

On the Numerical Solution of Cyclic Tridiagonal Systems.

M. Piller
DINMA-Sezione di Fisica Tecnica
Università di Trieste, 34127 Trieste, ITALY

November 17, 1999

1 Introduction

All numerical techniques for the solution of Partial Differential Equations (PDE) involve discretization. Many times this discretization leads to the solution of a large system of linear equations, the generic of which can be represented as $Ax = D$; A is known as the *coefficient matrix*, while x is the vector of unknowns and D is the *right-hand side* vector.

It often happens that the coefficient matrix has some particular properties, such that a general-purpose solution algorithm results too expensive in comparison with more *clever* ones, which use the favorable characteristics of the matrix itself.

In the present work we deal with one of these cases. In more detail, our purpose is to develop efficient algorithms for the solution of cyclic, linear tridiagonal systems whose coefficient matrix, in the most general case, is shown below:

$$\begin{pmatrix} a_1 & c_1 & 0 & 0 & b_1 \\ b_2 & a_2 & c_2 & 0 & 0 \\ 0 & b_3 & a_3 & c_3 & 0 \\ 0 & 0 & b_4 & a_4 & c_4 \\ c_5 & 0 & 0 & b_5 & a_5 \end{pmatrix} \quad (1)$$

Sometimes, the matrix A is symmetric, and this is an other opportunity to exploit in order to obtain efficient algorithms. If this is the case, the coefficient matrix becomes:

$$\begin{pmatrix} a_1 & c_1 & 0 & 0 & b_1 \\ c_1 & a_2 & c_2 & 0 & 0 \\ 0 & c_2 & a_3 & c_3 & 0 \\ 0 & 0 & c_3 & a_4 & c_4 \\ b_1 & 0 & 0 & c_4 & a_5 \end{pmatrix} \quad (2)$$

This is the case, for example, when central-difference formulas are adopted in the discretization. Moreover, the coefficient matrix can have constant values on the main and secondary diagonals, that is:

$$\begin{pmatrix} a & b & 0 & 0 & b \\ b & a & b & 0 & 0 \\ 0 & b & a & b & 0 \\ 0 & 0 & b & a & b \\ b & 0 & 0 & b & a \end{pmatrix} \quad (3)$$

This situation arises when central differences are applied on uniform grids.

An other important property, in order to deal with a *nice* coefficient matrix, is the diagonal dominance.

An $n \times n$ matrix $A = \{a_{ij}\}$ is claimed to be *diagonally-dominant* if $\sum_{i=1, i \neq j}^n |a_{ij}| \leq |a_{ii}| \forall i$, and there is at least one i for which $\sum_{i=1, i \neq j}^n |a_{ij}| < |a_{ii}|$.

All matrices we will consider in the following enjoy this property, whose importance leads in the fact that a system with a diagonally-dominant coefficient matrix is easily solvable by means of a Gaussian-elimination technique. Furthermore, it should be remembered that most direct-methods for the solution of algebraic, linear systems are based on Gaussian elimination.

In the following we will deal with each of the preceding special cases, developing ideas found essentially in [1].

2 The most generic case: neither symmetric nor constant coefficient matrix

In the Computational Fluid Dynamics (**CFD**) community the Cyclic Thomas Algorithm (see [2], and [3] for a general discussion about several variants) is widely adopted for the solution of cyclic tridiagonal systems. If the coefficient matrix has constant values on the diagonals, it is possible to use the cyclic-reduction algorithm, which is well-suited for vector computers.

The major problem with the Thomas algorithm, and its variants [3], is that it performs the process of elimination from the beginning each time a new system has to be solved, also if the coefficient matrix remains constant.

Though quite efficient if the whole solution of a single system is considered, Thomas algorithm can't compete with methods which perform a *factorization* of the coefficient matrix when the solution of a large number of linear systems with only different right-hand sides has to be performed. The Algorithm 4 of Temperton (see [1]) allows to exploit the factorization technique, as will be seen below.

2.1 Temperton's Algorithm 4

The idea which inspired this algorithm is very simple: if one can obtain in some way the value of the first unknown x_1 (i.e. the unknown which lies in the periodic boundary) the system in the residual unknowns is simply tridiagonal, and can be solved with one of the efficient methods for this kind of problems. For example one could adopt a (not cyclic) Thomas algorithm, a LU factorization technique or a cyclic-reduction method.

In order to evaluate the first unknown we need the first row of the inverse of the coefficient matrix, and the cost of this matrix inversion can be neglected only if it has to be payed only once or very seldomly (i.e. in a typical unsteady CFD calculation which treats implicitly only the diffusion fluxes, when the time-step is changed). As we will see, however, the cost of the matrix-inversion dramatically decreases when the matrix is symmetric or it has constant coefficients.

After the first unknown has been evaluated, the *simply* tridiagonal system becomes:

$$\begin{pmatrix} a_2 & c_2 & 0 & 0 \\ b_3 & a_3 & c_3 & 0 \\ 0 & b_4 & a_4 & c_4 \\ 0 & 0 & b_5 & a_5 \end{pmatrix} \begin{pmatrix} x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} d_2 - b_2 x_1 \\ d_3 \\ d_4 \\ d_5 - c_5 x_1 \end{pmatrix} \quad (4)$$

If r_1 is the first row of A^{-1} , the first unknown x_1 is evaluated as:

$$x_1 = \sum_{i=1}^n r_i d_i \quad (5)$$

3 Symmetric Coefficients-Matrix: Temperton's Algorithm 4

The underlying idea of this algorithm was described in 2.1. In the favorable case of symmetric coefficients-matrix, though, some simplifications can be introduced, both in the *initialization* phase (inversion of the coefficients-matrix) and in the subsequent solution phase.

In fact, if the matrix A is symmetric, its inverse is symmetric too. So, in order to evaluate the first row of A^{-1} , it suffices to solve the following system:

$$A\mathbf{r}_1 = \mathbf{e}_1 \quad (6)$$

where \mathbf{r}_1 is the first column of A^{-1} and \mathbf{e}_1 is the first unit vector of the canonical base: $\mathbf{e}_1 = (1, 0, 0, \dots, 0)^t$.

For the symmetry of A^{-1} one has $\mathbf{c}_1^t = \mathbf{r}_1$, being \mathbf{r}_1 the first row of A^{-1} . For example, I choosed the Thomas algorithm in order to solve system 6. Again, this practise is convenient only if the initialization phase has to be performed seldomly.

In the solution phase the symmetry of A allows to use more specific algorithms for the solution of the *residual* $(n - 1)$ equations system. Adopting the LU factorization technique as implemented in the routines LU_PRE and LU_SOLV, the solution phase takes about $7n$ floating-point operations (flops-additions and multiplications) per right-hand side. The solution phase of the LU factorization (routine LU_SOLV) takes about $5n$ flops.

4 Constant-values coefficient matrix: Temperton's Algorithm 4

In this case, the most relevant difference with what has been described in 3 lies in the evaluation of the first unknown x_1 , which can be obtained as described below:

$$A = \begin{pmatrix} a & b & 0 & 0 & b \\ b & a & b & 0 & 0 \\ 0 & b & a & b & 0 \\ 0 & 0 & b & a & b \\ b & 0 & 0 & b & a \end{pmatrix} \quad (7)$$

(8)

$$\lambda = \frac{a}{b} \quad (9)$$

$$\alpha = \frac{-\lambda \pm \sqrt{\lambda^2 - 4}}{2} \quad (10)$$

$$\sigma = \frac{1 + \alpha^2}{\lambda(1 - \alpha^2)(1 - \alpha^n)b} \quad (11)$$

$$r_{1,i+1} = \sigma \left(\alpha^i + \alpha^{(n-i)} \right) \quad i = 0 \dots n - 1 \quad (12)$$

In the above displayed formula for α one adopts the $+$ sign if $\lambda > 2$ and the $-$ sign if $\lambda < -2$. Finally, if n is odd, one has:

$$x_1 = r_{1,1}d_1 + \sum_{i=2}^m r_{i,1} (d_i + d_{n+2-i}) \quad (13)$$

and if n is even:

$$x_1 = r_{1,1}d_1 + r_{1,m+1}d_{m+1} + \sum_{i=2}^m r_{i,1} (d_i + d_{n+2-i}) \quad (14)$$

where m is the integer part of $\frac{n+1}{2}$.

This algorithm takes about $6.5n$ flops per right-hand side.

5 Symmetric Coefficients-matrix: Evans's Algorithm

Evans's algorithm is based on the following factorization of the coefficient matrix A :

$$A = \mu Q Q^t \quad (15)$$

$$Q = \begin{pmatrix} 1 & 0 & 0 & 0 & -\alpha \\ -\alpha & 1 & 0 & 0 & 0 \\ 0 & -\alpha & 1 & 0 & 0 \\ 0 & 0 & -\alpha & 1 & 0 \\ 0 & 0 & 0 & -\alpha & 1 \end{pmatrix} \quad (16)$$

$$\mu = \frac{b\lambda}{1 + \alpha^2} \quad (17)$$

The values of λ and α in the preceding formulas have already been given by equations 10 and 9.

The solution of the original system $Ax = D$ is decoupled in the solution of the following *almost bidiagonal* systems:

$$Q y = \mu^{-1} D \quad (18)$$

$$Q^t x = y \quad (19)$$

System 18 is solved through the following steps:

1. $y_1 = \frac{b_1 + \alpha^{n-1}b_2 + \alpha^{n-2}b_3 + \dots + \alpha^2b_{n-1} + \alpha b_n}{\mu(1 - \alpha^n)}$
2. $y_i = \frac{b_i}{\mu} + \alpha y_{i-1} \quad i = 2 \dots n$

Some remarks:

- It is convenient to store the values $\epsilon = \frac{1}{\mu}$ and $\delta = \frac{1}{1 - \alpha^n}$ to avoid to recalculate them each time they are needed, but this general rule should be tested on different machines. In fact, particularly with machines characterized by small cache memory, sometimes the access-to-memory time exceeds the time needed to recalculate the above-mentioned factors.
- The evaluation of y_1 is quite expensive if performed as shown in eq. 1, involving $\frac{n(n-1)}{2} + n + 1$ flops. A less expensive way to compute y_1 employs an *Horner-like* algorithm, as described below:

$$y_1 = [b_1 + ((\dots (b_2\alpha + b_3)\alpha + \dots) + b_n)] \epsilon \delta$$

In this way, the evaluation of y_1 requires only $2n$ flops. System 19 is solved in a similar way:

1. $x_n = \frac{\alpha y_1 + \alpha^2 y_2 + \alpha^3 y_3 + \dots + \alpha^{n-1} y_{n-1} + y_n}{1 - \alpha^n}$
2. $x_i = y_i + \alpha x_{i+1} \quad i = n - 1, n - 2 \dots, 1$

Again, x_n is evaluated with a *Horner-like* algorithm.

At the end, the total number of floating-point operations is $9n - 6$

6 Some benchmarks

Until now we have adopted the Thomas's algorithm in our CFD codes, mainly used for Direct Numerical Simulation (DNS) of turbulent convective heat transfer in channels, and we didn't worry too much about its computational cost. Performing some *timing* on our codes it became clear that a significant amount of CPU time was spent in the solution of the cyclic-tridiagonal systems. Thus, our major objective when developing this solvers has been to save some execution time in respect of the Thomas algorithm, and the following statistics aim to show if this goal has been reached.

First, we can remember the number of operations (additions and multiplications) required from the different algorithms, compared with two algorithms for the solution of *simply tridiagonal* linear systems, LU-factorization algorithm and a cyclic-reduction algorithm. Of course the number of flops can be slightly different from code to code, but the *tendency* should be well represented by the following formulas:

Algorithm	Flops per right-hand side (only solution phase)
LU fact., A simply tridiagonal	$5n$
Cyclic-red., A simply tridiagonal	$7n$
Cyclic Thomas's algorithm	$17n - 16$
Temperton's algorithm, A symmetric	$7n$
Temperton, A with constant values	$6.5n$
Evans algorithm	$9n - 6$

In order to have a feeling of the time-savings obtained in a computer program we implemented the algorithms on a Pentium III 450 MHz PC using Visual Fortran V. 5.0.

The following execution-times are normalized by the execution-time needed for the solution of the problem with 2×10^5 unknowns employing the Thomas's algorithm.

For each algorithm we reported in parenthesis the ratio between the execution-time spent for the actual value of n and that needed for $n = 2 \times 10^5$.

n^r of eq.s (n)	Thomas	Temperton, A symm.	Evans
2×10^5	1.00	0.41 (1.0)	0.29 (1.0)
4×10^5	2.00	0.76 (1.85)	0.65 (2.2)
8×10^5	4.47	1.47 (3.58)	1.23 (4.24)

As could be expected, the execution-time doesn't have exactly a linear dependence from the number of equations to be solved, due probably to the different *weight* of addition and multiplication operations and to the unpredictable behaviour when one addresses variables which may, or may not, reside in the cache.

References

- [1] C. Temperton, Algorithms for the Solution of Cyclic Tridiagonal Systems, *J. Comp. Physics*, **19**, 317 – 323, (1975).

- [2] S.V. Patankar, C.H. Liu and E.M. Sparrow, Fully Developed Flow and Heat Transfer in Ducts Having Streamwise-Periodic Variation of Cross-Sectional Area, *ASME J. Heat Transfer*, **99**, 180 – 186, (1977).
- [3] S. Sebben and B.R. Baliga, Some Extensions of Tridiagonal and Pentadiagonal Matrix Algorithms, *Numer. Heat Transfer, Part B: Fundamentals*, 2B, 323 – 351, (1995).