J Math Chem (2017) 55:1793–1807
DOI 10.1007/s10910-017-0761-9

ORIGINAL PAPER

A specialised cyclic reduction algorithm for linear algebraic equation systems with quasi-tridiagonal matrices

Lesław K. Bieniasz[1]

Received: 4 April 2017 / Accepted: 6 June 2017 / Published online: 17 June 2017

Abstract Extensions have been developed, of several variants of the stride of two cyclic reduction method. The extensions refer to quasi-tridiagonal linear equation systems involving two additional nonzero elements in the first and last rows of the equation matrix, adjacent to the main three diagonals. Equations of this kind arise, for example, in the simulations of biosensors or other electrochemical systems by solving relevant ordinary or partial differential equations by finite difference methods, when boundary derivatives are approximated by one-sided, multipoint finite differences. The correctness of the algorithms developed has been verified using example matrices with pseudo-random coefficients, under conditions of both sequential and parallel execution.

Keywords Quasi-tridiagonal · Parallel algorithms · Cyclic reduction · Multipoint finite differences · Biosensors

Mathematics Subject Classification 65F05 · 65N22

1 Introduction

In this paper we describe a specialised cyclic reduction (CR) algorithm for numerically solving linear algebraic equation systems:

$$\mathbf{A}\mathbf{x} = \mathbf{r}, \tag{1}$$

✉ Lesław K. Bieniasz
nbbienia@cyf-kr.edu.pl

[1] Faculty of Physics, Mathematics and Computer Science, Cracow University of Technology, ul. Warszawska 24, 31-155 Cracow, Poland

in which $\mathbf{x} = [x_1, \ldots, x_n]$ is an n-dimensional vector of unknowns, $\mathbf{r} = [r_1, \ldots, r_n]$ is an n-dimensional known vector of real numbers, and $\mathbf{A}$ is a real $n \times n$ known square matrix having the following quasi-tridiagonal structure:

$$\mathbf{A} = \begin{bmatrix} b_1 & c_1 & d_1 & e_1 & & \ldots & 0 \\ a_2 & b_2 & c_2 & & & & \vdots \\ & a_3 & b_3 & c_3 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & a_{n-2} & b_{n-2} & c_{n-2} & \\ \vdots & & & & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & \cdots & & f_n & g_n & a_n & b_n \end{bmatrix}. \tag{2}$$

Matrix $\mathbf{A}$ differs from a purely tridiagonal matrix by additional, possibly non-zero, elements d_1, e_1, f_n and g_n, present in the first and last rows of the matrix. A typical situation requiring the solution of linear systems with matrix (2) arises from finite difference discretisations of two-point boundary value problems for second order ordinary differential equations (ODEs), or from analogous discretisations of initial-boundary value problems for (typically parabolic) one-dimensional partial differential equations (PDEs) (for an overview of finite difference ODE/PDE solving, see, for example, Jain [1], Smith [2], and Ascher et al. [3]). In particular, the present work is motivated by finite-difference simulations occurring in electroanalytical chemistry [4,5]. In such applications, coefficients a_i, b_i and c_i for $i = 2, \ldots, n-1$ result from the replacement of spatial (second and first) solution derivatives in the ODEs or PDEs by standard or compact three-point central approximations. Coefficients in the first and last rows of $\mathbf{A}$ result, in turn, from one-sided three- or four-point, standard or compact approximations to the first spatial derivatives occurring in boundary conditions. The latter discretisations are perhaps not very popular, since many authors use just two-point (one-sided or central) approximations to the boundary derivatives, which lead to purely tridiagonal matrices. However, there is evidence that the use of the multipoint one-sided finite difference approximations improves the accuracy of the solutions (see, in particular, the literature pertinent to electrochemical digital simulations: [4,6–10]). Numerical algorithms for solving Eq. (1) should therefore be of interest.

Three-point one-sided approximations to boundary derivatives usually have a theoretical accuracy order consistent with that of the three-point central derivatives in the ODEs or PDEs. In such a case coefficients e_1 and f_n are just zeroes. However, the use of more points for approximating derivatives at the boundaries may give still better results [4,6–10]. Furthermore, we shall see that it is relatively easy to incorporate the non-zero e_1 and f_n into the algorithm described below, whereas the consideration of more non-zero coefficients in the first and last rows of $\mathbf{A}$, would be more complicated. For these reasons we admit a possibility of $e_1 \neq 0$ and $f_n \neq 0$, in addition to $d_1 \neq 0$ and $g_n \neq 0$.

In former works [4,6–10] Eq. (1) was solved on serial computers by modifications of the classical Thomas algorithm [11], that is by such or other version of the serial LU factorization approach. For an overview of the literature related to the sequential

algorithms of solving equations similar to Eq. (1), the Reader is referred to Bieniasz [12,13]. In contrast to those serial algorithms, the numerical algorithm to be described here is an adaptation of the CR method for tridiagonal matrices, first described by Hockney [14] and Buzbee et al. [15] for block-tridiagonal matrices, and attributed to Golub and Hockney. This choice is dictated by the growing importance of parallel and vector computers for scientific and technical computing. Fine-grained parallelism is inherent in CR, which has prompted many authors to implement this method on a number of parallel and vector computers, or computer architectures (reviews of the method are available in [16–19]; a few example implementations of CR for scalar tridiagonal and block-tridiagonal systems are described in Refs. [20–38]). We shall focus on the most frequently considered, "stride of 2" version of CR (using the terminology of Evans [39]).

Apart from being used for scalar- and block-tridiagonal matrices, CR has been extended to periodic (cyclic) tridiagonal matrices [14,40–43], which present a different class of quasi-tridiagonal matrices, compared to matrix (2).

Of course, matrix **A** given by Eq. (2) can be viewed as a special case of banded matrices, for which a variety of parallel algorithms is available (see, for example, Refs. [44,45]), possibly utilising the concept of CR in some way. In particular, extensions of CR to pentadiagonal and general banded matrices are known [46,47]. However, since many of the elements on the external diagonals in **A** are zeroes, the algorithms for general banded matrices are likely to be unnecessarily complex, and therefore computationally more expensive than the specially dedicated algorithm described here (the dependence of the arithmetic complexity of banded solvers on the matrix bandwidth is documented [48]). A similar unwanted overhead can be expected from algorithms that might be developed for other generalisations of matrix **A** (for example for bordered matrices).

The CR algorithms and computer codes will be developed assuming that (as is often in practice) we need to solve a sequence of Eqs. (1) sharing the matrix **A**, but differing by vectors **r**. We shall also assume that n can be an arbitrary positive integer, in contrast to the frequently adopted simplification that n or $n-1$ is a positive integer power of 2. Of course, for small $n = 1,\ 2,\ 3$ some of the extra coefficients d_1, e_1, f_n and g_n must be zeroes, and/or matrix **A** is no longer (quasi-)tridiagonal.

2 The CR algorithms

We shall begin the algorithms presentation with a brief reminder (in Sect. 2.1) of the "stride of 2" CR (hereafter called just CR, for brevity) for a purely tridiagonal matrix. Then, in Sect. 2.2 we shall provide details of the method extensions to the quasi-tridiagonal matrix **A**.

2.1 The tridiagonal system

The basic idea of CR is to recursively reduce a tridiagonal system to a smaller system possessing an analogous matrix structure, by eliminating every second equation and every second variable. A complete set of operations for performing such a reduction,

for a particular system of size n, will be called a "reduction step". The above idea is the easiest to explain in the case when n is sufficiently large, and equation index i sufficiently bigger than 1 and smaller than n, so that the following subset of three adjacent equations is contained in the original tridiagonal system:

$$\left.\begin{array}{c} a_{i-1}x_{i-2} + b_{i-1}x_{i-1} + c_{i-1}x_i = r_{i-1} \\ a_i x_{i-1} + b_i x_i + c_i x_{i+1} = r_i \\ a_{i+1}x_i + b_{i+1}x_{i+1} + c_{i+1}x_{i+2} = r_{i+1} \end{array}\right\}. \tag{3}$$

In order to eliminate the $i-1$st and $i+1$st equations, and retain the ith equation, the following operations are performed. The $i-1$st equation is multiplied by $p_i = a_i/b_{i-1}$, the $i+1$st equation is multiplied by $q_i = c_i/b_{i+1}$, and the resulting equations are subtracted from the ith equation, which then becomes:

$$a'_i x_{i-2} + b'_i x_i + c'_i x_{i+2} = r'_{i-1}. \tag{4}$$

In Eq. (4) primes mark coefficients of the new, reduced system, so that $a'_i = -p_i a_{i-1}$, $b'_i = b_i - p_i c_{i-1} - q_i a_{i+1}$, and $c'_i = -q_i c_{i+1}$ are new tridiagonal matrix coefficients, and $r'_i = r_i - p_i r_{i-1} - q_i r_{i+1}$ is the element of the new right-hand side vector. Note that not only the $i-1$st and $i+1$st equations are eliminated, but also the unknowns x_{i-1} and x_{i+1}.

The above basic idea needs to be completed with additional refinements (a)–(f) listed below.

(a) First of all one has to decide which of the equations are to be eliminated, and which retained in a given reduction step. This gives rise to (at least) two variants of the reduction step. In the "odd–even" variant odd equations are eliminated, and even equations are retained. Conversely, in the "even–odd" variant, even equations are eliminated and odd equations are retained. An issue usually overlooked is that the identification of the odd/even equations may depend on the direction of counting them. Although a forward counting (from 1 to n) is common, a backward counting (from n to 1) may give a different selection of the equations, since n can be either odd or even. A systematic application of the forward or backward counting in successive reduction steps may also result in a somewhat different evolution of the related and subsequent calculations, and in slightly different machine error accumulation. Any sequential numbering of the equations does not, however, imply that the equations must be transformed sequentially in some order. In fact, operations performed on every second equation are entirely independent and can be performed in parallel.

(b) Initial and final equations in a sufficiently large system have to be handled somewhat differently from Eq. (3), in a given reduction step. Their treatment is also different in the odd–even and even–odd reduction step variants. Consider, in particular, the first two equations (the handling of the last two equations is symmetrical). If the first equation is to be eliminated, we subtract it from the second equation, after multiplying it by p_2, just as we do it in the case of the $i-1$st equation in the subsystem (3). The only difference is that we formally take $a_1 = 0$. If the first equation is to be retained, we subtract from it the second equation multiplied by q_1. Coefficient p_1 is not needed in this case.

(c) Modifications of the procedure described for Eq. (3) are needed also in the case of small systems ($n = 1, 2, 3, 4$). We omit these details here, but the Reader will find all the relevant formulae in Tables 1 and 2 (see Sect. 2.2).

(d) Depending on how the reduction steps are applied to the equation systems, we can also distinguish "ordinary CR" and "parallel CR" (the name "parallel CR" is predominantly used, see Hockney and Jesshope [17], but an alternative name, "cyclic elimination", is also encountered; see, for example, Gopalan and Murthy [30]).

In ordinary CR, after obtaining a first reduced system (having about $n/2$ unknowns), an analogous reduction step variant is applied to it, giving a second reduced system (having about $n/4$ unknowns), to which an analogous reduction step variant is again applied, etc. The reduction steps are continued until one finally obtains a single equation with a single unknown. A sequence of the so-called "back-substitution" steps is then recursively performed. In the first back-substitution step the last reduced system (with the single unknown) is solved, and the result is substituted into the equations that were eliminated while constructing the last reduced system. In every next back-substitution step a next portion of the unknowns is determined (this refers to the unknowns that were formerly eliminated from a given reduced system), based on the already determined unknowns, and substituted into the equations that were eliminated while constructing the given reduced system. These back-substitution steps are continued until all unknowns are determined. In particular, if the unknowns x_{i-2}, x_i and x_{i+2} (and the remaining unknowns of the first reduced system containing Eq. (4) are already determined, then the back-substitution gives from Eq. (3):

$$x_{i-1} = (r_{i-1} - a_{i-1}x_{i-2} - c_{i-1}x_i)/b_{i-1}, \tag{5}$$

$$x_{i+1} = (r_{i+1} - a_{i+1}x_i - c_{i+1}x_{i+2})/b_{i+1}, \tag{6}$$

and so on, for other unknowns that were eliminated from the first reduced system. As the determinations of the individual unknowns in a given back-substitution step are all independent of each other, they can be performed in parallel.

Parallel CR consists in applying simultaneously the odd–even and even–odd reduction steps. In this way, after the first reduction step one obtains two reduced systems (instead of only one in ordinary CR), having altogether n unknowns. After the second step one obtains four systems, still having jointly n unknowns. After a sufficient number of steps one obtains n independent equations with one unknown each. The equations are then solved, which ends the calculations. Thus, parallel CR amounts to the diagonalization of the matrix $\mathbf{A}$. The back-substitution steps are not needed in parallel CR. Note that the word "parallel" in the name of this method does not necessarily mean a parallel execution; all calculations can be done either sequentially or in parallel.

(e) From the programming point of view, single instances of data structures containing matrix $\mathbf{A}$ and vectors $\mathbf{r}$ and $\mathbf{x}$ are sufficient in ordinary CR, since all reduction and back-substitution steps can be realised by gradually transforming the initial $\mathbf{A}$ and $\mathbf{r}$. It is convenient to introduce the stride s and half-stride $h = s/2$. The first reduction step operates on (physical locations of) matrix rows and vector elements separated by $s = 2$, and s and h are doubled with every next reduction step. Later, they are halved with every back-substitution step. Hence, if Eqs. (3–6) are to be interpreted

Table 1 Formulae pertinent to the reduction steps of matrix **A**

Case	Subcase	Coefficients p_i, q_i	Matrix transformations
$i = 1 = n$	–	–	$\mathbf{A}' = \mathbf{A}$
$i = 1 < n$	$n - s < i \leq n - h$	p_i unused	$a'_i = 0$
		$q_i = c_i / b_{i+h}$	$b'_i = b_i - q_i a_{i+h}$
			$c'_i = 0$
			$d'_1 = 0$
			$e'_1 = 0$
	$n - s - h < i \leq n - s$	p_i unused	$a'_i = 0$
		$q_i = c_i / b_{i+h}$	$b'_i = b_i - q_i a_{i+h}$
			$c'_i = d_1 - q_i c_{i+h}$
			$d'_1 = 0$
			$e'_1 = 0$
	$i = n - s - h$	$p_i = e_1 / b_n$	$a'_i = 0$
		$q_i = \frac{c_i - p_i g_n}{b_{i+h}}$	$b'_i = b_i - q_i a_{i+h} - p_i f_n$
			$c'_i = d_1 - q_i c_{i+h} - p_i a_n$
			$d'_1 = 0$
			$e'_1 = 0$
	$i < n - s - h$	$q_i = c_i / b_{i+h}$	$a'_i = 0$
		$p_i = e_1 / b_{i+s+h}$	$b'_i = b_i - q_i a_{i+h}$
			$c'_i = d_1 - q_i c_{i+h} - p_i a_{i+s+h}$
			$d'_1 = -p_i c_{i+s+h}$
			$e'_1 = 0$
$i = n > 1$	$1 + h \leq i < 1 + s$	$p_i = a_i / b_{i-h}$	$a'_i = 0$
		q_i unused	$b'_i = b_i - p_i c_{i-h}$
			$c'_i = 0$
			$f'_n = 0$
			$g'_n = 0$
	$1 + s \leq i < 1 + s + h$	$p_i = a_i / b_{i-h}$	$a'_i = g_n - p_i a_{i-h}$
		q_i unused	$b'_i = b_i - p_i c_{i-h}$
			$c'_i = 0$
			$f'_n = 0$
			$g'_n = 0$
	$i = 1 + s + h$	$q_i = f_n / b_1$	$a'_i = g_n - p_i a_{i-h} - q_i c_1$
		$p_i = \frac{a_i - q_i d_1}{b_{i-h}}$	$b'_i = b_i - p_i c_{i-h} - q_i e_1$
			$c'_i = 0$
			$f'_n = 0$
			$g'_n = 0$
	$i > 1 + s + h$	$q_i = f_n / b_{i-s-h}$	$a'_i = g_n - p_i a_{i-h} - q_i c_{i-s-h}$
		$p_i = a_i / b_{i-h}$	$b'_i = b_i - p_i c_{i-h}$
			$c'_i = 0$

Table 1 continued

Case	Subcase	Coefficients p_i, q_i	Matrix transformations
			$f'_n = 0$
			$g'_n = -q_i a_{i-s-h}$
$1 < i < n$	$i = 1 + h = n - h$	$p_i = \frac{a_i b_n - c_i g_n}{b_1 b_n - d_1 g_n}$	$a'_i = 0$
		$q_i = \frac{b_1 c_i - d_1 a_i}{b_1 b_n - d_1 g_n}$	$b'_i = b_i - p_i c_1 - q_i a_n$
			$c'_i = 0$
	$1 + h = i < n - h$	$p_i = a_i / b_1$	$a'_i = 0$
		$q_i = \frac{c_i - p_i d_1}{b_{i+h}}$	$b'_i = b_i - p_i c_1 - q_i a_{i+h}$
			$c'_i = -p_i e_1 - q_i c_{i+h}$
	$i < 1 + h \wedge n - s < i \leq n - h$	$q_i = c_i / b_{i+h}$	$a'_i = 0$
		p_i unused	$b'_i = b_i - q_i a_{i+h}$
			$c'_i = 0$
	$i < 1 + h \wedge i \leq n - s$	$q_i = c_i / b_{i+h}$	$a'_i = 0$
		p_i unused	$b'_i = b_i - q_i a_{i+h}$
			$c'_i = -q_i c_{i+h}$
	$1 + h < i = n - h$	$q_i = c_i / b_{i+h}$	$a'_i = -p_i a_{i-h} - q_i f_n$
		$p_i = \frac{a_i - q_i g_n}{b_{i-h}}$	$b'_i = b_i - p_i c_{i-h} - q_i a_{i+h}$
			$c'_i = 0$
	$1 + h \leq i < 1 + s \wedge i > n - h$	$p_i = a_i / b_{i-h}$	$a'_i = 0$
		q_i unused	$b'_i = b_i - p_i c_{i-h}$
			$c'_i = 0$
	$i \geq 1 + s \wedge i > n - h$	$p_i = a_i / b_{i-h}$	$a'_i = -p_i a_{i-h}$
		q_i unused	$b'_i = b_i - p_i c_{i-h}$
			$c'_i = 0$
	$1 + h < i < n - h$	$p_i = a_i / b_{i-h}$	$a'_i = -p_i a_{i-h}$
		$q_i = c_i / b_{i+h}$	$b'_i = b_i - p_i c_{i-h} - q_i a_{i+h}$
			$c'_i = -q_i c_{i+h}$
	Other (parallel CR only)	–	$a'_i = a_i$
			$b'_i = b_i$
			$c'_i = c_i$

as formulae involving physical locations of the original matrix/vector elements, then indices $i-1, i+1, i-2$, and $i+2$, have to be replaced by $i-h, i+h, i-s$, and $i+s$, respectively. The primed quantities $\mathbf{A'}$ or $\mathbf{r}$' then refer to the same data structures that contain $\mathbf{A}$ or $\mathbf{r}$. In the case of parallel CR two instances of data structures containing $\mathbf{A}$ and $\mathbf{r}$ appear necessary. If $\mathbf{A}$ (or $\mathbf{r}$) is contained in one instance during a particular reduction step, then $\mathbf{A'}$ (or $\mathbf{r}$') can be placed into the second instance. In the next reduction step $\mathbf{A}'$ (or $\mathbf{r}$') plays the role of $\mathbf{A}$ (or $\mathbf{r}$), and $\mathbf{A}$ (or $\mathbf{r}$) plays the role of $\mathbf{A}'$ (or $\mathbf{r}$').

Table 2 Formulae pertinent to the reduction steps of vector **r**

Case	Subcase	Vector transformations
$i = 1 = n$	–	$\mathbf{r}' = \mathbf{r}$
$i = 1 < n$	$n - s < i \leq n - h$	$r'_i = r_i - q_i r_{i+h}$
	$n - s - h < i \leq n - s$	$r'_i = r_i - q_i r_{i+h}$
	$i \leq n - s - h$	$r'_i = r_i - q_i r_{i+h} - p_i r_{i+s+h}$
$i = n > 1$	$1 + h \leq i < 1 + s$	$r'_i = r_i - p_i r_{i-h}$
	$1 + s \leq i < 1 + s + h$	$r'_i = r_i - p_i r_{i-h}$
	$i \geq 1 + s + h$	$r'_i = r_i - p_i r_{i-h} - q_i r_{i-s-h}$
$1 < i < n$	$i < 1 + h \wedge i \leq n - h$	$r'_i = r_i - q_i r_{i+h}$
	$i \geq 1 + h \wedge i > n - h$	$r'_i = r_i - p_i r_{i-h}$
	$1 + h \leq i \leq n - h$	$r'_i = r_i - p_i r_{i-h} - q_i r_{i+h}$
	Other (parallel CR only)	$r'_i = r_i$

(f) In order to allow for multiple right-hand sides **r** (assumed in Sect. 1) it is desirable to implement all reduction steps for matrix **A** in a one separate procedure. All reduction steps for vector **r**, and back-substitution steps or determinations of **x** should then be contained in a second separate procedure. Coefficients p_i and q_i (for all reduction steps) are computed by the first procedure, and must be stored and supplied to the second procedure. The total number of these coefficients is relatively small in the case of ordinary CR, but can be quite large for parallel CR.

2.2 The quasi-tridiagonal system

The main complication arising from nonzero coefficients d_1, e_1, f_n and g_n is that they cause the first and last equations to involve four unknowns, instead of only two in the purely tridiagonal case. Therefore, if the first equation is to be retained in a given reduction step, it is not sufficient to subtract from it the second equation multiplied by a relevant coefficient q_1, as was done for the purely tridiagonal system (see point (b) in Sect. 2.1). It is also necessary to subtract the fourth equation multiplied by another coefficient (we shall use for this purpose the coefficient p_1, unused in the purely tridiagonal case), in order to eliminate one more unknown from the retained equation (n is assumed sufficiently large, the cases of small n require a separate treatment). The coefficients p_1 and q_1 are thus calculated and used differently in the quasi-tridiagonal system case. After one reduction step, coefficient e'_1 in the retained first equation will vanish. After two reduction steps coefficient d'_1 will also vanish, and the first equation will become purely tridiagonal. A symmetrical situation arises when retaining the nth equation: an additional subtraction of the $n - 3$rd equation multiplied by an extra coefficient q_n is necessary.

The above complication seems relatively easy to handle, but in fact it generates a number of new cases and sub-cases in the formulae for reduction and back-substitution steps, that were not needed for the purely tridiagonal system. Tables 1, 2 and 3 provide

Table 3 Formulae pertinent to the calculations of vector **x** in back-substitution steps (for ordinary CR only)

Case	Subcase	Vector transformations
$n-h<i<1+h$	–	$x_i = \frac{r_i}{b_i}$
$i=1<n$	$n-s<i\le n-h$	$x_i = \frac{r_i - c_i x_{i+h}}{b_i}$
	$i=n-s$	$x_i = \frac{b_n(r_i - c_i x_{i+h}) - d_1(r_n - a_n x_{i+h})}{b_i b_n - d_1 g_n}$
	$i<n-s$	$x_i = \frac{r_i - d_1 r_{i+s}/b_{i+s} - (c_i - d_1 a_{i+s}/b_{i+s})x_{i+h} - (e_1 - d_1 c_{i+s}/b_{i+s})x_{i+s+h}}{b_i}$
$i=n>1$	$1+h\le i<1+s$	$x_i = \frac{r_i - a_i x_{i-h}}{b_i}$
	$i=1+s$	$x_i = \frac{b_1(r_i - a_i x_{i-h}) - g_n(r_1 - c_1 x_{i-h})}{b_1 b_i - d_1 g_n}$
	$i>1+s$	$x_i = \frac{r_i - g_n r_{i-s}/b_{i-s} - (a_i - g_n c_{i-s}/b_{i-s})x_{i-h} - (f_n - g_n a_{i-s}/b_{i-s})x_{i-s-h}}{b_i}$
$1<i<n$	$i<1+h$	$x_i = \frac{r_i - c_i x_{i+h}}{b_i}$
	$i>n-h$	$x_i = \frac{r_i - a_i x_{i-h}}{b_i}$
	$1+h\le i\le n-h$	$x_i = \frac{r_i - a_i x_{i-h} - c_i x_{i+h}}{b_i}$

a summary of all necessary cases and subcases, and related formulae, pertinent to the reduction steps of matrix **A** (Table 1), reduction steps of vector **r** (Table 2), and back-substitution steps for ordinary CR (Table 3). The formulae from Tables 1, 2 and 3 apply to physical locations of the original elements of **A**, **r**, and **x**, with index i numbering all the matrix rows and vector elements (from 1 to n). Stride s, and half-stride h depend on the reduction or back-substitution step, as was already noted (see point (e) in Sect. 2.1). The various cases and subcases usually correspond to the various total numbers of equations contained in a particular reduced system, and various positions of the ith equation of the initial system (1), inside this reduced system.

3 Numerical experiments

In order to test the CR algorithms described in Sect. 2, a large number of example systems (1) was solved. In a majority of these examples elements a_i, b_i, c_i, d_1, e_1, f_n and g_n, were initially filled with nonzero pseudo-random real numbers having a uniform distribution over a certain interval $(u, v) \subset \mathbb{R}$. The sums of absolute values of the off-diagonal elements were subsequently added to, or subtracted from the diagonal elements b_i, depending on whether b_i was positive or negative, respectively. In this way diagonally dominant matrices **A** were obtained. The diagonal dominance is known to be important for the numerical stability of the CR method for purely tridiagonal scalar matrices [49]. In order to have exactly known solution vectors $\mathbf{x}_{\text{exact}}$, vectors $\mathbf{x}_{\text{exact}}$ of n elements were prepared by filling them with with pseudo-random numbers. Vectors **r** were then computed as

$$\mathbf{r} = \mathbf{A}\mathbf{x}_{\text{exact}}. \tag{7}$$

After solving Eq. (1) errors $\mathbf{e}_x$ of the solutions were calculated as $\mathbf{e}_x = \mathbf{x} - \mathbf{x}_{\text{exact}}$.

The above tests using pseudo-random matrices **A** allow one to more comprehensively verify the correctness of the CR algorithms than any tests involving matrices

resulting from finite-difference ODE or PDE discretizations. The latter matrices often show regular patterns of coefficients (many coefficients may even be identical), so that the effects of individual coefficients may not be noticed in the tests.

All previously mentioned mutations of CR have been tested: ordinary CR with odd–even and even–odd reduction, and parallel CR, assuming either forward or backward equation counting. Purely sequential as well as parallel performances of the algorithms were examined (see implementation details below). In addition to using CR, the example systems were also solved, for comparison, by several variants of the sequential LU decomposition methods (or the Thomas algorithm [11]). These were: the Doolittle and the Crout methods (see, for example, Kincaid and Cheney [50]) with matrix factorization proceeding either in a forward or backward sweep. The relevant algorithms, applicable to Eq. (1), were obtained by modifying the procedures described in Refs. [8,12,13]. Hybrid algorithms, combining incomplete CR with LU decompositions were also tried, but we do not elaborate on them here, since they were not found more efficient under conditions of the present implementation.

All computer programs were written in C++ using double precision (*double* C variables having 64 bits and 16 digit precision, compliant with the IEEE 754 standard [51]), and compiled as 32-bit console applications under Bloodshed/Orwell Dev-C++ 5.7.1 environment [52,53], using the TDM-GCC 4.8.1 compiler, a 32-bit release. Matrices $\mathbf{A}$ were implemented as class objects containing three vectors (of length n each) with coefficients a_i, b_i, c_i, and four real variables d_1, e_1, f_n and g_n. The parallelism of the programs was achieved by multithreading, using OpenMP directives [54] to decompose loops over index i in matrix/vector reduction steps and in back-substitution steps, into portions performed by separate threads. Multithreading is generally not expected to provide the most efficient parallel CR performance for scalar systems (1), because the cost of maintaining and communicating the threads is considerable in comparison with the costs of CR calculations themselves, unless n is very large. OpenMP was previously used by Hirshman et al. [37] and Lecas et al. [38] in their CR codes for block-tridiagonal systems, in which case the costs of CR calculations were relatively larger, compared to the parallel overheads. Most efficient CR performance can probably be achieved at present by using modern GPU programming [35,36], but this requires a suitable hardware, which is still not widely available. In contrast, OpenMP coding is relatively straightforward and applicable to the majority of currently available processors, and it is entirely sufficient for our main purpose of checking the correctness of the CR algorithms for Eq. (1) under conditions of parallel execution.

Calculations were run mostly on a multicore computer with an Intel Core i7-4960X CPU, operating at 3.6 GHz, under the Windows 7 x64 Ultimate operating system. The computer allowed for a maximum of 12 simultaneous threads being run on 6 independent cores.

4 Results and conclusions

There are two essential aspects of the numerical algorithms for solving Eq. (1), that should be of interest in practice: errors and computational times (including possible parallel speedups). These aspects are explored below.

In the case of small n one may expect that the relative errors $\|\mathbf{e}_x\|_\infty / \|\mathbf{x}\|_\infty$ of the solutions result mostly from the relative perturbations $\|\mathbf{e}_r\|_\infty / \|\mathbf{r}\|_\infty$ of vectors $\mathbf{r}$, according to the well known estimate of the maximum error [50]:

$$\frac{\|\mathbf{e}_x\|_\infty}{\|\mathbf{x}\|_\infty} \approx \mathrm{cond}(\mathbf{A}) \frac{\|\mathbf{e}_r\|_\infty}{\|\mathbf{r}\|_\infty}, \tag{8}$$

where $\mathrm{cond}(\mathbf{A})$ is the condition number of matrix $\mathbf{A}$. This is because matrix $\mathbf{A}$ was known precisely in our experiments, but vectors $\mathbf{r}$ differed slightly from their exact values, since they were calculated numerically from Eq. (7). Assuming that $\|\mathbf{e}_r\|_\infty / \|\mathbf{r}\|_\infty$ is roughly at the level of the machine precision ν ($\nu \approx 1.11 \times 10^{-16}$ in the case of double precision variables [51]), Eq. (8) predicts:

$$\log \frac{\|\mathbf{e}_x\|_\infty}{\|\mathbf{x}\|_\infty} \approx \log \nu + \log[\mathrm{cond}(\mathbf{A})]. \tag{9}$$

As the matrices used in our experiments were random, their condition numbers also took random values.

With increasing n, errors $\|\mathbf{e}_r\|_\infty / \|\mathbf{r}\|_\infty$ are expected to be greater than ν, due to machine errors generated in arithmetic operations involved in formula (7). Similarly, the machine errors arising in the CR or LU procedures (omitted in Eqs. (8) and (9)) bring an increasing contribution to the solution errors $\|\mathbf{e}_x\|_\infty / \|\mathbf{x}\|_\infty$, above the maximum level indicated by these equations.

Figure 1 demonstrates that solution errors measured in numerical experiments are consistent with the above expectations. Figure 1 shows typical error distributions for ordinary CR with odd–even reduction, and forward counting, and for LU decomposition by the Doolittle method with matrix factorization in a forward sweep. The distributions look very similar, and they were alike for all other methods tested. As can be seen, for $n \leq 2000$ the solution errors vary approximately in the interval $10^{-16} \leq \|\mathbf{e}_x\|_\infty / \|\mathbf{x}\|_\infty \leq 10^{-11}$, so that, on average, a thousendfold increase of n corresponds to the increase of the errors by a factor of about 10^5. The generally small magnitudes of the errors confirm the overall correctness of the algorithms developed in this study.

The effect of the interval (u, v) of pseudo-random coefficient values, on the errors, was investigated assuming $(u, v) = (-10^2, 10^2)$, $(-10^5, 10^5)$, $(-10^{10}, 10^{10})$, $(-10^{20}, 10^{20})$ and $(-10^{100}, 10^{100})$, but no significant changes of the error distributions were observed. For still larger intervals (u, v) well expected overflow errors occurred, precluding obtaning correct solutions.

Computational times of the matrix $\mathbf{A}$ reduction phase, or of the LU decomposition, varied between the minimum of about 10^{-6} s for $n = 2$, and the maximum of about 1 s for $n = 10^6$ in the case of the sequential execution, and between 10^{-4} and 1 s

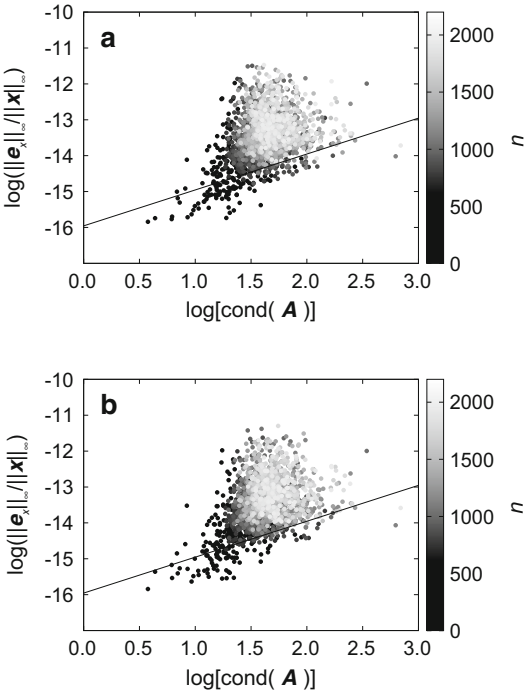

Fig. 1 Relative solution errors $\|\mathbf{e}_x\|_\infty / \|\mathbf{x}\|_\infty$ obtained in experiments with pseudo-random coefficients of Eq. (1), by using ordinary CR with odd–even reduction and forward counting, a sequential execution (**a**), and LU decomposition by the Doolittle method with forward matrix reduction (**b**). The *errors* are plotted as functions of the condition number cond(**A**) of matrix **A**, for various values of the system dimension n from the interval [2, 2000], indicated by various degrees of *shadow*. The pseudo-random coefficient values were chosen from the interval $(u, v) = (-10^2, 10^2)$. *Solid lines* represent plots of Eq. (9)

in the case of the parallel execution of the algorithms. This refers to all algorithms examined, although differences in timings, reaching even two orders of magnitude, were observed between various algorithms, also depending on the number of threads. Parallel CR was systematically much slower than ordinary CR.

Comparable ranges of computational times were observed in the vector **r** reduction and solution phases.

At low n a performance loss of the parallel calculations, relative to the sequential calculations, was noticeable, due to the parallel overheads associated with maintaining and communicating the threads (see Sect. 3). Acceleration of the calculations, resulting from the parallel execution, was observed only at very large n (greater than about 10^4). Parallel speedups (relative to the fastest sequential algorithm, which is LU decomposition) exceeding unity were recorded only in the case of ordinary odd–even or even–odd CR, at $n \gtrsim 10^5$. Speedups corresponding to this particular situation are plotted in Fig. 2, as functions of n. As can be seen, with six threads the speedups approach maximally the value of about 1.4. Using more threads resulted in a decrease of the speedup. This was probably caused by the way the threads were distributed among the six available cores, although other reasons, especially the issues of the cache memory access, and the role played by the operating system, might also be important; these questions were not studied further. It can also be seen that ordinary odd–even CR is about 2–2.5 times slower than LU decomposition, in the case of the sequential execution. This result is consistent with the arithmetic cost evaluations for purely tridiagonal systems [18].

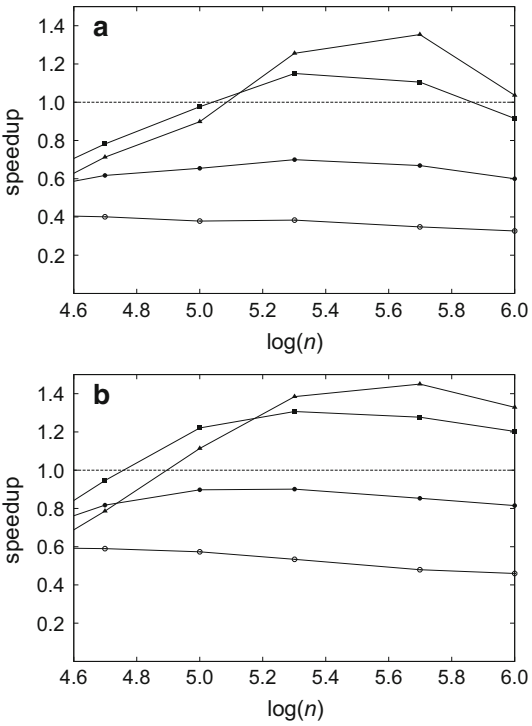

Fig. 2 Parallel speedups obtained for ordinary odd–even CR, by using sequential execution (*white circles*), two threads (*black circles*), four threads (*black squares*), and six threads (*black triangles*). *Dotted lines* indicate the speedup = 1. Subfigure **a** refers to the speedups of the matrix **A** reduction phase, relative to the timings of the matrix LU decomposition phase by the Doolittle method with matrix factorization in a forward sweep. Subfigure **b** refers to the speedups of the vector **r** reduction and back-substitution phases, relative to the timings of the solution phase in the LU decomposition method. All computational times were obtained as averages from 2000 program runs

The speedups obtained confirm that the particular software and hardware configuration used in the tests (multithreading using OpenMP on the multicore Intel Core i7-4960X CPU, operating at 3.6 GHz, under the Windows 7 x64 Ultimate) cannot be recommended for the most efficient implementations of cyclic reduction. Alternative configurations (in particular those using GPUs) are likely to be more attractive.

Acknowledgements The computer code developed for the present work is available from the author, upon request.

References

1. M.K. Jain, *Numerical Solution of Differential Equations* (Wiley, New Delhi, 1984)
2. G.D. Smith, *Numerical Solution of Partial Differential Equations, Finite Difference Methods* (Clarendon Press, Oxford, 1985)
3. U.M. Ascher, R.M.M. Mattheij, R.D. Russell, *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations* (SIAM, Philadelphia, 1995)
4. D. Britz, J. Strutwolf, *Digital Simulation in Electrochemistry* (Springer, Berlin, 2016)
5. R. Baronas, F. Ivanauskas, J. Kulys, *Mathematical Modeling of Biosensors* (Springer, Dordrecht, 2010)
6. D. Britz, Anal. Chim. Acta **193**, 277 (1987)

7. D. Britz, J. Electroanal. Chem. **352**, 17 (1993)
8. L.K. Bieniasz, Comput. Chem. **26**, 633 (2002)
9. L.K. Bieniasz, Comput. Biol. Chem. **27**, 315 (2003)
10. L.K. Bieniasz, Electrochim. Acta **52**, 2203 (2007)
11. L.H. Thomas, *Elliptic Problems in Linear Difference Equations Over a Network* (Tech. rep, New York, 1949)
12. L.K. Bieniasz, Computing **67**, 269 (2001)
13. L.K. Bieniasz, Computing **70**, 275 (2003)
14. R.W. Hockney, J. ACM **12**, 95 (1965)
15. B.L. Buzbee, G.H. Golub, C.W. Nielson, SIAM J. Numer. Anal. **7**, 627 (1970)
16. H.S. Stone, A.C.M. Trans, Math. Softw. **1**, 289 (1975)
17. R.W. Hockney, C.R. Jesshope, *Parallel Computers 2* (Institute of Physics Publishing, Bristol, 1988)
18. W. Gander, G.H. Golub, in *Proceedings of the Workshop on Scientific Computing*, ed. by G.H. Golub, S.H. Liu, F.T. Luk, R.J. Plemmons (Springer, New York, 1997)
19. D.A. Bini, B. Meini, Numer. Algorithms **51**, 23 (2009)
20. J.J. Lambiotte Jr., R.G. Voigt, A.C.M. Trans, Math. Softw. **1**, 308 (1975)
21. R.N. Kapur, J.C. Browne, SIAM J. Sci. Stat. Comput. **5**, 701 (1984)
22. S.L. Johnsson, Comput. Phys. Commun. **37**, 195 (1985)
23. M.W. Berry, A. Sameh, Int. J. Supercomput. Appl. **2**, 37 (1988)
24. R. Reuter, Parallel Comput. **8**, 371 (1988)
25. K. Dowers, S. Lakshmivarahan, S.K. Dhall, J. Diaz, in *Parallel Processing for Scientific Computing*, ed. by G. Rodrigue (SIAM, Philadelphia, 1989), pp. 56–60
26. F. Reale, Parallel Comput. **16**, 361 (1990)
27. C.L. Cox, J.A. Knisely, J. Parallel Distrib. Comput. **13**, 325 (1991)
28. J.M. Badía, A.M. Vidal, in *Proceedings of Euromicro Workshop on Parallel and Distributed Processing* (IEEE, 1992), pp. 203–210
29. P. Amodio, N. Mastronardi, Parallel Comput. **19**, 1273 (1993)
30. K. Gopalan, C.S.R. Murthy, Int. J. High Speed Comput. **9**, 311 (1997)
31. K. Sumiyoshi, T. Ebisuzaki, Parallel Comput. **24**, 287 (1998)
32. E.E. Santos, in *Proceedings of the International Conference on Parallel and Distributed Processing and Techniques (PDPTA)* (1998), pp. 1788–1791
33. E.E. Santos, J. Comput. Inf. Technol. CIT **8**, 1 (2000)
34. S. Allmann, T. Rauber, G. Rünger, in *Proceedings of the 9th Euromicro Workshop on Parallel and Distributed Processing* 2001, pp. 290–297
35. J. Lamas-Rodríguez, M. Bóo, D.B. Heras, F. Argüello, in *XX Jornadas de Paralelismo, La Coruña, España* (2009), pp. 349–354
36. P. Alfaro, P. Igounet, P. Ezzatti, Mech. Comput. **29**, 2951 (2010)
37. S.P. Hirshman, K.S. Perumalla, V.E. Lynch, R. Sanchez, J. Comput. Phys. **229**, 6392 (2010)
38. D. Lecas, R. Chevalier, P. Joly, Multicore parallelization of block cyclic reduction algorithm (2015), http://www.prace-ri.eu/IMG/pdf/multicoreparallelizationblockcyclicreduction.pdf. Accessed 10 Mar 2017
39. D.J. Evans, Int. J. Comput. Math. **41**, 237 (1992)
40. C. Temperton, J. Comput. Phys. **19**, 317 (1975)
41. R. Reuter, ACM SIGAPL APL Quote Quad **18**, 6 (1988)
42. R.F. Boisvert, SIAM J. Sci. Stat. Comput. **12**, 423 (1991)
43. M.P. Bekakos, Bull. Greek Math. Soc. **33**, 107 (1992)
44. S.J. Wright, SIAM J. Sci. Stat. Comput. **12**, 824 (1991)
45. I.S. Duff, H.A. van der Vorst, Parallel Comput. **25**, 1931 (1999)
46. N.K. Madsen, G.H. Rodrigue, in *Parallel Computers-Parallel Mathematics*, ed. by M. Feilmeier (North Holland, Amsterdam, 1977), pp. 103–106
47. G.H. Rodrigue, N.K. Madsen, J.I. Karush, J. ACM **26**, 72 (1979)
48. M.M. Chawla, D.J. Evans, Int. J. Comput. Math. **80**, 473 (2003)
49. P.Y. Yalamov, BIT **34**, 428 (1994)
50. D. Kincaid, W. Cheney, *Numerical Analysis, Mathematics of Scientific Computing* (Brooks and Cole, Pacific Grove, 2002)
51. IEEE. IEEE 754: Standard for binary floating point arithmetic (2017), http://grouper.ieee.org/groups/754. Accessed 10 Mar 2017

52. Bloodshed Software. Dev-C++ (2017) , http://www.bloodshed.net. Accessed 10 Mar 2017
53. Orwell. Dev-C++ Blog (2017), http://orwelldevcpp.blogspot.com. Accessed 10 Mar 2017
54. OpenMP Architecture Review Board. OpenMP C and C++ application program interface version 2.0 march 2002 (2002), http://www.openmp.org/mp-documents/cspec20.pdf. Accessed 10 Mar 2017