Tri-Diagonal and Band Diagonal Matrices

Stephen D. Butalla & Valerie Kobzarenko

September 26, 2019

CSE-5400
Florida Institute of Technology

Tridiagonal Systems

- Tridiagonal matrix: a matrix with three diagonal bands of non-zero elements (one above, one below, and one on the main diagonal)

$$\begin{pmatrix} b_0 & c_0 & 0 & \cdots & 0 & 0 \\ a_1 & b_1 & c_1 & \cdots & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \ddots & 0 & a_{N-2} & b_{N-2} & c_{N-2} \\ 0 & \cdots & 0 & 0 & a_{N-1} & b_{N-1} \end{pmatrix}$$

- If the matrix has only one sub and super-diagonal (and main diagonal) then it is a tridiagonal matrix
- Number of super-diagonals is called upper bandwidth
- Number of sub-diagonals is called lower bandwidth
- Total number of diagonals is the bandwidth

Tridiagonal Systems

- This system of equations is solved in $\mathcal{O}(N)$ operations by forward/back-substitution
- Algorithm stores the three nonzero bands as 3 vectors
- Note: algorithm `tridag` from [1] does not use pivoting
- This can be an issue when dividing by extremely small numbers or if a zero pivot is encountered
- However, if

$$|b_k| > |a_k| + |c_k|, \quad k \in \{0, 1, \cdots, N-1\}$$

  is satisfied, a zero pivot cannot be encountered (recall that b_{jk} are located in the $j = k$ positions)
- An instability resulting from a zero (or very small) pivot is rarely encountered in practice

Tridiagonal Systems

- We are trying to solve the set of equations given by

$$\begin{pmatrix} b_0 & c_0 & 0 & \cdots & 0 & 0 \\ a_1 & b_1 & c_1 & \cdots & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \ddots & 0 & a_{N-2} & b_{N-2} & c_{N-2} \\ 0 & \cdots & 0 & 0 & a_{N-1} & b_{N-1} \end{pmatrix} = \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-2} \\ u_{N-1} \end{pmatrix} = \begin{pmatrix} r_0 \\ r_1 \\ \vdots \\ r_{N-2} \\ r_{N-1} \end{pmatrix}$$

- Given three input vectors for the three non-zero diagonal bands in the coefficient matrix, the system is solved via the following method:

$$\beta_i = b_i - a_i \gamma_i$$

$$\gamma_i = \frac{c_{i-1}}{\beta_i}$$

$$u_i = \frac{1}{\beta_i}(r_i - a_i u_{i-1})$$

Tridiagonal Algorithm

a, **b**, **c** are the three non-zero coefficient matrix vectors

u, **r** are the variable and RHS vectors, respectively

```
Int j,n=a.size();
Doub bet;
VecDoub gam(n); //intermediate results
//main diagonal must be full
if (b[0] == 0.0) throw("Error 1 in tridag");
  u[0]=r[0]/(bet=b[0]);
  for (j=1;j<n;j++) {//decomp and forward substitution
    gam[j]=c[j-1]/bet;
        bet=b[j]-a[j]*gam[j];
        if (bet == 0.0) throw("Error 2 in tridag");
        u[j]=(r[j]-a[j]*u[j-1])/bet;
  }
for (j=(n-2);j>=0;j--)//"back substitution"
  u[j] -= gam[j+1]*u[j+1];
```

Tridiagonal Example

- We are trying to solve the set of equations given by

$$\begin{pmatrix} 1 & 2 & 0 & 0 \\ 3 & 4 & 5 & 0 \\ 0 & 6 & 7 & 8 \\ 0 & 0 & 9 & 10 \end{pmatrix} = \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 6 \\ 8 \end{pmatrix}$$

- The three coefficient vectors are

$$\mathbf{a} = \begin{pmatrix} 0 \\ 3 \\ 6 \\ 9 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 1 \\ 4 \\ 7 \\ 10 \end{pmatrix}, \quad \mathbf{c} = \begin{pmatrix} 2 \\ 5 \\ 8 \\ 0 \end{pmatrix}$$

- Intermediate expressions

$$\gamma_i = \frac{c_{i-1}}{\beta_i}$$

$$\beta_i = b_i - a_i \gamma_i$$

$$u_i = \frac{1}{\beta_i}(r_i - a_i u_{i-1})$$

- Final backsubstitution expression:

$$u_i = u_i - \gamma_{i+1} u_{i+1}$$

Tridiagonal Example

$$\mathbf{a} = \begin{pmatrix} 0 \\ 3 \\ 6 \\ 9 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 1 \\ 4 \\ 7 \\ 10 \end{pmatrix}, \quad \mathbf{c} = \begin{pmatrix} 2 \\ 5 \\ 8 \\ 0 \end{pmatrix}, \quad \mathbf{r} = \begin{pmatrix} 2 \\ 4 \\ 6 \\ 8 \end{pmatrix}$$

- Implementing the algorithm, we find

$$\gamma_0 = \frac{c_{-1}}{\beta_0} = 0$$

$$\beta_0 = b_0 - a_0\gamma_0 = b_0 = 1$$

$$u_0 = \frac{1}{\beta_0}(r_0 - a_0 u_{-1}) = \frac{r_0}{\beta_0} = 2$$

Tridiagonal Example

$$\gamma_i = \frac{c_{i-1}}{\beta_i}, \qquad \beta_i = b_i - a_i\gamma_i, \qquad u_i = \frac{1}{\beta_i}(r_i - a_i u_{i-1})$$

- Etc.,

$$\gamma_1 = 2, \quad \beta_1 = -2 \quad u_1 = 1$$
$$\gamma_2 = -2.5, \quad \beta_2 = 22 \quad u_2 = 0$$
$$\gamma_3 = 0.363636, \quad \beta_3 = 6.72727 \quad u_3 = 1.18919$$

- Finally, "back substitute":

$$u_i = u_i - \gamma_{i+1} u_{i+1}$$

So,

$$u_3 = u_3 - \gamma_4 u_4 = u_3 = 1.18919$$
$$u_2 = u_2 - \gamma_3 u_3 = 0 - 0.363636(1.18919) = -0.432432$$
$$u_1 = u_1 - \gamma_2 u_2 = -0.0810811$$
$$u_0 = u_0 - \gamma_1 u_1 = 2.16216$$

Band-Diagonal Systems

- Band-diagonal are tridiagonal systems with either a diagonal band above, below the main three, or both
- Definition: A matrix $\mathbb{A}$ is band-diagonal if [1]

$$a_{ij} = 0 \quad \text{when} \quad j > i + m_2 \quad \text{or} \quad i > j + m_1$$

  where m_1 are the number of subdiagonal bands, and m_2 are the number of elements of super diagonal bands
- To save storage space, band-diagonal matrices can be transformed into a "compact form" $\Rightarrow$ "rotate" the matrix 45° clockwise. For a 6×6 matrix $\mathbb{A}$:

$$\mathbb{A} = \begin{pmatrix} 1 & 2 & 3 & 0 & 0 & 0 \\ 3 & 4 & 5 & 4 & 0 & 0 \\ 6 & 7 & 8 & 9 & 5 & 0 \\ 0 & 10 & 11 & 12 & 13 & 2 \\ 0 & 0 & 14 & 15 & 16 & 1 \\ 0 & 0 & 0 & 15 & 16 & 3 \end{pmatrix}$$

  For this matrix: $m_1 = 2$ and $m_2 = 2$
- The new matrix will have $N = 6$ rows and $m_1 + m_2 + 1 = 5$ columns

Band-diagonal Systems

- New "rotated" matrix

$$\mathbb{A} = \begin{pmatrix} 1 & 2 & 3 & 0 & 0 & 0 \\ 3 & 4 & 5 & 4 & 0 & 0 \\ 6 & 7 & 8 & 9 & 5 & 0 \\ 0 & 10 & 11 & 12 & 13 & 2 \\ 0 & 0 & 14 & 15 & 16 & 1 \\ 0 & 0 & 0 & 15 & 16 & 3 \end{pmatrix} \quad \Rightarrow \quad \mathbb{A}' = \begin{pmatrix} x & x & 1 & 2 & 3 \\ x & 3 & 4 & 5 & 4 \\ 6 & 7 & 8 & 9 & 5 \\ 10 & 11 & 12 & 13 & 2 \\ 14 & 15 & 16 & 1 & x \\ 15 & 16 & 3 & x & x \end{pmatrix}$$

where the xs are free space (not referenced by the algorithm)

Band-Diagonal Algorithm

- Given a set of linear equations:

$$\mathbb{A}\mathbf{x} = \mathbf{b}$$

Construct your object with the "rotated," compact version of $\mathbb{A}$:

```
Banddec bandD = Banddec(compactA);
```

- And solve by the same algorithm as LU decomposition:

```
bandD.solve(RHSvec, solnVec)
```

- Code example

References

[1] W. Press, S. Teukolsky, W. Vetterling, & B. Flannery, *Numerical Recipes: The Art of Scientific Computing*, 2007, Cambridge University Press.