored and used during system solution (for example, if iterative methods are used for the solution).

A square matrix $\mathbf{A}$ is **positive definite** if its *quadratic form* $\mathbf{x}^T\mathbf{A}\mathbf{x}$ is positive for all real non-null vectors $\mathbf{x}$. Solution of equation systems with positive definite matrices is easier because it is usually not necessary to use pivoting during matrix decomposition.

## Matrix Storage

**Symmetric band storage.** A symmetric matrix with nonzero coefficients grouped around the main diagonal is shown below (only coefficients which should be stored are depicted):

$$\begin{bmatrix} \mathrm{x} & \mathrm{x} & \mathrm{x} & & & \\ & \mathrm{x} & \mathrm{x} & \mathrm{x} & & \\ & & \mathrm{x} & \mathrm{x} & \mathrm{x} & \\ & & & \mathrm{x} & \mathrm{x} & \mathrm{x} \\ & & & & \mathrm{x} & \mathrm{x} \\ & & & & & \mathrm{x} \end{bmatrix}$$

Matrix coefficients can be stored as a two-dimensional or one-dimensional array by rows or columns. A *bandwidth* value $h$ is also should be memorized. It is necessary to take into account that several last rows or several first columns has variable length.

To simplify storage by columns it is possible to store all columns with the same height adding necessary number of zeroes to the first $h-1$ columns.

For example, the matrix shown above can be stored as the following array containing columns of constant length:

$$A = \{0, 0, a_{11}, 0, a_{12}, a_{22}, a_{13}, a_{23}, a_{33}, a_{24}, a_{34}, a_{44}, a_{35}, a_{45}, a_{55}, a_{46}, a_{56}, a_{66}\} \qquad h = 3$$

**Symmetric profile storage.** In many practical cases matrices have variable bandwidth with big bandwidth only for certain small amount of rows or columns. In such cases the *symmetrical profile* storage is more efficient in terms of both memory and operation count.

$$\begin{bmatrix} \mathrm{x} & & \mathrm{x} & & \mathrm{x} & \\ & \mathrm{x} & \mathrm{x} & & \mathrm{x} & \\ & & \mathrm{x} & \mathrm{x} & \mathrm{x} & \\ & & & \mathrm{x} & \mathrm{x} & \mathrm{x} \\ & & & & \mathrm{x} & \mathrm{x} \\ & & & & & \mathrm{x} \end{bmatrix}$$

Symmetric global stiffness matrix $[A] = a_{ij}$ of the order $n$ is stored by columns. Each column starts from the first top nonzero element and ends at the diagonal element. The matrix is represented by two arrays:

1) real array $A$, containing matrix elements column by column;
2) integer pointer array $pcol$.

The $i$th element of $pcol$ contains the address of the first column element minus one. The length of the $i$th column is given by $pcol[i+1] - pcol[i]$. The length of the array $A$ is equal to $pcol[n+1]$ (assuming that array indices begin from 1). Proper node ordering can decrease the matrix profile significantly.

For example, the matrix shown above can be stored as the following two arrays:

$$A = \{a_{11}, a_{22}, a_{13}, a_{23}, a_{33}, a_{34}, a_{44}, a_{15}, a_{25}, a_{35}, a_{45}, a_{55}, a_{46}, a_{56}, a_{66}\}$$
$$pcol = \{0, 1, 2, 5, 7, 12, 15\}$$

**Sparse row storage.** The sparse row-wise format is a suitable storage scheme for iterative methods of solution of equation systems.

$$\begin{bmatrix} \mathrm{x} & & \mathrm{x} & \mathrm{x} & & \\ & \mathrm{x} & & & & \\ \mathrm{x} & & \mathrm{x} & & \mathrm{x} & \\ \mathrm{x} & & & \mathrm{x} & & \\ & & \mathrm{x} & & \mathrm{x} & \\ & & & & & \mathrm{x} \end{bmatrix}$$

In this scheme, the values of nonzero entries of matrix $\mathbf{A}$ are stored by rows along with their corresponding column indices; additional array points to the beginning of each row. Thus the matrix in the sparse row format is represented by the following three arrays:

$A[prow[n+1]]$ = array containing nonzero entries of the matrix;
$col[prow[n+1]]$ = column numbers for nonzero entries;
$prow[n+1]$ = pointers to the beginning of each row.

## Direct Solution Methods

Direct solution methods are based on algorithms, which provide solution of a linear equation system with a number of operations known in advance.

**LDU method.** LDU decomposition of a symmetric matrix is widely used in computational practice. Solution of symmetric linear algebraic system $[A]\{x\} = \{b\}$ consists of three stages:

| | |
|---|---|
| Factorization: | $[A] = [U]^T[D][U]$ |
| Forward reduction: | $\{y\} = [U]^{-T}\{b\}$ |
| Back substitution: | $\{x\} = [U]^{-1}[D]^{-1}\{y\}$ |

Consider two implementations of the LDU method for the solution of symmetric equation systems with a matrix stored in band column format and in profile format.

**LDU method with band storage by columns.** The algorithm of LDU factorization for upper symmetric part of the band can be illustrated with the following pseudo-code.

```
LDU factorization
do $j = 2, n$
    do $i = max(j - h1, 1), j - 1$
        do $k = max(j - h1, 1), i - 1$
            $A_{ij} = A_{ij} - A_{ki}A_{kj}$
        end do
    end do
    do $i = max(j - h1, 1), j - 1$
        $w = A_{ij}$
        $A_{ij} = A_{ij}/A_{ii}$
        $A_{jj} = A_{jj} - wA_{ij}$
    end do
end do
```

While all computations are done inside symmetric part of the matrix band the subscripts for coefficients are given as for the full matrix $A_{ij}$. The following correspondence exists between two-index notation $A_{ij}$ and the elements of one-dimensional array $a$ where columns of constant height stored:

$$A_{ij} \to a[i + (h-1)j - 1]$$

Here $h$ is a matrix bandwidth. Forward reduction and back substitution for the right-hand side can be described by the following algorithm:

```
do $j = 2, n$
    do $i = max(j - h1, 1), j - 1$
        $b_j = b_j - A_{ij} b_i$
    end do
end do
do $j = 1, n$
    $b_j = b_j / A_{jj}$
end do
do $j = n, 1, -1$
    do $i = max(j - h1, 1), j - 1$
        $b_i = b_i - A_{ij} b_j$
    end do
end do
```

**LDU method with profile storage.** The matrix in profile format is represented by two arrays: real array $a$, containing matrix elements and integer pointer array $pcol$.

We are going to present algorithms in full matrix notation $a_{ij}$. Then, it is necessary to have relations between two-index notation for the global stiffness matrix $A_{ij}$ and array $a$ used in computer codes. The location of the first nonzero element in the $i$th column of the matrix is given by the following function:

$$FN(i) = i - (pcol[i+1] - pcol[i]) + 1.$$

The following correspondence relations can be easily obtained for a transition from two-index notation to the notation for a one-dimensional array $a$:

$$A_{ij} \to a[i - 1 + pcol[j] - j]$$

The left-looking algorithm of factorization of a symmetric profile matrix and the algorithm for forward reduction and back substitution are given below:

**LDU factorization**

```
do $j = 2, n$
    do $i = FN(j), j - 1$
        do $k = max(FN(i), FN(j)), i - 1$
            $A_{ij} = A_{ij} - A_{ki} A_{kj}$
        end do
    end do
    do $i = FN(j), j - 1$
        $w = A_{ij}$
        $A_{ij} = A_{ij} / A_{ii}$
        $A_{jj} = A_{jj} - w A_{ij}$
    end do
end do
```

**Forward reduction and back substitution**

```
do $j = 2, n$
    do $i = FN(j), j - 1$
        $b_j = b_j - A_{ij} b_i$
    end do
end do

do $j = 1, n$
    $b_j = b_j / A_{jj}$
end do
do $i = n, 1, -1$
    do $i = FN(j), j - 1$
        $b_i = b_i - A_{ij} b_j$
    end do
end do
```

## Iterative Methods

**Gauss-Seidel iteration.** Let the system $[A]\{x\} = \{b\}$ with $n$ unknowns is given. To start iteration process, it is possible to select

$$x_i^{(0)} = b_i / a_{ii}$$

as an initial approximation of the solution. If $x_i^{(k)}$ is the value of unknown $x_i$ after $k$ iterations the next approximation for $x_i$ is

$$x_i^{(k+1)} = \left( b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) / a_{ii}$$

For each iteration $r_i = x_i^{(k+1)} - x_i^{(k)}$ differences can be calculated and the iteration process is terminated when the error $\max |r_i| < \varepsilon$ is true. Sometimes speed of the iteration process can be improved by applying successive over-relaxation (SOR) method. The SOR method differs from Gauss-Seidel iteration only by multiplication of $x_i^{(k+1)} - x_i^{(k)}$ by over-relaxation parameter $\omega$, giving

$$y_i^{(k+1)} = \left( b_i - \textstyle\sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) / a_{ii}$$
$$x_i^{(k+1)} = x_i^{(k)} + \omega (y_i^{(k+1)} - x_i^{(k)})$$

The optimal value of the over-relaxation parameter lies in the range $1 < \omega < 2$.

**Preconditioned conjugate gradient method.** A simple and efficient iterative method widely used for the solution of sparse systems is the conjugate gradient (CG) method. In many cases the convergence rate of CG method can be too slow for practical purposes. The convergence rate can be considerably improved by using preconditioning of the equation system:

$$[M]^{-1}[A]x = [M]^{-1}\{b\}$$

where $[M]^{-1}$ is the preconditioning matrix which in some sense approximates $[A]^{-1}$. The simplest preconditioning is diagonal preconditioning, in which $[M]$ contains only diagonal entries of the matrix $[A]$. Typical algorithm of PCG method can be presented as the following sequence of computations:

**PCG algorithm**

```
Compute $[M]$
$\{x_0\} = 0$
$\{r_0\} = \{b\}$
do $i = 0, 1 ...$
    $\{w_i\} = [M]^{-1}\{r_i\}$
    $\gamma_i = \{r_i\}^T \{w_i\}$
    if $i = 0$ $\{p_i\} = \{w_i\}$
    else $\{p_i\} = \{w_i\} + (\gamma_i / \gamma_{i-1})\{p_{i-1}\}$
    $\{w_i\} = [A]\{p_i\}$
    $\beta_i = \{p_i\}^T \{w_i\}$
    $\{x_i\} = \{x_{i-1}\} + (\gamma_i / \beta_i)\{p_i\}$
    $\{r_i\} = \{r_{i-1}\} - (\gamma_i / \beta_i)\{p_i\}$
    if $\gamma_i / \gamma_0 < \varepsilon$ exit
end do
```

In the above PCG algorithm, matrix $[A]$ is not changed during computations. This means that no additional fill arise in the solution process. Because of this, the sparse row-wise format is an efficient storage scheme for PCG iterative method.

The following pseudocode explains matrix-vector multiplication $\{w\} = [A]\{p\}$ when matrix $[A]$ is stored in the sparse row-wise format:

**Matrix-vector multiplication in sparse-row format**

```
do $j = 1, N$
    $w[j] = 0$
    do $i = prow[j], prow[j+1] - 1$
        $w[j] = w[j] + A[i] * p[col[i]]$
    end do
end do
```