Large-Scale Scientific Computations of Engineering and Environmental Problems

Proceedings of the First Workshop on „Large-Scale Scientific Computations“ Varna, Bulgaria, June 7–11, 1997

Edited by
Michael Griebel
Olep P. Iliev
Svetozar D. Margenov
Panayot S. Vassilevski

vieweg

A New Stable Solver and Block Elimination for Bordered Systems

Plamen Y. Yalamov
Center of Applied Mathematics and Informatics, University of Rousse
7017 Rousse, Bulgaria
e-mail: yalamov@ami.ru.acad.bg

Marcin Paprzycki
Department of Computer Science and Statistics
University of Southern Mississippi, Southern Station, Box 5167
Hattiesburg, Mississippi 39406, USA
e-mail: marcin@usm.edu

Summary

A block elimination method for bordered systems is studied. The case when the leading principal block is ill-conditioned, or singular, and the method is strongly unstable, is analysed. A perturbation approach is proposed to enhance the stability. The new algorithm is compared to existing approaches, and it is shown that it works faster while preserving the stability of the solution process.

Key words: bordered system, stable solver, block elimination, fast solver
AMS subject classifications: 65F05, 65G05, 65F30

1 Introduction

Let us suppose that we are given a linear system of equations in the following block form:

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix}, \quad \text{or} \quad Mz = b,$$

where $A \in \mathcal{R}^{n\times n}, B \in \mathcal{R}^{n\times m}, C \in \mathcal{R}^{m\times n}, D \in \mathcal{R}^{m\times m}, x, f \in \mathcal{R}^{n\times 1}, y, g \in \mathcal{R}^{m\times 1}$ are matrices or vector blocks. Usually, m is very small in comparison to n. Matrix A may be ill-conditioned, or singular. Such problems arise in various fields, e. g. parallel solution of banded systems [10], numerical continuation and bifurcation [3, 5, 12, 14], symmetry-breaking bifurcations [13], constrained optimization [6], and domain decomposition methods [2] and others.

Typically, a standard black-box solver is applied to the matrix A. When A has some special structure (e. g. banded, block diagonal, sparse) a special solver is applied which accelerates the solution process significantly. But, as we mentioned above, this matrix can be ill-conditioned, or singular. Thus the solver for this matrix can produce incorrect results. We can apply a stable solver for the whole matrix M but then we will loose the structure, and the computational cost (time) will increase considerably because we do not utilize the properties of matrix A.

In several papers [4, 8, 9] algorithms have been proposed that utilize the information about the structure of A and allow a stable computation of the solution z. A method for the stable computation of z proposed in [4] requires the computation of several singular vectors of A, and at least $3m + 1$ solves with A (in the case where A has maximum rank defficiency) are needed. In [9] a new algorithm presented by Govaerts and Pryce needs 2 solves involving A and 1 solve involving A^T at each bordering step (the number of bordering steps is equal to m). The latter algorithm is faster and produces quite accurate results.

In the present paper we will develop a new algorithm which produces better timing results than those in [9] but preserves similar stability properties. The algorithm is based on applying perturbations to stabilize the algorithm (similarly to, for instance, the methods introduced in [1, 11]). The general idea is to add relatively small perturbations whenever in the algorithm there is a division by a small number, which can produce a blowup of the rounding error. Because of these perturbations we solve not the original problem but a perturbed one. After the algorithm is finished we apply the usual iterative refinement [7, §3.5.3] to recover the solution of the original system. This step is cheap because we use a black-box factorization of A (e. g. LU factorization). In practice one step of iterative refinement is enough. So, this results in 1 block solve and 2 scalar solves involving A in most of the cases. This means that we do not need a solver for A^T, and that the most time consuming part of the algorithm is the LU decomposition of A. Even if we need more steps of iterative refinement for some matrices this will not lead to essential increase in the computational time. The speedup of the new algorithm comes also from the fact that we use the BLAS (Basic Linear Algebra Subrourtines) implementation of block operations whenever possible. These are more efficient on high-performance architectures. The main difference between our approach and the Govaerts-Pryce algorithm [9] is that our approach is not so sensible to block operations while the algorithm in [9] is quite unstable if block operations are applied.

The outline of the paper is as follows. In Section 2 we summarize the Govaerts-Pryce algorithm. Then in Section 3 we present the new algorithm and some theoretical analysis. Section 4 presents the numerical experiments.

2 The Govaerts-Pryce algorithm

This algorithm has been originally developed for bordering with width 1 (i. e. $m = 1$ in our notations). For systems with wider borders the algorithm is applied recursively. The algorithm is summarized as follows:

```
Step 1. Solve $A^T V^* = C^T$.
Step 2. Compute $\delta^* = D - V^{*T} B$.
Step 3. Solve $AV = B$.
Step 4. Compute $\delta = D - CV$.
Step 5. Compute $y_1 = (g - V^{*T} f)/\delta^*$.
Step 6. Compute $f_1 = f - B y_1$.
Step 7. Compute $g_1 = g - d y_1$.
Step 8. Solve $Aw = f_1$.
Step 9. Compute $y_2 = (g_1 - Cw)/\delta$.
Step 10. Compute $x = w - V y_2$.
Step 11. Compute $y = y_1 + y_2$.
```

The stability of the algorithm and a number of examples are given in [9]. The algotihm is quite stable even for almost singular matrices A. But the stability is guaranteed only for borders with width 1 (i. e. recursive application for wider borders), and it is not difficult to find numerical examples in which any block bordering leads to strong instability.

3 The perturbation approach

Our approach is based on the following block LU-decomposition of the matrix M:

$$M = \begin{pmatrix} PLU & 0 \\ C & I_m \end{pmatrix} \begin{pmatrix} I_n & V \\ 0 & \Delta \end{pmatrix}, \tag{1}$$

where $V = A^{-1}B, \Delta = D - CV$. Here we calculate $A = PLU$ by LU-decomposition with partial pivoting (this is the most time consuming part of the algorithm). Then if A is ill-conditioned (or singular) the matrix U has small (or zero) elements along its principal diagonal. The division by these will lead to a blowup of the error in the final solution.

To overcome this difficulty we perturb the diagonal entries u_{ii} of U by a number δ which will be specified later on. We have

$$\tilde{u}_{ii} = u_{ii} + \delta_{u_{ii}} = u_{ii} + \mathrm{Sgn}(u_{ii})\delta,$$

where

$$\mathrm{Sgn}(a) = \begin{cases} \mathrm{sign}(a), & \text{if } a \neq 0, \\ 1, & \text{if } a = 0. \end{cases}$$

In this way the growth of intermediate results is bounded to some extent, and we have some correct digits in the solution. After the perturbation is done the solution is computed by an obvious application of the block decomposition (1).

Summarizing, the algorithm looks as follows:

```
Step 1. Compute $PLU = A$.
Step 2. If $|u_{ii}| < \delta$, then perturb $u_{ii} = u_{ii} + \mathrm{Sgn}(u_{ii})\delta$.
Step 3. Solve $AV = B$ by the the LU-decomposition.
Step 4. Compute $\Delta = D - CV$.
Step 5. Solve $Ax_1 = f$ by the LU-decomposition.
Step 6. Compute $y_1 = g - Cx_1$.
Step 7. Solve $\Delta y_\delta = y_1$.
Step 8. Compute $x_\delta = x_1 - Vy_\delta$.
```

The result of the algorithm is the perturbed solution $z_\delta^T = (x_\delta^T \; y_\delta)^T$.

Since the solution is perturbed we are probably away from the exact solution. Therefore, we apply the standard iterative refinement [7, §3.5.3] in order to recover the accurate solution. Usually one step of iterative refinement is enough to obtain a solution with accuracy close to the machine precision. This procedure is quite well-known, and we omit it here.

Our purpose is also to estimate the influence of the perturbation. At first glance the influence is not clear because we perturb some of the intermediate results. But from the equation

$$u_{ii} = a_{ii} - \sum_{j=1}^{i-1} l_{ij} u_{ji},$$

which defines the element u_{ii}, we have that

$$\tilde{u}_{ii} = u_{ii} + \delta_{u_{ii}} = a_{ii} + \delta_{a_{ii}} - \sum_{j=1}^{i-1} l_{ij} u_{ji}.$$

So, the perturbation in u_{ii} is equivalent to a perturbation in a_{ii}, and both perturbations are equal. Let us note that a_{ii} is a diagonal entry in the permuted matrix A but we stay with the notation a_{ii} for simplicity of notation. This does not influence the final conclusions.

Summarizing, the perturbation of the elements u_{ii} is equivalent to solution of the original problem but with a perturbed matrix $\tilde{M} = M + \delta M$, where $|\delta M| \leq \delta I$. The matrix δM is diagonal, where some of diagonal entries are nonzero and equal to $\pm\delta$. Now it is not difficult to estimate the influence of the perturbation on the error of the solution but we shall do this together with some roundoff error analysis in the next section. From this analysis we shall derive a rule for choosing the value of δ.

4 Stability issues

In this section we use the result for the roundoff analysis obtained in [16]. Backward error analysis (see [15]) for the scalar and the block versions of the bordering method is presented there in the following form

$$(M + \varepsilon_M)\tilde{z} = b,$$

where ε_M is the equivalent perturbation of matrix M, and $\tilde{z}$ is the computed solution. A bound of the following type is derived for ε_M in our notations:

$$\|\varepsilon_M\|_\infty \leq c_{n+m} K \|M\|_\infty \rho_0, \tag{2}$$

where c_{n+m} is a constant linearly depending on the matrix size $n+m$, K is a bound for the growth factor (growth of intermediate results), and ρ_0 is the machine roundoff unit. We shall use this bound here for the perturbed matrix $M + \delta M$:

$$(M + \delta M + \varepsilon_{\tilde{M}})\tilde{z}_\delta = b, \tag{3}$$

where we assume that the bound 2 is also valid. The subscript δ denotes that the solution is computed with perturbations (possibly).

The term $c_{n+m}K$ is difficult to bound in our algorithm. Since we do not allow divisions by numbers less than δ we model this term by

$$c_{n+m}K \approx 1/\delta^s,$$

where s is again difficult to estimate. Then from (3) simple perturbation analysis (after some manipulations) produces the bound

$$\frac{\|\tilde{z}_\delta - z\|_\infty}{\|z\|_\infty} \leq \frac{\kappa(M)(\delta + \rho_0/\delta^s)}{1 - \kappa(M)(\delta + \rho_0/\delta^s)}, \tag{4}$$

$$\kappa(M) = \|M^{-1}\|_\infty \|M\|_\infty,$$

provided that $\|M\|_\infty \geq 1$ (which can be assumed true by some type of scaling).

We would like to have as small error in the solution as possible. Therefore, the value of δ is chosen so as to minimize the expression in the brackets on the right hand side of (4):

$$\delta = (s\rho_0)^{1/(s+1)}.$$

From our extensive practical experience (including a large set of random matrices) we observed that the value of $1 \leq s \leq 2$ is most probable (in most of the cases $s = 1$). Therefore, we recommend the value of $\delta \approx \sqrt{\rho_0}$. In all the numerical experiments this value is chosen equal to 10^{-4} in single precision ($\rho_0 \approx 10^{-7}$).

5 Numerical tests

We present the results from tests with random matrices in single precision ($\rho_0 \approx 10^{-7}$). The exact solution in all the examples is chosen to be $z = (1, \dots, 1)^T$. The value of δ in all the examples is 10^{-4}.

The matrices A are generated randomly as follows:

$$A = H_1 \dots H_{100}\mathrm{diag}(0, 0, 0, 0.7 + 0.04n, 0.7 + 0.04(n-1), \dots, 0.86)H_{101} \dots H_{200},$$

where

$$H_i = I - 2h_i h_i^T,$$

and h_i is a vector of unit length which entries are random and uniformly distributed in [0,1]. Similar example matrices are used in [9].

Figures 1,2 present experimental results from SGI POWER CHALLENGE 8000 (SGI PC 8000), and Figures 3,4 from SGI POWER CHALLENGE 10000 (SGI PC 10000) supercomputers. In each figure the first picture shows the time, the second one shows the speedup (i. e. time of the Govaerts-Pryce algorithm over time of the perturbation approach), and the third one presents the forward error $\|\tilde{z} - z\|_\infty / \|z\|_\infty$ in the computed solution $\tilde{z}$. For more clarity in the pictures representing the error we include also information about the machine precision $\rho_0 \approx 10^{-7}$. We have made experiments with other values of n achieving similar results.

All the results lead to the following observations:

1. The computational time for the Govaerts-Pryce algorithm grows linearly with m, while the computational time of our approach is almost constant when m varies. This is due to the increased speed of blocked BLAS-based operations.

2. The perturbation approach works several times faster, and the speedup grows linearly with m. Let us note that in all the examples the perturbation approach needs only one step of iterative refinement.

3. The computational times on both SGI PC 10000 and SGI PC 8000 are almost the same (the former is slightly faster but the difference is so small that it can not be seen from the graphs). This result is slightly surprising as the speed of the floating point operations on the 10000 machine should be about 25% faster than the 8000 machine.

4. The error in both algorithms is similar. In most of the cases the perturbation approach produces a slightly better error.

5. The error does not change essentially with m.

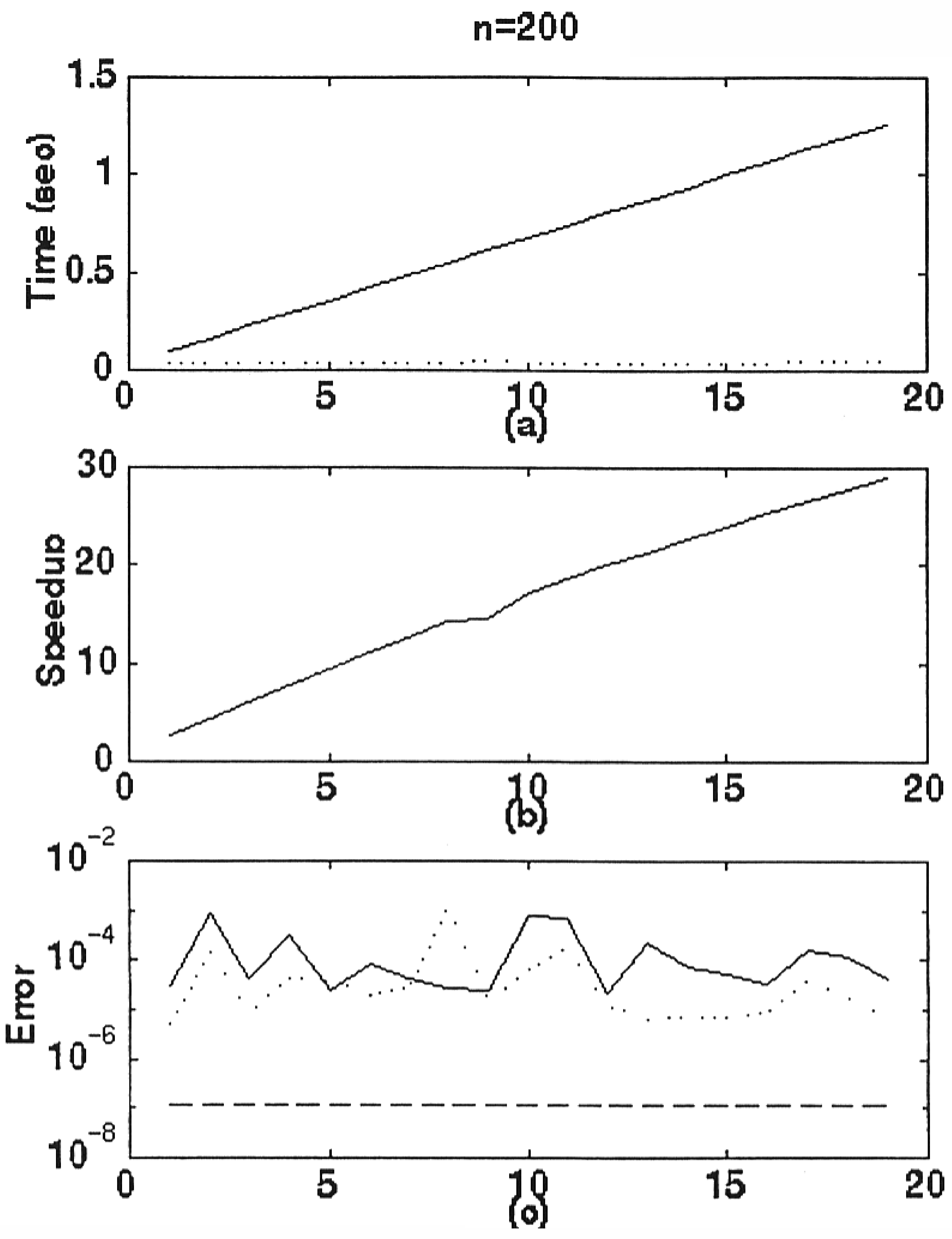

Figure 1: The computational time (a), the speedup (b), and the error (c) on POWER CHALLENGE 8000 for both algorithms: Govaerts-Pryce (continuous line), perturbation approach (dotted line); $n = 200, m = 1, \ldots, 19$, and the dashed line denotes the machine roundoff unit

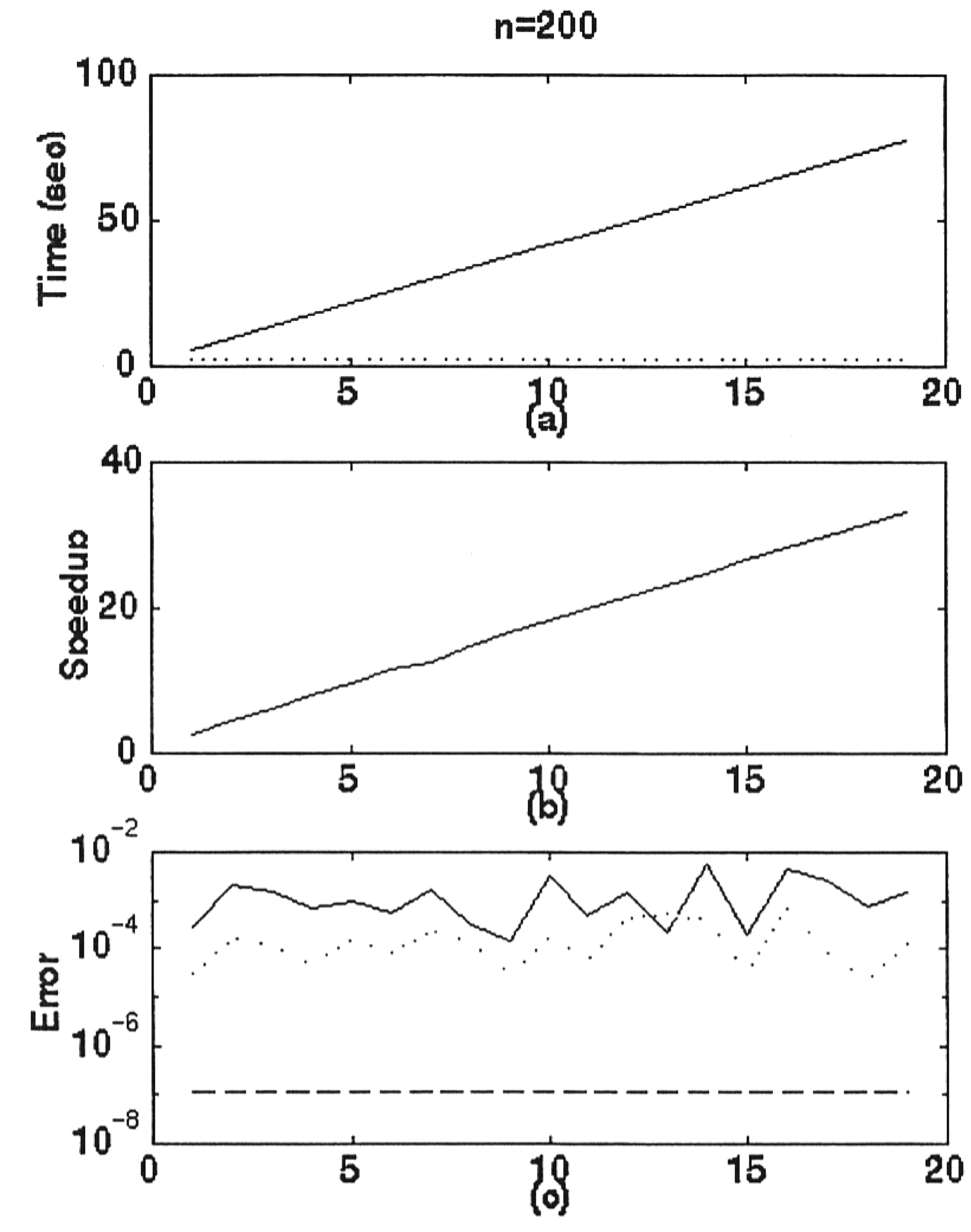

Figure 2: The computational time (a), the speedup (b), and the error (c) on POWER CHALLENGE 8000 for both algorithms: Govaerts-Pryce (continuous line), perturbation approach (dotted line); $n = 900, m = 1, \ldots, 19$, and the dashed line denotes the machine roundoff unit

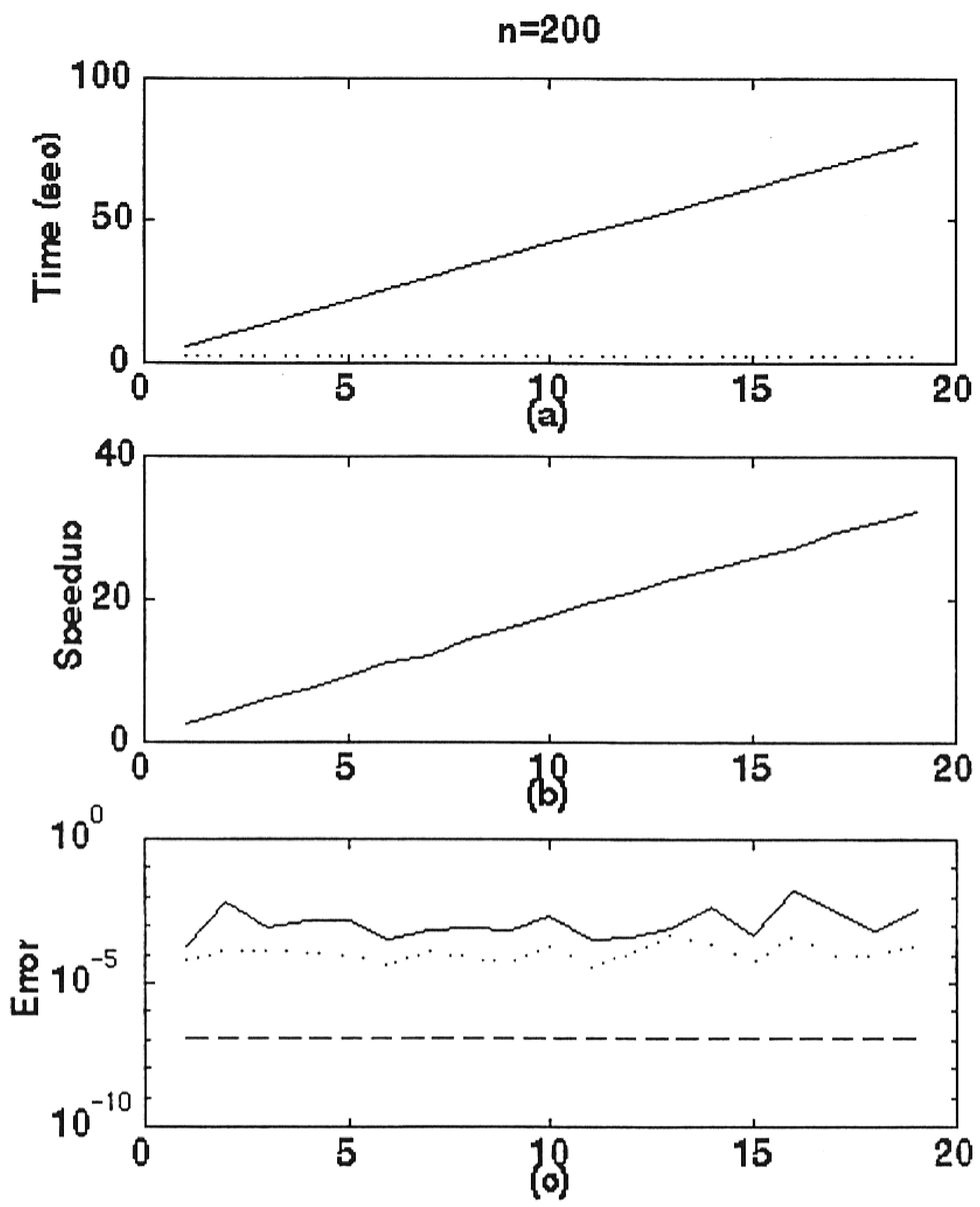

Figure 3: The computational time (a), the speedup (b), and the error (c) on POWER CHALLENGE 10000 for both algorithms: Govaerts-Pryce (continuous line), perturbation approach (dotted line); $n = 200, m = 1, \ldots, 19$, and the dashed line denotes the machine roundoff unit

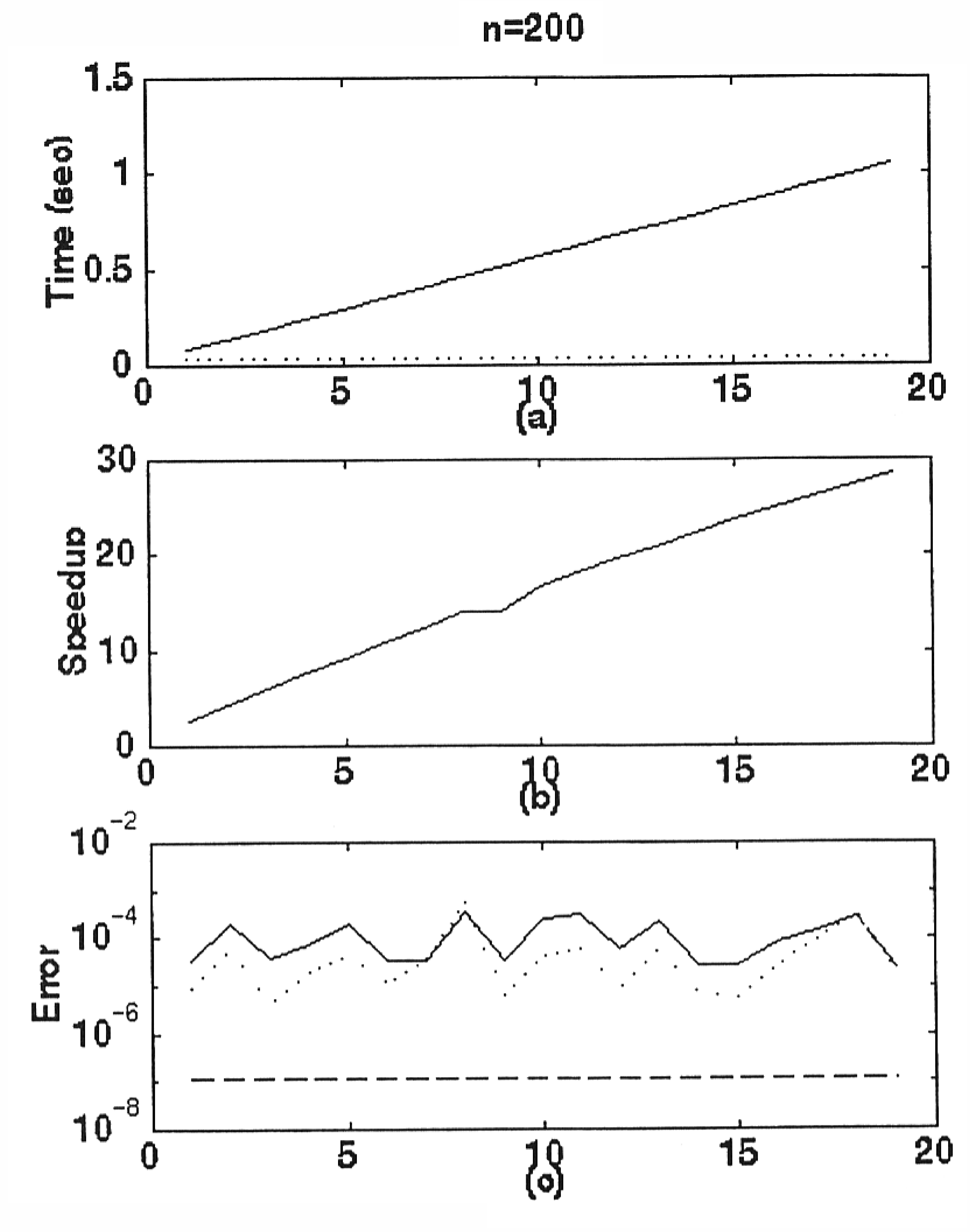

Figure 4: The computational time (a), the speedup (b), and the error (c) on POWER CHALLENGE 10000 for both algorithms: Govaerts-Pryce (continuous line), perturbation approach (dotted line); $n = 900, m = 1, \ldots, 19$, and the dashed line denotes the machine roundoff unit

Let us also note that we used one processor each time on the SGI machines. So, multiprocessing was not an issue in these experiments. The behaviour on both SGI machines was similar because the code is written mostly in terms of BLAS kernels, and these have been optimized on SGI PC 8000, but not on SGI PC 10000.

Acknowledgment

This work was supported partially by Grant MM-434/94 from the Bulgarian Ministry of Education, Science, and Technologies. The work was started under the COBASE program from the National Research Council. Computer time grant from NCSA in Urbana-Champain is kindly acknowledged.

References

[1] S. M. Balle and P. C. Hansen, A Strassen-type matrix inversion algorithm for the Connection Machine, Report UNIC-93-11, October 1993.

[2] J. Barlow and U. Vemulapati, An improved method for one-way dissection with singular diagonal blocks, *SIAM J. Matrix Anal. Appl.*, **11** (1990), pp. 575–588.

[3] T. F. Chan, Newton-like pseudo-arclength methods for computing simple turning points, *SIAM J. Sci. Stat. Comput.*, **5** (1984), pp. 135–148.

[4] T. F. Chan and D. C. Resasco, Generalized deflated block elimination, *SIAM J. Numer. Anal.*, **23** (1986), pp. 913–924.

[5] D. W. Decker and H. B. Keller, Multiple limit point bifurcation, *J. Math. Anal. Appl.*, **75** (1980), pp. 417–430.

[6] P. E. Gill, W. Murray and M. Wright, *Practical optimization*, Academic Press, New York, 1981.

[7] G. Golub and C. Van Loan, *Matrix Computations*, 3rd edition, John Hopkins University Press, Baltimore, 1996.

[8] W. Govaerts, Stable solvers and block elimination for bordered systems, *SIAM J. Matrix Anal. Appl.*, **12** (1991), pp. 469–483.

[9] W. Govaerts and J. D. Pryce, Mixed block elimination for linear systems with wider borders, *IMA J. Numer. Anal.*, **13** (1993), pp. 161–180.

[10] I. N. Hajj and S. Skelboe, A multilevel parallel solver for block tridiagonal and banded linear systems, *Parallel Computing*, **15** (1990), pp. 21–45.

[11] P. C. Hansen, P. Y. Yalamov, Stabilization by perturbation of a $4n^2$ Toeplitz solver, Preprint N25, Technical University of Russe, January 1995.

[12] H. B. Keller, The bordering algorithm and path following near singular points of higher nullity, *SIAM J. Sci. Stat. Comput.*, **4** (1983), pp. 573–582.

[13] B. Werner and A. Spence, The computation of symmetry-breaking bifurcations, *SIAM J. Numer. Anal.*, **21** (1984), pp. 388–399.

[14] B. Werner, Computation of Hopf bifurcation with bordered matrices, *SIAM J. Numer. Anal.*, **33** (1996), pp. 435–455.

[15] J. H. Wilkinson, *The algebraic eigenvalue problem*, Clarendon Press, Oxford, 1965.

[16] P. Y. Yalamov, The bordering and the block bordering method, in: *Advances in Parallel Computing*, (I. Dimov, O. Tonev eds.), IOS Press, 1994, pp. 60–65.