Tridiagonal matrix algorithm

In numerical linear algebra, the **tridiagonal matrix algorithm**, also known as the **Thomas algorithm** (named after Llewellyn Thomas), is a simplified form of Gaussian elimination that can be used to solve tridiagonal systems of equations. A tridiagonal system for n unknowns may be written as

$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i,$$

where $a_1 = 0$ and $c_n = 0$.

$$\begin{bmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{bmatrix}.$$

For such systems, the solution can be obtained in $O(n)$ operations instead of $O(n^3)$ required by Gaussian elimination. A first sweep eliminates the a_i's, and then an (abbreviated) backward substitution produces the solution. Examples of such matrices commonly arise from the discretization of 1D Poisson equation and natural cubic spline interpolation.

Thomas' algorithm is not stable in general, but is so in several special cases, such as when the matrix is diagonally dominant (either by rows or columns) or symmetric positive definite;[1][2] for a more precise characterization of stability of Thomas' algorithm, see Higham Theorem 9.12.[3] If stability is required in the general case, Gaussian elimination with partial pivoting (GEPP) is recommended instead.[2]

Contents

Method

The forward sweep consists of the computation of new coefficients as follows, denoting the new coefficients with primes:

$$c'_i = \begin{cases} \dfrac{c_i}{b_i}, & i = 1, \\ \dfrac{c_i}{b_i - a_i c'_{i-1}}, & i = 2, 3, \ldots, n-1 \end{cases}$$

and

$$d'_i = \begin{cases} \dfrac{d_i}{b_i}, & i = 1, \\ \dfrac{d_i - a_i d'_{i-1}}{b_i - a_i c'_{i-1}}, & i = 2, 3, \ldots, n. \end{cases}$$

The solution is then obtained by back substitution:

$$x_n = d'_n,$$

$$x_i = d'_i - c'_i x_{i+1}, \quad i = n-1, n-2, \ldots, 1.$$

The method above does not modify the original coefficient vectors, but must also keep track of the new coefficients. If the coefficient vectors may be modified, then an algorithm with less bookkeeping is:

For $i = 2, 3, \ldots, n$, do

$$\begin{aligned} w &= \frac{a_i}{b_{i-1}}, \\ b_i &:= b_i - w c_{i-1}, \\ d_i &:= d_i - w d_{i-1}, \end{aligned}$$

followed by the back substitution

$$\begin{aligned} x_n &= \frac{d_n}{b_n}, \\ x_i &= \frac{d_i - c_i x_{i+1}}{b_i} \quad \text{for } i = n-1, n-2, \ldots, 1. \end{aligned}$$

The implementation in a VBA subroutine without preserving the coefficient vectors:

```
Sub TriDiagonal_Matrix_Algorithm(N%, A#(), B#(), C#(), D#(), X#())
    Dim i%, W#
    For i = 2 To N
        W = A(i) / B(i - 1)
        B(i) = B(i) - W * C(i - 1)
        D(i) = D(i) - W * D(i - 1)
    Next i
    X(N) = D(N) / B(N)
    For i = N - 1 To 1 Step -1
        X(i) = (D(i) - C(i) * X(i + 1)) / B(i)
    Next i
End Sub
```

Derivation

The derivation of the tridiagonal matrix algorithm is a special case of Gaussian elimination.

Suppose that the unknowns are $x_1, \ldots, x_n$, and that the equations to be solved are:

$$\begin{aligned} b_1 x_1 \quad + c_1 x_2 \quad &= d_1 \\ a_i x_{i-1} + b_i x_i + c_i x_{i+1} &= d_i\,, \quad i = 2, \ldots, n-1 \\ a_n x_{n-1} + b_n x_n \quad &= d_n\,. \end{aligned}$$

Consider modifying the second ($i = 2$) equation with the first equation as follows:

$$(\text{equation } 2) \cdot b_1 - (\text{equation } 1) \cdot a_2$$

which would give:

$$(b_2 b_1 - c_1 a_2) x_2 + c_2 b_1 x_3 = d_2 b_1 - d_1 a_2.$$

Note that x_1 has been eliminated from the second equation. Using a similar tactic with the **modified** second equation on the third equation yields:

$$(b_3(b_2 b_1 - c_1 a_2) - c_2 b_1 a_3) x_3 + c_3(b_2 b_1 - c_1 a_2) x_4 = d_3(b_2 b_1 - c_1 a_2) - (d_2 b_1 - d_1 a_2) a_3.$$

This time x_2 was eliminated. If this procedure is repeated until the n^{th} row; the (modified) n^{th} equation will involve only one unknown, x_n. This may be solved for and then used to solve the $(n-1)^{th}$ equation, and so on until all of the unknowns are solved for.

Clearly, the coefficients on the modified equations get more and more complicated if stated explicitly. By examining the procedure, the modified coefficients (notated with tildes) may instead be defined recursively:

$$\tilde{a}_i = 0$$

$$\begin{aligned}\tilde{b}_1 &= b_1 \\ \tilde{b}_i &= b_i \tilde{b}_{i-1} - \tilde{c}_{i-1} a_i\end{aligned}$$

$$\begin{aligned}\tilde{c}_1 &= c_1 \\ \tilde{c}_i &= c_i \tilde{b}_{i-1}\end{aligned}$$

$$\begin{aligned}\tilde{d}_1 &= d_1 \\ \tilde{d}_i &= d_i \tilde{b}_{i-1} - \tilde{d}_{i-1} a_i .\end{aligned}$$

To further hasten the solution process, $\tilde{b}_i$ may be divided out (if there's no division by zero risk), the newer modified coefficients, each notated with a prime, will be:

$$a'_i = 0$$

$$b'_i = 1$$

$$\begin{aligned}c'_1 &= \frac{c_1}{b_1} \\ c'_i &= \frac{c_i}{b_i - c'_{i-1} a_i}\end{aligned}$$

$$\begin{aligned}d'_1 &= \frac{d_1}{b_1} \\ d'_i &= \frac{d_i - d'_{i-1} a_i}{b_i - c'_{i-1} a_i}.\end{aligned}$$

This gives the following system with the same unknowns and coefficients defined in terms of the original ones above:

$$\begin{aligned}x_i + c'_i x_{i+1} &= d'_i \quad ; \quad i = 1, \ldots, n-1 \\ x_n &= d'_n \quad ; \quad i = n.\end{aligned}$$

The last equation involves only one unknown. Solving it in turn reduces the next last equation to one unknown, so that this backward substitution can be used to find all of the unknowns:

$$x_n = d'_n$$

$$x_i = d'_i - c'_i x_{i+1} \quad ; \; i = n-1, n-2, \ldots, 1.$$

Variants

In some situations, particularly those involving periodic boundary conditions, a slightly perturbed form of the tridiagonal system may need to be solved:

$$\begin{aligned}a_1 x_n \quad + b_1 x_1 \; + c_1 x_2 \quad &= d_1 \\ a_i x_{i-1} \; + b_i x_i \; + c_i x_{i+1} &= d_i \,, \quad i = 2, \ldots, n-1 \\ a_n x_{n-1} + b_n x_n + c_n x_1 \quad &= d_n \,.\end{aligned}$$

In this case, we can make use of the Sherman–Morrison formula to avoid the additional operations of Gaussian elimination and still use the Thomas algorithm. The method requires solving a modified non-cyclic version of the system for both the input and a sparse corrective vector, and then combining the solutions. This can be done efficiently if both solutions are computed at once, as the forward portion of the pure tridiagonal matrix algorithm can be shared.

If we indicate by:

$$A = \begin{bmatrix} b_1 & c_1 & & & a_1 \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ c_n & & & a_n & b_n \end{bmatrix}, x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}, d = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{bmatrix}$$

Then the system to be solved is:

$$Ax = d$$

In this case the coefficients a_1 and c_n are, generally speaking, non-zero, so their presence does not allow to apply the Thomas algorithm directly. We can therefore consider $B \in \mathbb{R}^{n \times n}$ and $u, v \in \mathbb{R}^n$ as following:

$$B = \begin{bmatrix} b_1 - \gamma & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_n & b_n - \frac{c_n a_1}{\gamma} \end{bmatrix}, u = \begin{bmatrix} \gamma \\ 0 \\ 0 \\ \vdots \\ c_n \end{bmatrix}, v = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ a_1/\gamma \end{bmatrix}.$$

Where $\gamma \in \mathbb{R}$ is a parameter to be chosen. The matrix A can be reconstructed as $A = B + uv^T$. The solution is then obtained in the following way:[4] first we solve two tridiagonal systems of equations applying the Thomas algorithm:

$$By = d \qquad\qquad Bq = u$$

Then we reconstruct the solution x using the Shermann-Morrison formula:

$$x = A^{-1}d = (B + uv^T)^{-1}d = B^{-1}d - \frac{B^{-1}uv^TB^{-1}}{1 + v^TB^{-1}u}d = y - \frac{qv^Ty}{1 + v^Tq}$$

The implementation in Dev-C++ without preserving the coefficient vectors:

```
typedef struct{
    double A[n+2];
    double B[n+2];
    double C[n+2];
    double D[n+2];
} COEFFICIENTS;

//Apply Thomas Alg., unknowns x[1],...,x[n]
void ThomasAlg(double x[n+1],COEFFICIENTS* coeff){
    double u[n+1]={},v[n+1]={};
    double y[n+1]={},q[n+1]={};
    double* A=coeff->A, *B=coeff->B,*C=coeff->C,*D=coeff->D;
    double Value=0;
    double w;
    int i;
```

```
    u[1]=gamma;
    u[n]=C[n];

    v[1]=1;
    v[n]=A[1]/gamma;

    //create matrix B
    A[1]=0;
    B[1]=B[1]-gamma;
    B[n]=B[n]-(C[n]*A[n])/gamma;
    C[n]=0;

    for(i=2;i<n+1;i++){
        w=A[i]/B[i-1];
        B[i]=B[i]-w*C[i-1];
        D[i]=D[i]-w*D[i-1];
        u[i]=u[i]-w*u[i-1];
    }
    y[n]=D[n]/B[n];
    q[n]=u[n]/B[n];

    for(i=n-1;i>0;i--){
        y[i]=(D[i]-C[i]*y[i+1])/B[i];
        q[i]=(u[i]-C[i]*q[i+1])/B[i];
    }
    Value=(v[1]*y[1]+v[n]*y[n])/(1+v[1]*q[1]+v[n]*q[n]);

    for(i=1;i<n+1;i++){
        x[i]=y[i]-q[i]*Value;
    }
}
```

There is also another way to solve the slightly perturbed form of the tridiagonal system considered above.[5] Let us consider two auxiliary linear systems of dimension $(n-1)\times(n-1)$:

$$\begin{aligned} b_2u_2+c_2u_3 &= d_2 \\ a_3u_2+b_3u_3+c_3u_4 &= d_3 \\ a_iu_{i-1}+b_iu_i+c_iu_{i+1} &= d_i \qquad i=4,\dots,n-1 \\ &\dots \\ a_nu_{n-1}+b_nu_n &= d_n\,. \end{aligned} \qquad\qquad \begin{aligned} b_2v_2+c_2v_3 &= -a_2 \\ a_3v_2+b_3v_3+c_3v_4 &= 0 \\ a_iu_{i-1}+b_iu_i+c_iu_{i+1} &= 0 \qquad i=4,\dots,n-1 \\ &\dots \\ a_nv_{n-1}+b_nv_n &= -c_n\,. \end{aligned}$$

For convenience, we additional define $u_1=0$ and $v_1=1$. We can now find the solutions $\{u_2,u_3\dots,u_n\}$ and $\{v_2,v_3\dots,v_n\}$ applying Thomas algorithm to the two auxiliary tridiagonal system.

The solution $\{x_1,x_2\dots,x_n\}$ can be then represented in the form:

$$x_i=u_i+x_1v_i \qquad i=1,2,\dots,n$$

Indeed, multiplying each equation of the second auxiliary system by x_1, adding with the corresponding equation of the first auxiliary system and using the representation $x_i=u_i+x_1v_i$, we immediately see that equations number 2 through n of the original system are satisfied; it only remains to satisfy equation number 1. To do so, consider formula for $i=2$ and $i=n$ and substitute $x_2=u_2+x_1v_2$and $x_n=u_n+x_1v_n$ into the first equation of the original system. This yields one scalar equation for x_1:

$$b_1x_1+c_1(u_2+x_1v_2)+a_1(u_n+x_1v_n)=d_1$$

As such, we find:

$$x_1=\frac{d_1-a_1u_n-c_1u_2}{b_1+a_1v_n+c_1v_2}$$

The implementation in Dev-C++ without preserving the coefficient vectors:

```
typedef struct{
    double A[n+2];
    double B[n+2];
```

```
    double C[n+2];
    double D[n+2];
} COEFFICIENTS;

//Apply Thomas Alg., unknowns x[1],...,x[n]
void ThomasAlg(double x[n+1],COEFFICIENTS* coeff){
    double u[n+1]={},v[n+1]={};
    double* A=coeff->A, *B=coeff->B,*C=coeff->C,*D=coeff->D;
    double w,F[n+1]={};
    F[2]=-A[2];
    F[n]=-C[n];
    int i;
    u[1]=0;
    v[1]=1;
    for(i=3;i<n+1;i++){
        w=A[i]/B[i-1];
        B[i]=B[i]-w*C[i-1];
        D[i]=D[i]-w*D[i-1];
        F[i]=F[i]-w*F[i-1];
    }
    u[n]=D[n]/B[n];
    v[n]=F[n]/B[n];
    for(i=n-1;i>1;i--){
        u[i]=(D[i]-C[i]*u[i+1])/B[i];
        v[i]=(F[i]-C[i]*v[i+1])/B[i];
    }
    x[1]=(D[1]-A[1]*u[n]-C[1]*u[2])/(B[1]+A[1]*v[n]+C[1]*v[2]);

    for(i=2;i<n+1;i++){
        x[i]=u[i]+x[1]*v[i];
    }
}
```

In both cases the auxiliary systems to be solved are genuinely tri-diagonal, so the overall computational complexity of solving system $\boldsymbol{Ax} = \boldsymbol{d}$ remains linear with the respect to the dimension of the system $\boldsymbol{n}$, that is $\boldsymbol{O(n)}$ arithmetic operations.

In other situations, the system of equations may be **block tridiagonal** (see block matrix), with smaller submatrices arranged as the individual elements in the above matrix system (e.g., the 2D Poisson problem). Simplified forms of Gaussian elimination have been developed for these situations.[6]

The textbook *Numerical Mathematics* by Quarteroni, Sacco and Saleri, lists a modified version of the algorithm which avoids some of the divisions (using instead multiplications), which is beneficial on some computer architectures.

Parallel tridiagonal solvers have been published for many vector and parallel architectures, including GPUs[7] [8]

For an extensive treatment of parallel tridiagonal and block tridiagonal solvers see [9]

References

1. Pradip Niyogi (2006). *Introduction to Computational Fluid Dynamics*. Pearson Education India. p. 76. ISBN 978-81-7758-764-7.
2. Biswa Nath Datta (2010). *Numerical Linear Algebra and Applications, Second Edition*. SIAM. p. 162. ISBN 978-0-89871-765-5.
3. Nicholas J. Higham (2002). *Accuracy and Stability of Numerical Algorithms: Second Edition*. SIAM. p. 175. ISBN 978-0-89871-802-7.
4. Batista, Milan; Ibrahim Karawia, Abdel Rahman A. (2009). "The use of the Sherman–Morrison–Woodbury formula to solve cyclic block tri-diagonal and cyclic block penta-diagonal linear systems of equations" (https://dx.doi.org/10.1016/j.amc.2009.01.003). *Applied Mathematics and Computation*. **210** (2): 558–563. doi:10.1016/j.amc.2009.01.003 (https://doi.org/10.1016%2Fj.amc.2009.01.003). ISSN 0096-3003 (https://www.worldcat.org/issn/0096-3003).
5. Ryaben' kii, Victor S.; Tsynkov, Semyon V. (2006-11-02), "Introduction" (https://dx.doi.org/10.1201/9781420011166-1), *A Theoretical Introduction to Numerical Analysis*, Chapman and Hall/CRC, pp. 1–19, ISBN 978-0-429-14339-7, retrieved 2022-05-25
6. Quarteroni, Alfio; Sacco, Riccardo; Saleri, Fausto (2007). "Section 3.8". *Numerical Mathematics*. Springer, New York. ISBN 978-3-540-34658-6.

7. Chang, L.-W.; Hwu, W,-M. (2014). "A guide for implementing tridiagonal solvers on GPUs". In V. Kidratenko (ed.). *Numerical Computations with GPUs*. Springer. ISBN 978-3-319-06548-9.
8. Venetis, I.E.; Kouris, A.; Sobczyk, A.; Gallopoulos, E.; Sameh, A. (2015). "A direct tridiagonal solver based on Givens rotations for GPU architectures". *Parallel Computing*. **49**: 101–116. doi:10.1016/j.parco.2015.03.008 (https://doi.org/10.1016%2Fj.parco.2015.03.008).
9. Gallopoulos, E.; Philippe, B.; Sameh, A.H. (2016). "Chapter 5". *Parallelism in Matrix Computations*. Springer. ISBN 978-94-017-7188-7.

- Conte, S. D.; de Boor, C. (1972). *Elementary Numerical Analysis* (https://archive.org/details/elementarynumeri00samu). McGraw-Hill, New York. ISBN 0070124469.
- This article incorporates text from the article Tridiagonal_matrix_algorithm_-_TDMA_(Thomas_algorithm) (http://www.cfd-online.com/Wiki/Tridiagonal_matrix_algorithm_-_TDMA_(Thomas_algorithm)) on CFD-Wiki (http://www.cfd-online.com/Wiki/Main_Page) that is under the GFDL license.
- Press, W. H.; Teukolsky, S. A.; Vetterling, W. T.; Flannery, B. P. (2007). "Section 2.4" (http://apps.nrbook.com/empanel/index.html?pg=56). *Numerical Recipes: The Art of Scientific Computing* (3rd ed.). New York: Cambridge University Press. ISBN 978-0-521-88068-8.
