mathematics

Article

ABS-Based Direct Method for Solving Complex Systems of Linear Equations

József Abaffy [1] and Szabina Fodor [2,*]

1 Institute of Applied Mathematics, Óbuda University, Bécsi út 96/b, 1034 Budapest, Hungary; jozsef.abaffy@uni-obuda.hu
2 Department of Computer Science, Corvinus University of Budapest, Fővám tér 13-15, 1093 Budapest, Hungary
* Correspondence: szabina.fodor@uni-corvinus.hu; Tel.: +36-1-482-7468

Abstract: Efficient solution of linear systems of equations is one of the central topics of numerical computation. Linear systems with complex coefficients arise from various physics and quantum chemistry problems. In this paper, we propose a novel ABS-based algorithm, which is able to solve complex systems of linear equations. Theoretical analysis is given to highlight the basic features of our new algorithm. Four variants of our algorithm were also implemented and intensively tested on randomly generated full and sparse matrices and real-life problems. The results of numerical experiments reveal that our ABS-based algorithm is able to compute the solution with high accuracy. The performance of our algorithm was compared with a commercially available software, Matlab's *mldivide* (\) algorithm. Our algorithm outperformed the Matlab algorithm in most cases in terms of computational accuracy. These results expand the practical usefulness of our algorithm.

Keywords: complex linear system; direct method; ABS class of methods

check for updates

Citation: Abaffy, J.; Fodor, S. ABS-Based Direct Method for Solving Complex Systems of Linear Equations. *Mathematics* **2021**, *9*, 2527. https://doi.org/10.3390/math9192527

Academic Editors: Lorentz Jäntschi and Daniela Roșca

Received: 24 August 2021
Accepted: 3 October 2021
Published: 8 October 2021

1. Introduction

The problem of solving systems of linear equations plays a central role in diverse scientific fields such as signal processing, economics, computer science, and physics [1–5]. Often the problems arising from areas of practical life result in a system of equations with real coefficients, but there are important applications that lead to the following complex linear systems:

$$Ax = b, \text{ where } A \in \mathbb{C}^{m,n},\ x \in \mathbb{C}^n \text{ and } b \in \mathbb{C}^m. \tag{1}$$

Partial differential equations modelling dissipative processes usually involve complex coefficient functions or complex boundary conditions [6]. Other applications leading to complex linear systems include the discretization of time-dependent Schrödinger equations with implicit differential equations [7–9], inverse scattering problems, underwater acoustics, eddy current calculations [10], diffuse optical tomography [11], numerical calculations in quantum chromodynamics and numerical conformal mapping [12]. There are several methods to solve complex systems of linear equations. Moreover, a complex linear system with n unknowns can be reformulated to a linear system of equations with $2n$ real coefficients [13].

There are two basic popular approaches in solving complex linear systems of equations. The first one is when the preconditioned classical conjugate gradient method is used to solve the system of equations [14–16]. In most of these cases, the algorithms often do not work with the original general, i.e., non-Hermitian A coefficient matrix, but with a modified Hermitian positive definite normal equations

$$A^H Ax = A^H b, \text{ where } A^H \text{ is the conjugate transpose of the } A \text{ matrix.} \tag{2}$$

The second popular approach is to solve usually non-symmetric linear systems of equations using one of the generalized CG methods [17,18], such as GMRES ([19] pp. 164–185) or other approaches based on the Arnoldi or the Lanczos biconjugate algorithms. These approaches generate an orthogonal basis for the Krylov subspaces associated with A and an initial vector v_1. The usual way to obtain an approximation to the exact solution of (1) from this basis is to force a biconjugate gradient condition. In both cases, the resulting iterative methods tend to converge relatively slowly.

Mainly, two factors influence the practical usability of a method: the accuracy of the solutions and the computation cost of the method. There are two main classes of techniques for solving linear systems of equations: iterative and direct methods.

Iterative solvers are generally faster on large-scale problems, while direct ones give more accurate results. Not surprisingly, iterative methods have come to the fore in solving complex linear systems of equations and many new algorithms have been published in recent decades [20–22]. However, the robustness and the speed of the convergence of iterative algorithms significantly depend on the condition number of the coefficient matrices. To avoid convergence problems, robust preconditioners need to be used. In some cases, problem-specific preconditioners are highly effective, but they are often difficult to parallelize on modern high performance computing (HPC) platforms [23]. Moreover, Koric et al. [24] revealed that no iterative preconditioned solver combination could correctly solve the highly ill-conditioned systems of equations. Therefore, researchers often have to compromise between accuracy of the solutions and computational effort. Direct solvers typically do not have these limitations but require much more computational power to execute. Thus, today's high performance solutions such as parallel computing frameworks, may provide new possibilities for direct algorithms and they can be more suitable choices to ill-conditioned problems.

In this paper, we present a new ABS-based direct method that can solve large-scale problems. The ABS class of methods was originally introduced by Abaffy, Broyden and Spedicato (1984) developed to solve linear systems of equations where coefficients of the equations are real numbers [25]. These algorithms can also be used for various other purposes such as solving non-linear systems of equations or optimization problems [26–28]. ABS-based algorithms can be easily and effectively parallelised [29,30] which underlines the practical usefulness of these algorithms.

The remainder of this paper is organized as follows. In Section 2, we present our new scaled complex ABS algorithm, which can solve complex linear systems of equations and we prove some of its basic features. We also show a special choice of parameters of our algorithm, which ensures that the search vectors (p_i) are $A^H A$ conjugate. We provide a detailed numerical analysis of four variants of our algorithm in Section 3. Section 4 contains concluding remarks and a brief description of our plans on the application of the outcomes of this work.

2. The Scaled Complex ABS (scABS) Algorithm

In this section, we present our new ABS-based algorithm (scABS) that can solve systems of complex linear problems. Instead of the original system (1) let us consider the following scaled complex system of linear equations

$$V^H A x = V^H b, \text{ where } V \in \mathbb{C}^{m,n} \text{ is an arbitrary, non-singular matrix.} \tag{3}$$

Note that the systems (1) and (3) are equivalent. Thus, if we want our algorithm to solve the original unscaled system of Equation (1), then by choosing the scaling matrix as the unit matrix we obtain the desired formulas and values. The sole role of the scaling matrix V is to provide significant flexibility to the algorithm [25] which allows us to reduce the computational inaccuracy [31] and also to reduce the computational complexity of the algorithm [27].

It is also of interest to note that our scaled complex ABS algorithm defines not one single algorithm, but a class of variants. Each particular variant is determined by the choice of the parameters H_1, v_i, z_i, w_i. These variants have different properties, e.g., by choosing $H_1 = I$ unit matrix, $v_i = AH_i^H z_i$, and w_i to be a multiple of z_i, the ABS algorithm is a reformulation of the QR algorithm, as we will discuss in Section 2.2.

The state of the complex system of linear equations is checked by the variable $s_i = H_i A^H v_i$. An important property of our algorithm is that s_i is zero if, and only if, the current row of matrix A is a linear combination of the previous rows. It depends on the value of the right-hand side whether our system is then a linear combination of the previous equations or even incompatible with them. We use the value of x_i to distinguish between the two cases, namely whether it solves the ith equation or not. If it does, we simply skip the equation ($x_{i+1} = x_i$, $H_{i+1} = H_i$), otherwise, we stop the algorithm. The state of the system is stored in the $iflag$ variable. If $iflag$ is zero, then neither linear dependency nor incompatibility is detected in the algorithm. If the value of $iflag$ is positive, then a number of linearly dependent equations are found. If the value of $iflag$ is negative then the $-iflagth$ equation is incompatible with the previous ones.

H_1 was selected to be a unit matrix for the sake of simplicity. However, H_1 could theoretically be any arbitrary unimodular nonsingular matrix provided that $H_1 \in \mathbb{C}^{n,n}$.

2.1. Basic Properties of the Scaled Complex ABS Algorithm

We consider some fundamental properties of the H_i generated by the scaled complex ABS (scABS) algorithm (Algorithm 1).

Theorem 1. *Given the $A^H v_1, \ldots, A^H v_m \in \mathbb{C}^n$ vectors and the $H_1 \in \mathbb{C}^{n,n}$ arbitrary nonsingular matrix, consider the sequence of matrices $H_2, \ldots, H_{m+1}$ generated by (6). The following relations are true for $i = 2, \ldots, m+1$:*

$$H_i A^H v_j = 0, \quad 1 \le j < i. \tag{4}$$

Proof. We proceed by induction. For $i = 2$, the theorem is true since

$$H_2 = H_1 - \frac{H_1 A^H v_1 * w_1^H H_1}{\widetilde{w_1}} \overline{w_1^H H_1 A^H v_1}.$$

$$\begin{aligned}
H_2 A^H v_1 &= H_1 A^H v_1 - \frac{H_1 A^H v_1 * w_1^H H_1}{\widetilde{w_1}} \overline{w_1^H H_1 A^H v_1} * A^H v_1 \\
&= H_1 A^H v_1 - \frac{H_1 A^H v_1 * w_1^H H_1 A^H v_1}{\widetilde{w_1}} \overline{w_1^H H_1 A^H v_1} \\
&= H_1 A^H v_1 - \frac{H_1 A^H v_1 * \widetilde{w_1}}{\widetilde{w_1}} = 0.
\end{aligned}$$

Assuming that the theorem is true up to $i < m$, we prove it for $i + 1$.

$$H_{i+1} = H_i - \frac{H_i A^H v_i * w_i^H H_i}{\widetilde{w_i}} \overline{w_i^H H_i A^H v_i}$$

$$\begin{aligned}
H_{i+1} A^H v_j &= H_i A^H v_j - \frac{H_i A^H v_i * w_i^H H_i}{\widetilde{w_i}} \overline{w_i^H H_i A^H v_i} * A^H v_j \\
&= H_i A^H v_j - \frac{H_i A^H v_i * w_i^H H_i A^H v_j}{\widetilde{w_i}} \overline{w_i^H H_i A^H v_i} \\
&= H_i A^H v_j - \frac{H_i A^H v_i * \widetilde{w_i}}{\widetilde{w_i}} = 0.
\end{aligned}$$

□

Algorithm 1: Scaled complex ABS (scABS) algorithm.

```
Input: Set $x_1 \in \mathbb{C}^n$, $H_1 = I \in \mathbb{C}^{n,n}$ where $I$ is the unit matrix, $i = 1$, and $iflag = 0$.
Output: Solution to (3) $x_i \in \mathbb{C}^n$, if the solution exists, otherwise the $-iflag$th
        equation incompatible with the previous ones.
while $(i \leq m)$ and $(iflag \geq 0)$ do
    Compute the scalar $\tau_i = v_i^H r_i \in \mathbb{C}$ and the vector $s_i = H_i A^H v_i$, where
     $r_i = Ax_i - b$.
    if $(s_i = 0)$ and $(\tau_i = 0)$ and $(i < m)$ then
        /* The $ith$ equation linearly depends on the previous ones.            */
        $p_i = 0,$
        $x_{i+1} = x_i,$
        $H_{i+1} = H_i,$
        $iflag = iflag + 1.$
    else if $(s_i = 0)$ and $(\tau_i = 0)$ and $(i = m)$ then
        $x_m$ is the solution.
    else if $(s_i = 0)$ and $(\tau_i \neq 0)$ then
        /* The $ith$ equation is incompatible with the previous ones.           */
        $iflag = -i.$
    else
        Compute the search direction $p_i$
            $$p_i = H_i^H z_i, \quad (5)$$
        where $z_i \in \mathbb{C}^n$ is arbitrary with the condition $z_i^H H_i A^H v_i \neq 0$.
            $$x_{i+1} = x_i - \alpha_i p_i.$$
        Let introduce the notation $\widetilde{\alpha_i}$ as the denominator of $\alpha_i$
            $$\widetilde{\alpha_i} = v_i^H A p_i * \overline{v_i^H A p_i},$$
        where $\overline{v_i^H A p_i}$ is the complex conjugate of $v_i^H A p_i$ vector. Therefore, $\widetilde{\alpha_i}$ is a
        real number and the step size $\alpha_i$ is given by
            $$\alpha_i = \frac{\tau_i}{\widetilde{\alpha_i}} * \overline{v_i^H A p_i}.$$
        Update the projection matrix. Compute
            $$H_{i+1} = H_i - \frac{H_i A^H v_i * w_i^H H_i}{\widetilde{w_i}} \overline{w_i^H H_i A^H v_i}, \quad (6)$$
        where we use the notion $\widetilde{w_i}$ as denominator of $w_i$. $w_i \in \mathbb{C}^n$ is arbitrary with
        the condition
            $$\widetilde{w_i} = w_i^H H_i A^H v_i * \overline{w_i^H H_i A^H v_i} \neq 0.$$
    $i = i + 1$
```

Remark 1. *The null space of complex projection matrices $Null(H_i) = \{A^H v_1, A^H v_2, \ldots, A^H v_{i-1}\}$ follows immediately from Theorem 1 and it means that $H_{m+1} = 0$*

Remark 2. *$H_i A^H v_i$ computed by the scABS algorithm is zero only if $A^H v_i$ is a linear combination of $A^H v_1, A^H v_2, \ldots, A^H v_{i-1}$.*

Remark 3. *The scaled complex ABS algorithm is well defined.*

Theorem 2. *Consider the matrices H_i generated by (6) with the starting H_1 is nonsingular. For $i = 1, \dots, m+1$ and $1 \le j \le i$ the following relations are true:*

$$H_i H_1^{-1} H_j = H_i, \tag{7}$$

$$H_j H_1^{-1} H_i = H_i. \tag{8}$$

Proof. We only prove (7) since the proof for (8) is similar. We proceed by induction. For $i = 1$, $H_1 H_1^{-1} H_1 = H_1$ is trivially true. Assuming now that (7) is true up to the index i, we have to prove that $H_{i+1} H_1^{-1} H_j = H_{i+1}$.

For $j = i+1$, we have

$$\begin{aligned}
H_{i+1} H_1^{-1} H_{i+1} = {} & (H_i - \frac{H_i A^H v_1 * w_i^H H_i}{\widetilde{w}_i} \overline{w_i^H H_i A^H v_i}) H_1^{-1} \\
& * (H_i - \frac{H_i A^H v_i * w_i^H H_i}{\widetilde{w}_i} \overline{w_i^H H_i A^H v_i}) \\
& = H_i H_1^{-1} H_i - \frac{H_i A^H v_i * w_i^H H_i H_1^{-1} H_i}{\widetilde{w}_i} \overline{w_i^H H_i A^H v_i} \\
& - \frac{H_i H_1^{-1} H_i A^H v_i * w_i^H H_i}{\widetilde{w}_i} \overline{w_i^H H_i A^H v_i} \\
& + \frac{H_i A^H v_i * w_i^H H_i}{\widetilde{w}_i} \overline{w_i^H H_i A^H v_i} H_1^{-1} * \frac{H_i A^H v_i * w_i^H H_i}{\widetilde{w}_i} \overline{w_i^H H_i A^H v_i} \\
& = H_i H_1^{-1} H_i - 2 * \frac{H_i A^H v_i * w_i^H H_i H_1^{-1} H_i}{\widetilde{w}_i} \overline{w_i^H H_i A^H v_i} + \\
& \frac{H_i A^H v_i * w_i^H H_i}{\widetilde{w}_i} H_1^{-1} * \frac{H_i A^H v_i * w_i^H H_i}{\widetilde{w}_i} (\overline{w_i^H H_i A^H v_i})^2 \\
& = H_i - 2 * \frac{H_i A^H v_i * w_i^H H_i}{\widetilde{w}_i} \overline{w_i^H H_i A^H v_i} \\
& + \frac{H_i A^H v_i * \widetilde{w}_i * w_i^H H_i}{\widetilde{w}_i^{\,2}} \overline{w_i^H H_i A^H v_i} \\
& = H_i - \frac{H_i A^H v_i * w_i^H H_i}{\widetilde{w}_i} \overline{w_i^H H_i A^H v_i} = H_{i+1}.
\end{aligned}$$

For $j < i+1$, we have

$$\begin{aligned}
H_{i+1} H_1^{-1} H_j &= (H_i - \frac{H_i A^H v_1 * w_i^H H_i}{\widetilde{w}_i} \overline{w_i^H H_i A^H v_i}) H_1^{-1} H_j \\
&= H_i - \frac{H_i A^H v_1 * w_i^H H_i}{\widetilde{w}_i} \overline{w_i^H H_i A^H v_i} = H_{i+1}
\end{aligned}$$

again by the induction. □

Remark 4. *If H_1 is the unit matrix then Theorem 2 implies that H_i matrices are idempotent, i.e., $H_i H_i = H_i$.*

Remark 5. *The $H_i H_i = H_i$ feature of our algorithm enables further potential improvement in computational accuracy of the scaled complex ABS algorithm. We have recently published that the reprojection of real ABS algorithms, i.e., using $p_i = H_i^H (H_i^H z_i)$ projection vectors instead of $p_i = H_i^H z_i$, enhances the precision of numerical calculation [30,31], which expands the practical usefulness of our algorithms.*

Theorem 3. *Any vector $y \in \mathbb{C}^n$ that solves the first i equations of (3) can be formulated as*

$$y = x_{i+1} + H_{i+1}^H s \tag{9}$$

where s is a vector in $\mathbb{C}^n$.

Proof. Decompose $y \in \mathbb{C}^n$ as $y = x_{i+1} + y^+$. As y solves the first i equations,

$$A^H v_j y = b^H v_j = A^H v_j x_{i+1} + A^H v_j y^+, \text{ where } 1 \leq j \leq i. \tag{10}$$

Thus, $A^H v_j y^+ = 0$ for $1 \leq j \leq i$ which means that $y^+ \perp \{A^H v_1, .., A^H v_i\} = \perp Null$ (H_{i+1}), so $y^+ = H_{i+1}^H r$ where $r \in \mathbb{C}^n$ according to the properties of the scaled complex ABS algorithm. So $y^+ = H_{i+1}^H r$ where $r \in \mathbb{C}^n$ according to the properties of ABS.

$$y^+ = H_{i+1}^H r = H_{i+1}^H \cdot H_{i+1}^H r = H_{i+1}^H y^+ \text{ where } y^+ \in \mathbb{C}^n. \tag{11}$$

□

Remark 6. *Referring to the proofs of the original ABS algorithm [26], it can be shown that, for simplicity, assuming that the A matrix is full rank, $L_i = V_i^H A_i P_i$ is a nonsingular lower triangular matrix where $P_i = (p_1, \ldots, p_i)$, $V_i = (v_1, \ldots, v_i)$ and $A_i = (a_1^H, \ldots, a_i^H)$ the transpose of the rows of the matrix A. If $i = m$, then the following semifactorization of the inverse can be obtained:*

$$A^{-1} = PL^{-1}V^H, \textit{ where } P = P_m, \textit{ and } V = V_m. \tag{12}$$

This semi-factorization of the inverse of the A matrix may provide an opportunity to develop an ABS-based preconditioner to accelerate various Krylov subspace methods. For several choices of the matrix V, the matrix L is diagonal, hence formula (12) gives an explicit factorization of the inverse of the coefficient matrix A.

Remark 7. *Examining the properties of the scaled complex ABS algorithm in terms of practical usability, we have already mentioned in Remark 4 that the idempotent property of the H_i matrix can be used to increase the computational accuracy of our algorithm by reprojection stepwise. Broyden showed [32] that the computation error in the solution is reduced up to 2 orders in $\frac{\|A^T v_i\|_2}{\|H_i A^T v_i\|_2}$ for the real ABS algorithm. If the projection vectors (p_i) are re-projected with an additional computational cost, i.e., $p_i = H_i^T z_i = H_i^T H_i^T z_i$, a more accurate result can be obtained due to the cancellation errors in computational accuracy [33,34]. Interestingly, our preliminary results showed that the scABS algorithm has similar properties. Increasing accuracy in this way will result in a significant increase in computational costs. One solution to speed up the algorithm may be to parallelize the processes. In a recent paper [30], we found that for a real ABS algorithm, parallelization yielded a significant gain in computational speed.*

2.2. Special Choice of the Scaling Vector

We consider the following choice of the scaling vector:

$$v_i = Ap_i. \tag{13}$$

This selection of the scaling vector has many beneficial features such as

- the scaling vectors defined by (13) are mutually orthogonal, i.e., $v_i^H v_j = 0$ for $i > j$ and $v_i^H v_i \neq 0$.
- the search vectors p_i are $A^H A$ conjugate if A is square and nonsingular matrix.
- the algorithm generates an implicit factorization of A into the product of an orthogonal and an upper triangular matrix if w_i is a multiple of z_i and $H_1 = I$ (unit matrix). The algorithm with these selections can be considered as an alternative method of the classic QR factorization.

- by the choice of $H_1 = I$, $z_i = w_i = e_i$, the required number of multiplication is $\frac{11}{6}n^3 + \mathcal{O}(n^2)$.

The above statements about of the oscABS algorithm (Algorithm 2) can be easily verified based on the proofs of Theorem 8.11, Theorem 8.16 and Theorem 8.18 in [25].

Algorithm 2: Orthogonally scaled complex ABS (oscABS) algorithm.

```
Input: Set $x_1 \in \mathbb{C}^n$, $H_1 = I \in \mathbb{C}^{n,n}$ where $I$ is the unit matrix; $i = 1$ and $iflag = 0$.
Output: Solution to (3) $x_i \in \mathbb{C}^n$, if the solution exists, otherwise the $-iflag$th
        equation incompatible with the previous ones.
while $(i \leq m)$ and $(iflag \geq 0)$ do
    Compute the search direction
        $$p_i = H_i^H z_i, \qquad (14)$$
     where $z_i \in \mathbb{C}^n$ is arbitrary with condition $AH_i^H z_i \neq 0$.
    Compute the scalar $\tau_i = (Ap_i)^H r_i$ and the vector $s_i = H_i A^H A p_i$.
    if $(s_i = 0)$ and $(\tau_i = 0)$ and $(i < m)$ then
        /* The $ith$ equation linearly depends on the previous ones.   */
        $p_i = 0$,
        $x_{i+1} = x_i$,
        $H_{i+1} = H_i$,
        $iflag = iflag + 1$.
    else if $(s_i = 0)$ and $(\tau_i = 0)$ and $(i = m)$ then
        $x_m$ is the solution.
    else if $(s_i = 0)$ and $(\tau_i \neq 0)$ then
        /* The $ith$ equation is incompatible with the previous ones.   */
        $iflag = -i$.
    else
        Update the approximate solution
            $$x_{i+1} = x_i - \alpha_i p_i.$$
         Let us introduce the notation $\widetilde{\alpha_i}$ as the denominator of $\alpha_i$
            $$\widetilde{\alpha_i} = p_i^H A^H A p_i * \overline{p_i^H A^H A p_i},$$
         where $\overline{p_i^H A^H A p_i}$ is the complex conjugate of $p_i^H A^H A p_i$ vector. Therefore,
         $\widetilde{\alpha_i}$ is a real number and the step size $\alpha_i$ is given by
            $$\alpha_i = \frac{\tau_i}{\widetilde{\alpha_i}} * \overline{p_i^H A^H A p_i}.$$
         Update the projection matrix. Compute
            $$H_{i+1} = H_i - \frac{H_i A^H A p_i * w_i^H H_i}{\widetilde{w_i}} \overline{w_i^H H_i A^H A p_i}, \qquad (15)$$
         where we use the notion $\widetilde{w_i}$ as denominator of $w_i$. $w_i \in \mathbb{C}^n$ is arbitrary
         vector with the condition
            $$\widetilde{w_i} = w_i^H H_i A^H A p_i * \overline{w_i^H H_i A^H A p_i} \neq 0.$$
    $i = i + 1$
```

3. Numerical Experiments

We were also interested in the numerical features of our scaled complex ABS algorithm. To this end, four variants of the orthogonally scaled complex ABS algorithm were implemented in MatlabR2013b (Mathworks, Inc., USA). The experiments were performed on a personal computer with Intel Core i7-2600 3.4GHz CPU with integrated graphics, 4 GB RAM running Microsoft Windows 10 Professional and MATLAB version R2013b. No software other than the operating system tasks, MATLAB and ESET NOD32 antivirus were running during the experiments.

The four implemented variants of the oscABS algorithm and the Matlab function used are as follows:

- S3rr: z_i and w_i are selected such that $z_i = r_i$, $w_i = r_i$.
- S3ATA: z_i and w_i are selected such that $z_i = A^H r_i$, $w_i = A^H r_i$.
- S3ee: z_i and w_i are selected such that $z_i = e_i$, $w_i = e_i$.
- S3ep: z_i and w_i are selected such that $z_i = e_i$, $w_i = H_i^H p_i$.
- Matlab: the built-in *mldivide* (\) function of Matlab.

We evaluated all methods in aspects of accuracy of the computed solution $\|Ax_n - b\|_2$, and execution time in seconds.

3.1. Randomly Generated Problems

In our first numerical experiments, we tested the variants of our algorithm on randomly generated dense and sparse matrices. These matrices were generated using the `rand` and `sprand` MATLAB functions. In general, we performed 10 separate tests on independent matrices to obtain each data point in this subsection. The mean values from those results are depicted in the following figures. The standard variation of the values usually fell within well below 5% of the mean value (data not shown). The solutions were randomly generated using the `rand` function and the right sides of the systems were calculated as the products of the matrix and the solution. In these experiments, we tested how the performance of our algorithm was affected by gradually increasing the dimension of the coefficient matrix from 10×10 to 1500×1500. As shown in Figure 1, each of the variants of the ABS-based algorithm was able to solve the systems of equations within 10^{-10} accuracy, and overall, the S3ee variant was the most accurate. We did not see a large difference up to 1500 dimensions between the four variants of the orthogonally scaled complex ABS algorithm.

Our next aim was to compare the performance of the best variant of oscABS algorithm (S3ee) with a commercially available software able to solve complex linear systems of equations. We chose the *mldivide* (\) algorithm of the Matlab software package from MathWorks, Inc. for that purpose. Note that Matlab's *mldivide* algorithm is not one specific algorithm, but consists of several algorithms, from which it chooses depending on the property of the matrix A. The *mldivide* function performs several checks on the coefficient matrix to determine whether it has some special property (e.g., whether it is sparse or symmetric) and, knowing this, selects the appropriate matching solver. For example, if A is full but not a square matrix, then it uses the QR algorithm, if not, then depending on the properties of A, possibly the Hessenberg, Cholesky, LU, or even the LDL solver. This means that we compared our algorithm with different solvers, selected to be most appropriate for the given problem by the Matlab program [35].

The experiments were performed with the randomly generated matrices described above.

Our results are summarized in Figure 2. The S3ee algorithm outperformed Matlab algorithm in both full rank, indefinit and rank-deficient problems as well. It is clear from Figure 2 that the difference increases significantly as the dimension increases.

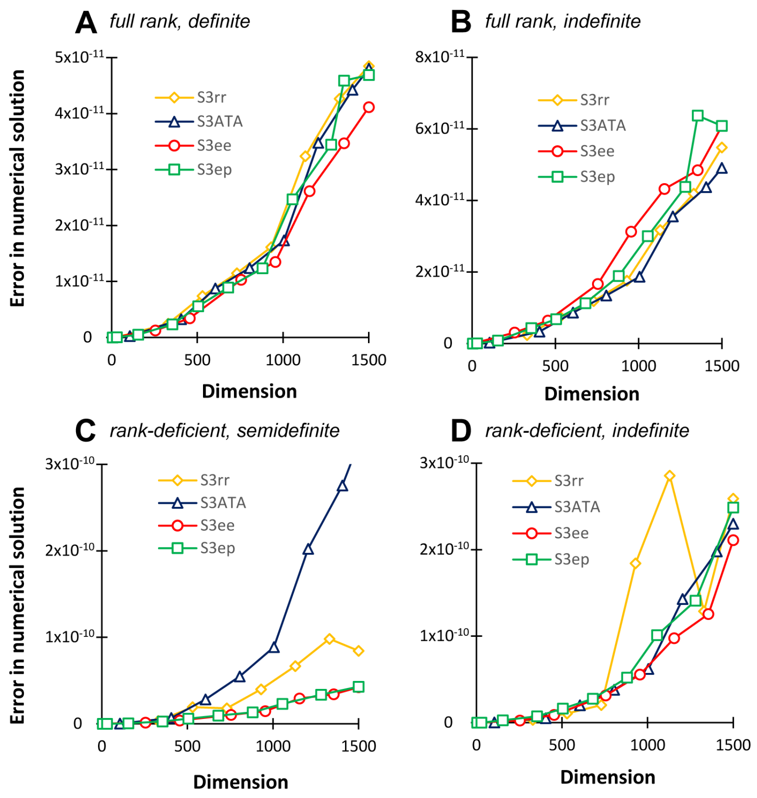


Figure 1. Comparative analysis of the four variants of the orthogonally scaled complex ABS algorithm on randomly generated dense complex systems of linear equations.

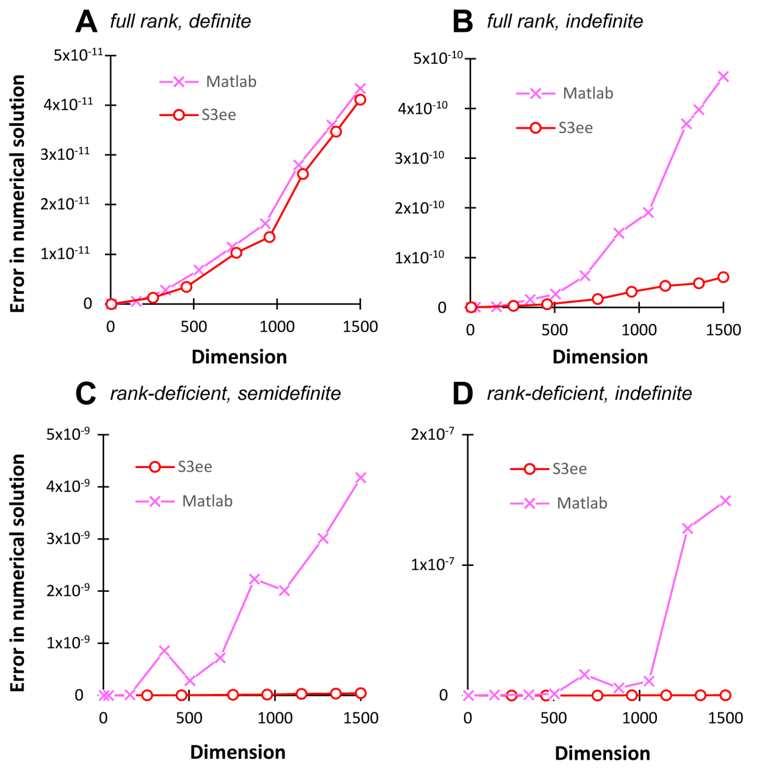


Figure 2. Comparative analysis of the S3ee implementation of the orthogonally scaled ABS algorithm and the Matlab *mldivide* function on randomly generated dense complex systems of linear equations.

Our next set of experiments focused on the analysis of the computational accuracy of the algorithms under different matrix densities. These experiments were performed with uniformly distributed random numbers of the coefficient matrix at different densities. As shown in Figure 3, the S3ATA implementation calculated the solution most accurately and the S3rr the least accurately in most cases. The other two implementations of the orthogonally scaled complex ABS algorithms worked similar to S3ATA (i.e., within the same order of magnitude). Even at 1500 dimensions, these algorithms were able to calculate the result with an accuracy of 10^{-11}. It is also worth noting that reducing the density from 100% to 1% resulted in a significant improvement in computational accuracy. The accuracy increased from 10^{-11} to 10^{-13} for the S3ATA, S3ee, and S3ep variants.

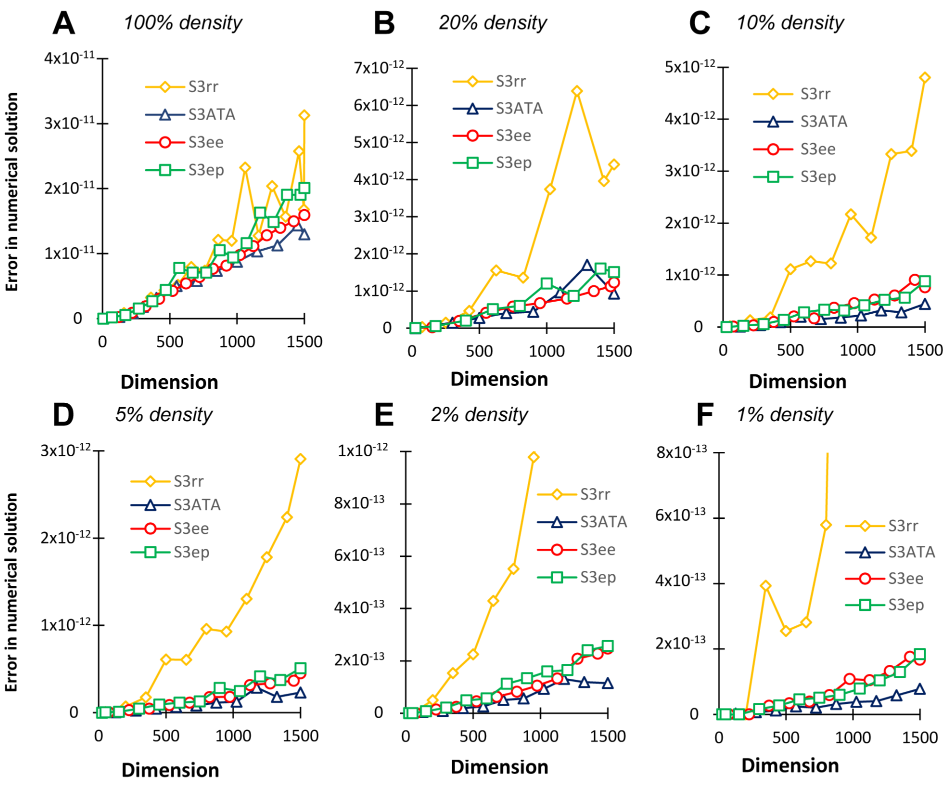


Figure 3. Comparative analysis of the four variants of orthogonally scaled complex ABS algorithm on randomly generated matrices with different densities.

Next, we compared the computational accuracy of our ABS-based algorithm with the Matlab algorithm on problems of different densities. As shown in Figure 4, the S3ee variant of orthogonally scaled complex ABS algorithms outperformed the Matlab algorithm again. It is worth noting that in the case of the Matlab algorithm, the change in matrix density resulted in a moderate improvement in the accuracy of the computations, as it increased from 10^{-11} to 10^{-12}.

In order to gain a deeper understanding of the numerical characteristics of the four variants of our algorithm, we compared the computational accuracy and execution time required to solve the 1500-dimensional systems of equations (Figure 5).

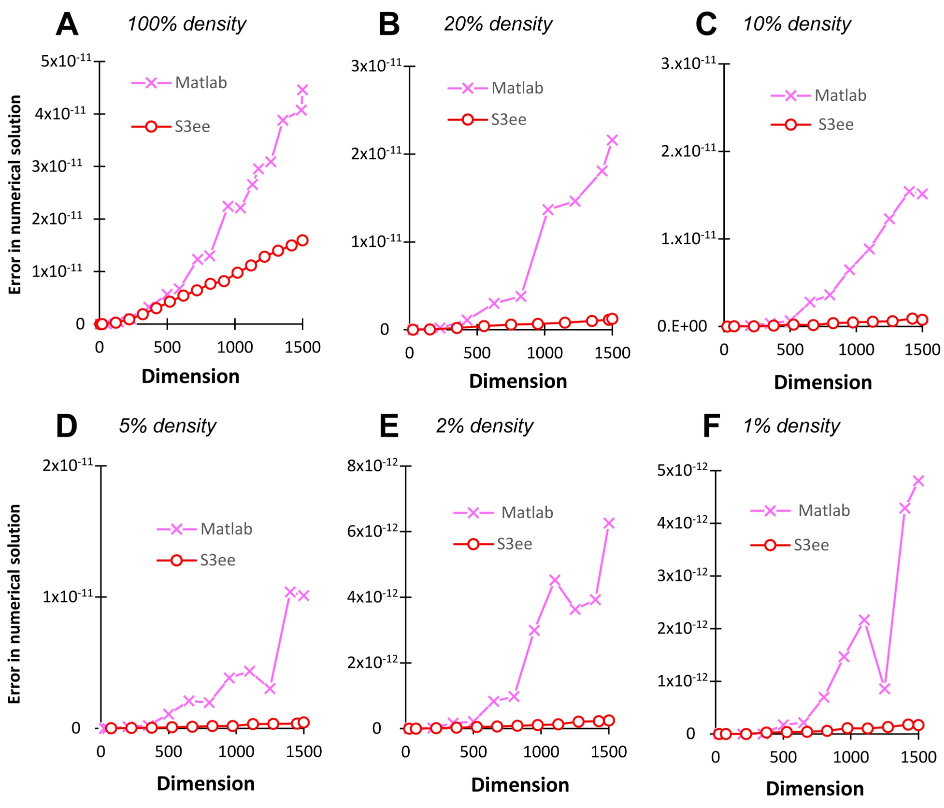


Figure 4. Comparative analysis of the computational accuracy of the ABS-based S3ee algorithm and the Matlab *mldivide* (\) algorithm on randomly generated different-density problems.

A Error in numerical solution

	Problem	Matlab	S3rr	S3ATA	S3ee	S3ep
dense	full rank, definite	$4.3\text{x}10^{-11}$	$4.8\text{x}10^{-11}$	$4.8\text{x}10^{-11}$	$4.1\text{x}10^{-11}$	$4.7\text{x}10^{-11}$
dense	full rank, indefinite	$4.6\text{x}10^{-10}$	$5.5\text{x}10^{-11}$	$4.9\text{x}10^{-11}$	$6.1\text{x}10^{-11}$	$6.1\text{x}10^{-11}$
dense	rank-deficient, semidefinite	$4.2\text{x}10^{-9}$	$8.4\text{x}10^{-11}$	$3.2\text{x}10^{-10}$	$4.2\text{x}10^{-11}$	$4.3\text{x}10^{-11}$
dense	rank-deficient, indefinite	$1.5\text{x}10^{-7}$	$2.6\text{x}10^{-10}$	$2.3\text{x}10^{-10}$	$2.1\text{x}10^{-10}$	$2.5\text{x}10^{-10}$
sparse	*20% density*	$2.2\text{x}10^{-11}$	$4.4\text{x}10^{-12}$	$9.3\text{x}10^{-13}$	$1.2\text{x}10^{-12}$	$1.5\text{x}10^{-12}$
sparse	*10% density*	$1.5\text{x}10^{-11}$	$4.8\text{x}10^{-12}$	$4.5\text{x}10^{-13}$	$7.7\text{x}10^{-13}$	$8.8\text{x}10^{-13}$
sparse	*5% density*	$1.0\text{x}10^{-11}$	$2.9\text{x}10^{-12}$	$2.3\text{x}10^{-13}$	$4.5\text{x}10^{-13}$	$5.1\text{x}10^{-13}$
sparse	*2% density*	$6.3\text{x}10^{-12}$	$2.3\text{x}10^{-12}$	$1.2\text{x}10^{-13}$	$2.5\text{x}10^{-13}$	$2.6\text{x}10^{-13}$
sparse	*1% density*	$4.8\text{x}10^{-12}$	$4.0\text{x}10^{-12}$	$7.8\text{x}10^{-14}$	$1.7\text{x}10^{-13}$	$1.8\text{x}10^{-13}$

B Execution time (s)

	Problem	S3rr	S3ATA	S3ee	S3ep
dense	full rank, definite	21.5	20.8	19.7	23.8
dense	full rank, indefinite	80.4	77.2	70.5	85.7
dense	rank-deficient, semidefinite	19.3	17.6	16.4	20.2
dense	rank-deficient, indefinite	18.0	17.4	16.4	20.2
sparse	*20% density*	22.1	22.0	21.5	24.0
sparse	*10% density*	11.9	11.0	10.3	12.7
sparse	*5% density*	18.8	19.3	16.8	18.8
sparse	*2% density*	17.0	15.9	15.2	17.7
sparse	*1% density*	16.3	15.2	14.8	17.9

Smallest value — Largest value

Figure 5. Numerical properties of the four ABS-based variants and Matlab algorithm for solving 1500-dimensional random systems of equations. Panel (**A**) shows the computational accuracy. Panel (**B**) shows the execution times for our ABS-based variants in seconds. Note that the execution times of Matlab algorithms are not presented, since the Matlab built-in functions are implemented in C, which allows the algorithms to run significantly faster than our programs written in Matlab script regardless of the actual numerical performance of the algorithm.

Figure 5 shows that the numerical properties of the S3ee variant are the best, both in terms of computational accuracy and execution time. This may be explained by the fact that no computation is needed for parameter definitions (z_i, w_i) since the appropriate unit vectors are used and the lack of computation may also explain the accuracy since round-off errors do not accumulate. This round-off error may also explain the relatively poor numerical performance of S3ATA and S3rr.

To obtain a more comprehensive view of the numerical properties of our algorithms, we also investigated their computational accuracy for large, i.e., more than 5000 dimension problems. The dimensions of these problems were determined according to Golub and Van Loan's suggestion [36] that for each problem, the q value, which is the characteristic of the problem, remains substantially below 1. The $q = u \cdot k_\infty(A)$, where u (The value of the unit round-off in Matlab is 2^{-52}) is unit round-off and $k_\infty(A)$ condition number of the coefficient matrix A in infinity-norm. Our results for the high-dimensional problems are summarized in Figures S1 and S2 in the Supplementary Material. Our experiments in high dimensions clearly showed that the different variants of the ABS-based algorithm solved the random problems significantly more accurately than Matlab solver. This is especially true in the rank-deficient cases, where for 5000 dimensions the Matlab function was not able to solve any problem.

3.2. Chosen Test Problems from MATLAB Gallery

Our next experiments compared the computation accuracy of our algorithm on known complex matrices in the Matlab Gallery. Table 1 summarizes the problems we chose.

Table 1. Summary of test problems chosen from MATLAB Gallery.

Name	**Description**
Symmetric Toeplitz	Diagonal-constant matrix using Matlab `toeplitz(r)` function where r is the first row. The vector r was randomly generated in this experiment.
Non-symmetric Toeplitz	Diagonal-constant matrix using Matlab `toeplitz(c,r)` function where c is first column, r is the first row of the matrix. The vector c, and the vector r were generated randomly.
Hankel	Symmetric and constant across the anti-diagonals matrix using Matlab `hankel(c)` function where c defines the first column of the matrix. The c column was generated randomly in this experiment.
Smoke	Complex matrix with a 'smoke ring' pseudospectrum using Matlab `gallery('smoke',n)` function where n is the dimension of the matrix.

For each matrix, we tested the dimensions between 10–1500. Experiments presented in Figure 6 revealed that all four variants of the complex ABS algorithm were able to solve the problems with an accuracy of approximately 10^{-10} and in most cases the computation error remained significantly below 10^{-11}. Examining the different variants of the orthogonally scaled complex ABS algorithm, we found that the S3ee and S3ep algorithms computed the problems with similar accuracy, with the S3ATA variant being the least accurate except for the Smoke problem.

Next, we compared the S3ee variant with the Matlab algorithm (see Figure 7). The ABS-based algorithm slightly outperformed the Matlab algorithm on all but the Smoke matrices.

In order to gain a deeper understanding of the numerical characteristics of the four variants of our algorithm, we compared the computational accuracy and execution time required to solve the 1500-dimensional systems of equations (see Figure 8).

For Matlab Gallery problems, the running properties of the algorithms are slightly different from those seen for randomly generated problems. For the first three matrix (Symmetric Toeplitz, Non-symmetric Toeplitz, Hankel) problems, we see similar results as for the randomly generated problems, the ABS-based variants giving more accurate results than the Matlab algorithm. However, for the fourth, Smoke matrix, for the first time Matlab's algorithm gave the most accurate result tested. This fact can be partly explained by the special structure of the Smoke matrix since the diagonal of the n-dimensional Smoke matrix consists of the set of all nth roots of unity, and 1's on its superdiagonal and 1 in the (n,1) position. It is interesting to note that the Matlab algorithm and the S3ATA variant behaved very similarly in these problems, both computing the solution with relatively large errors for the first three problems, but giving the most accurate solutions for the Smoke matrices. In the case of running speeds, it is clear that the lower computational demand of the S3ee variant in the 1500 dimension already resulted in significantly shorter running times.

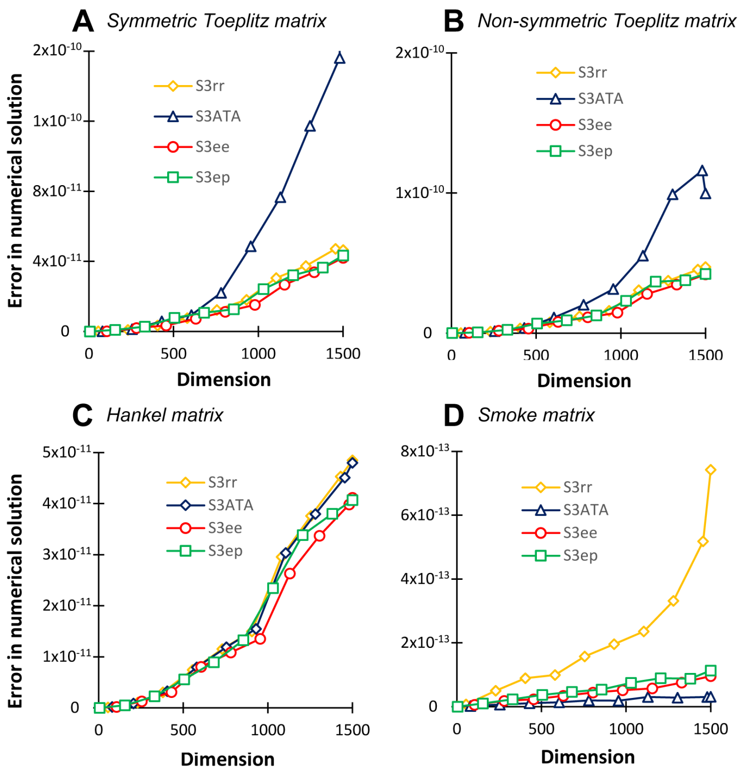


Figure 6. Comparative analysis of the four variants of the orthogonally scaled complex ABS algorithm on selected Matlab Gallery problems.

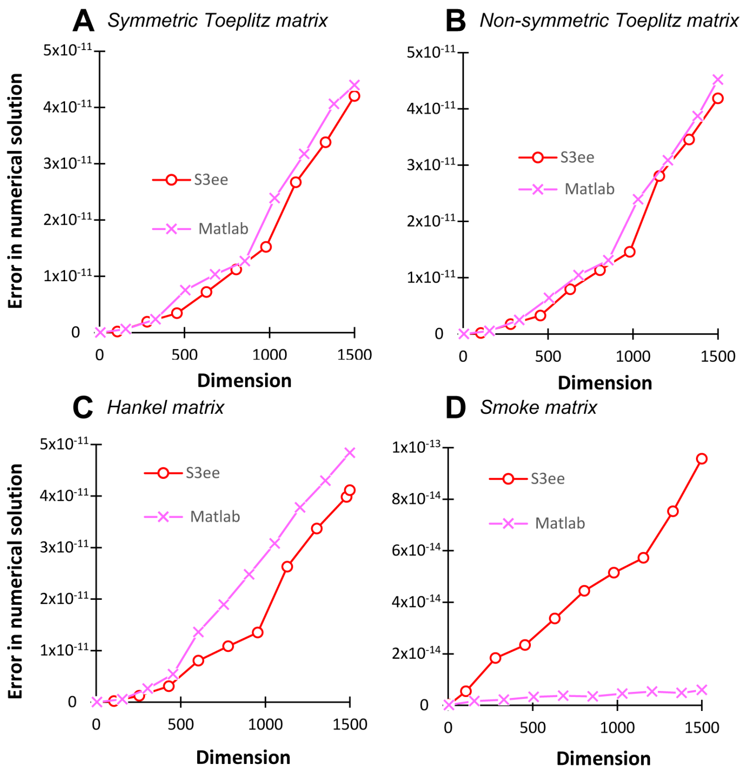
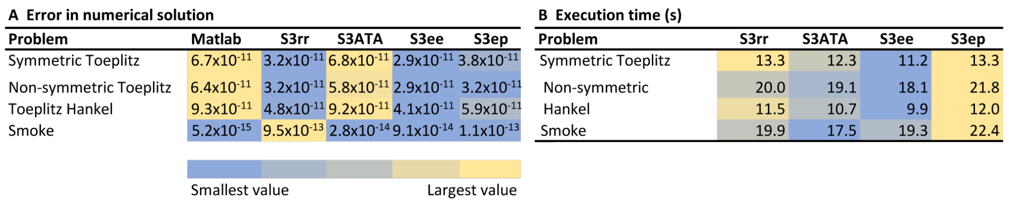


Figure 7. Comparative analysis of the computational accuracy of the ABS-based S3ee algorithm and the Matlab *mldivide* algorithm on selected Matlab gallery problems.



A Error in numerical solution

Problem	Matlab	S3rr	S3ATA	S3ee	S3ep
Symmetric Toeplitz	6.7×10^{-11}	3.2×10^{-11}	6.8×10^{-11}	2.9×10^{-11}	3.8×10^{-11}
Non-symmetric Toeplitz	6.4×10^{-11}	3.2×10^{-11}	5.8×10^{-11}	2.9×10^{-11}	3.2×10^{-11}
Toeplitz Hankel	9.3×10^{-11}	4.8×10^{-11}	9.2×10^{-11}	4.1×10^{-11}	5.9×10^{-11}
Smoke	5.2×10^{-15}	9.5×10^{-13}	2.8×10^{-14}	9.1×10^{-14}	1.1×10^{-13}

B Execution time (s)

Problem	S3rr	S3ATA	S3ee	S3ep
Symmetric Toeplitz	13.3	12.3	11.2	13.3
Non-symmetric	20.0	19.1	18.1	21.8
Hankel	11.5	10.7	9.9	12.0
Smoke	19.9	17.5	19.3	22.4

Figure 8. Numerical properties of the four ABS-based variants and the Matlab algorithm for solving 1500-dimensional Matlab Gallery problems. Panel (**A**) shows the computational accuracy. Panel (**B**) shows the execution times for our ABS-based variants in seconds.

We have also examined the behaviour of our ABS-based variants on large, i.e., more than 5000 dimensions of selected Matlab Gallery problems. Our results of the ABS-based variants are summarized in Figure S3, while the comparison of S3ee and Matlab results are summarized in Figure S4 in Supplementary Materials. For high dimensions, the numerical properties of the algorithms were very similar to those for low dimensions. Overall, of the ABS-based variants, the S3ee variant computed the most accurately and all ABS-based variants gave more accurate results than the Matlab algorithm except for the Smoke problem.

3.3. *Real-Life Problems*

We also wanted to examine how the four variants of our ABS-based algorithm work on real-life problems. Our next experiments focused on the performance of the algorithms on examples from the SuiteSparse matrix collection [37] and from the SWUFE-Math Test Matrices Library [38]. Table 2 summarizes the matrices we used and their key properties.

Table 2. Summary of used real-life problems. Matrices marked by italic font come from the SWUFE-Math Test Matrices Library, while non-italicized matrices come from the SuiteSparse matrix collection.

Matrix Name	n	m	Condition Number	Application Area
qc324	324	324	7.38×10^4	Electromagnetics
young1c	841	841	9.91×10^2	Acoustics
young2c	841	841	9.91×10^2	Duplicate Acoustics
young4c	841	841	5.55×10^2	Acoustics
dwg961b	961	961	3.35×10^7	Electromagnetics
cube1800_test3	1800	1800	4.98×10^1	Electromagnetics
parallelepipede_test2	2016	2016	8.43×10^1	Electromagnetics
sphere2430_test1	2431	2431	4.32×10^1	Electromagnetics
qc2534	2534	2534	5.19×10^5	Electromagnetics
conf5_0-4x4-14	3072	3072	9.17×10^3	Quantum Chemistry
mplate	5962	5962	4.83×10^{16}	Acoustics

The calculation accuracies are outlined in Panel A of Figure 9. It can be stated that each of the variants of the ABS-based algorithm was able to solve the problems with acceptable accuracy. For these problems, the S3rr algorithm performed best. It should be noted, however, that in most cases, the Matlab function calculated the solution most accurately. This may partly be explained by the fact that Matlab uses different solvers for different (i.e., sparse, dense) matrices. In addition to the accuracy of the solutions and execution times, we show the relative 2-norm of the residual vectors in the Panel **C** to ensure that the numerical properties of our ABS-based algorithm can be compared with other published algorithms [38–40] developed to solve linear systems of equations. These comparisons revealed that our method is significantly more accurate than expected for iterative solutions. Furthermore, some iterative methods achieved this accuracy of 10^{-6} only after a relatively large number of iterations. In the case of the young1c problem, several algorithms needed nearly 400 steps to achieve an accuracy of 10^{-6} [38,39], while our algorithm achieved an accuracy of 10^{-14} in about twice as many steps.

A Error in numerical solution

	Problem	Matlab	S3rr	S3ATA	S3ee	S3ep
dense	cube1800_test3	2.1×10^{-13}	5.2×10^{-13}	4.3×10^{-13}	3.3×10^{-13}	3.5×10^{-13}
	parallelepipede_test2	6.6×10^{-14}	1.4×10^{-13}	1.2×10^{-13}	1.2×10^{-13}	1.2×10^{-13}
	sphere2430_test1	1.5×10^{-13}	4.0×10^{-13}	3.6×10^{-13}	2.2×10^{-13}	2.3×10^{-13}
sparse	qc324	3.9×10^{-15}	4.9×10^{-15}	5.9×10^{-15}	1.2×10^{-13}	1.2×10^{-13}
	young1c	1.3×10^{-12}	3.1×10^{-12}	3.8×10^{-12}	5.3×10^{-12}	5.8×10^{-12}
	young2c	1.3×10^{-12}	3.1×10^{-12}	3.8×10^{-12}	5.3×10^{-12}	5.8×10^{-12}
	young4c	1.1×10^{-12}	3.1×10^{-12}	3.9×10^{-12}	4.2×10^{-12}	4.1×10^{-12}
	dwg961b	1.4×10^{-12}	1.0×10^{-11}	1.0×10^{-6}	4.2×10^{-11}	5.4×10^{-10}
	qc2534	2.3×10^{-14}	5.0×10^{-14}	5.9×10^{-14}	1.5×10^{-11}	1.6×10^{-11}
	conf5_0-4x4-14	4.8×10^{-12}	7.0×10^{-12}	3.9×10^{-13}	2.8×10^{-13}	2.9×10^{-13}
	mplate	4.2×10^{-9}	4.1×10^{-7}	3.1×10^{-6}	1.7×10^{-3}	7.4×10^{-3}

B Execution time (s)

	Problem	S3rr	S3ATA	S3ee	S3ep
dense	cube1800_test3	198.7	214.5	187.6	236.1
	parallelepipede_test2	297.0	299.3	295.8	356.9
	sphere2430_test1	476.5	541.4	472.1	557.3
sparse	qc324	1.6	1.0	0.8	1.0
	young1c	14.1	15.8	11.1	16.3
	young2c	17.9	18.5	16.3	20.2
	young4c	18.4	20.1	15.1	19.2
	dwg961b	24.8	27.1	23.1	29.1
	qc2534	451.5	494.9	406.3	509.5
	conf5_0-4x4-14	693.9	788.8	636.8	837.6
	mplate	3870.8	5370.6	4277.1	5573.5

C Relative 2-norm of the residual vector

	Problem	Matlab	S3rr	S3ATA	S3ee	S3ep
dense	cube1800_test3	1.5×10^{-15}	3.2×10^{-15}	3.8×10^{-15}	2.4×10^{-15}	2.5×10^{-15}
	parallelepipede_test2	1.5×10^{-15}	2.8×10^{-15}	3.3×10^{-15}	2.8×10^{-15}	2.8×10^{-15}
	sphere2430_test1	1.8×10^{-15}	4.3×10^{-15}	4.9×10^{-15}	2.6×10^{-15}	2.8×10^{-15}
sparse	qc324	7.1×10^{-16}	1.5×10^{-15}	2.1×10^{-15}	3.0×10^{-14}	2.7×10^{-14}
	young1c	1.8×10^{-16}	2.0×10^{-15}	1.9×10^{-15}	2.7×10^{-15}	2.7×10^{-15}
	young2c	1.8×10^{-16}	2.0×10^{-15}	1.9×10^{-15}	2.7×10^{-15}	2.7×10^{-15}
	young4c	1.7×10^{-16}	2.0×10^{-15}	1.6×10^{-15}	2.2×10^{-15}	2.2×10^{-15}
	dwg961b	2.5×10^{-16}	1.2×10^{-9}	1.2×10^{-14}	5.1×10^{-14}	6.6×10^{-13}
	qc2534	1.2×10^{-15}	2.3×10^{-15}	2.8×10^{-15}	6.9×10^{-13}	6.7×10^{-13}
	conf5_0-4x4-14	3.6×10^{-16}	3.8×10^{-15}	3.9×10^{-14}	2.2×10^{-15}	2.2×10^{-15}
	mplate	4.0×10^{-16}	9.5×10^{-13}	1.3×10^{-13}	5.1×10^{-10}	1.8×10^{-9}

Smallest value Largest value

Figure 9. Comparative analysis of the performance of the four variants of the orthogonally scaled complex ABS algorithm and the *mldivide* (\) algorithm of Matlab on real-life problems. Panel (**A**) shows the computational accuracy. Panel (**B**) shows the execution times for our ABS-based variants in seconds. Panel (**C**) shows the relative 2-norm of the residual vector ($\frac{\|b-Ax_n\|_2}{\|b\|_2}$).

4. Conclusions

In this paper, we presented a new ABS-based algorithm for solving complex linear systems of equations and we proved some of its basic features. We also showed a special choice of parameters of our algorithm, which ensures that the search vectors (p_i) are $A^H A$ conjugate. A detailed numerical analysis of four variants of the ABS-based orthogonally scaled algorithm has also been provided. These variants were tested on randomly generated full and rank deficient, different-density systems of linear equations. Furthermore, the computational accuracy of the algorithm was tested on real-life problems. These numerical experiments showed that the ABS-based algorithm solved the problem with acceptable accuracy in all cases and provided more accurate results than the MATLAB built-in function in most test cases. These numerical results demonstrate the practical usefulness of the algorithm in addition to its theoretical significance.

Additionally, a valuable numerical property of our algorithm is that if we want to compute a complex system of linear equations with several right-hand sides [40], it is not necessary to recompute the matrix H_i. Instead, it is sufficient to store the vectors p_i and recompute the updates x_i, which can significantly reduce the computational cost. and thus our algorithm can be used effectively to solve such problems.

Supplementary Materials: The following are available online at https://www.mdpi.com/article/10.3390/math9192527/s1.

Author Contributions: The authors contributed equally to this work. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The Matlab code for the two variants of the complex ABS algorithms (S3ee, Serr) is available on GitHub, at https://github.com/SzabinaFodor/complexABS (accessed: 2 October 2021).

Acknowledgments: The authors would like to express their sincere thanks to Attila Mócsai and the anonymous reviewers for their valuable comments and constructive suggestions that greatly improved the presentation of our work.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

A^T	transpose of the matrix A
$\overline{A}$	the conjugate of the matrix A
$A^H = \overline{A}^T$	conjugate transpose of the matrix A

References

1. Wong, S.S.M. *Computational Methods in Physics and Engineering*; World Scientific Publishing Co Pte Ltd.: Hackensack, NJ, USA, 1992.
2. Metzler, L.A. Taxes and subsidies in Leontief's input-output model. *Q. J. Econ.* **1951**, *65*, 433–438. [CrossRef]
3. Bar-On, I.; Ryaboy, V. Fast diagonalization of large and dense complex symmetric matrices, with applications to quantum reaction dynamics. *SIAM J. Sci. Comput.* **1997**, *18*, 1412–1435. [CrossRef]
4. Nesemann, J. *PT-Symmetric Schrödinger Operators with Unbounded Potentials*; Springer: Berlin/Heidelberg, Germany, 2011.
5. Lancaster, P. Inverse spectral problems for semisimple damped vibrating systems. *SIAM J. Matrix Anal. Appl.* **2007**, *29*, 279–301. [CrossRef]
6. Keller, J.B.; Givoli, D. Exact non-reflecting boundary conditions. *J. Comput. Phys.* **1989**, *82*, 172–192. [CrossRef]
7. Van Dijk, W.; Toyama, F. Accurate numerical solutions of the time-dependent Schrödinger equation. *Phys. Rev. E* **2007**, *75*, 036707. [CrossRef]
8. Benia, Y.; Ruggieri, M.; Scapellato, A. Exact solutions for a modified Schrödinger equation. *Mathematics* **2019**, *7*, 908. [CrossRef]
9. Obaidat, S.; Mesloub, S. A New Explicit Four-Step Symmetric Method for Solving Schrödinger's Equation. *Mathematics* **2019**, *7*, 1124. [CrossRef]
10. Biddlecombe, C.; Heighway, E.; Simkin, J.; Trowbridge, C. Methods for eddy current computation in three dimensions. *IEEE Trans. Magn.* **1982**, *18*, 492–497. [CrossRef]
11. Arridge, S.R. Optical tomography in medical imaging. *Inverse Probl.* **1999**, *15*, R41. [CrossRef]
12. Benzi, M.; Bertaccini, D. Block preconditioning of real-valued iterative algorithms for complex linear systems. *IMA J. Numer. Anal.* **2008**, *28*, 598–618. [CrossRef]
13. Day, D.; Heroux, M.A. Solving complex-valued linear systems via equivalent real formulations. *SIAM J. Sci. Comput.* **2001**, *23*, 480–498. [CrossRef]
14. Gu, X.M.; Clemens, M.; Huang, T.Z.; Li, L. The SCBiCG class of algorithms for complex symmetric linear systems with applications in several electromagnetic model problems. *Comput. Phys. Commun.* **2015**, *191*, 52–64. [CrossRef]
15. Wang, J.; Guo, X.P.; Zhong, H.X. Accelerated GPMHSS method for solving complex systems of linear equations. *East Asian J. Appl. Math.* **2017**, *7*, 143–155. [CrossRef]
16. Li, L.; Huang, T.Z.; Ren, Z.G. A preconditioned COCG method for solving complex symmetric linear systems arising from scattering problems. *J. Electromagn. Waves Appl.* **2008**, *22*, 2023–2034. [CrossRef]
17. Gu, X.M.; Huang, T.Z.; Li, L.; Li, H.B.; Sogabe, T.; Clemens, M. Quasi-minimal residual variants of the COCG and COCR methods for complex symmetric linear systems in electromagnetic simulations. *IEEE Trans. Microw. Theory Tech.* **2014**, *62*, 2859–2867. [CrossRef]
18. Jacobs, D.A. A generalization of the conjugate-gradient method to solve complex systems. *IMA J. Numer. Anal.* **1986**, *6*, 447–452. [CrossRef]
19. Saad, Y. *Iterative Methods for Sparse Linear Systems*; SIAM: Philadelphia, PA, USA, 2003.
20. Fischer, B.; Reichel, L. A stable Richardson iteration method for complex linear systems. *Numer. Math.* **1989**, *54*, 225–242. [CrossRef]
21. Bai, Z.Z.; Benzi, M.; Chen, F. Modified HSS iteration methods for a class of complex symmetric linear systems. *Computing* **2010**, *87*, 93–111. [CrossRef]
22. Li, X.; Yang, A.L.; Wu, Y.J. Lopsided PMHSS iteration method for a class of complex symmetric linear systems. *Numer. Algorithms* **2014**, *66*, 555–568. [CrossRef]
23. Puzyrev, V.; Koric, S.; Wilkin, S. Evaluation of parallel direct sparse linear solvers in electromagnetic geophysical problems. *Comput. Geosci.* **2016**, *89*, 79–87. [CrossRef]
24. Koric, S.; Lu, Q.; Guleryuz, E. Evaluation of massively parallel linear sparse solvers on unstructured finite element meshes. *Comput. Struct.* **2014**, *141*, 19–25. [CrossRef]
25. Abaffy, J.; Spedicato, E. *ABS Projection Algorithms: Mathematical Techniques for Linear and Nonlinear Equations*; Prentice-Hall, Inc.: Chichester, UK, 1989.
26. Spedicato, E.; Xia, Z.; Zhang, L. ABS algorithms for linear equations and optimization. *J. Comput. Appl. Math.* **2000**, *124*, 155–170. [CrossRef]

27. Fodor, S. Symmetric and non-symmetric ABS methods for solving Diophantine systems of equations. *Ann. Oper. Res.* **2001**, *103*, 291–314. [CrossRef]
28. Abaffy, J.; Fodor, S. Solving Integer and Mixed Integer Linear Problems with ABS Method. *Acta Polytech. Hung.* **2013**, *10*, 81–98. [CrossRef]
29. Galántai, A. Parallel ABS projection methods for linear and nonlinear systems with block arrowhead structure. *Comput. Math. Appl.* **1999**, *38*, 11–17. [CrossRef]
30. Fodor, S.; Németh, Z. Numerical analysis of parallel implementation of the reorthogonalized ABS methods. *Cent. Eur. J. Oper. Res.* **2019**, *27*, 437–454. [CrossRef]
31. Abaffy, J.; Fodor, S. Reorthogonalization methods in ABS classes. *Acta Polytech. Hung.* **2015**, *12*, 23–41.
32. Broyden, C. On the numerical stability of Huang's and related methods. *J. Optim. Theory Appl.* **1985**, *47*, 401–412. [CrossRef]
33. Hegedüs, C.J. Reorthogonalization Methods Revisited. *Acta Polytech. Hung.* **2015**, *12*, 7–26.
34. Parlett, B. *The Symmetric Eigenvalue Problem*; Republished amended version of original published by Prentice-Hall; SIAM: Philadelphia, PA, USA; Prentice-Hall: Englewood Cliffs, NJ, USA, 1980; p. 1980.
35. Attaway, D.C. *Matlab: A Practical Introduction to Programming and Problem Solving*; Butterworth-Heinemann Elsevier Ltd.: Oxford, UK, 2013.
36. Golub, G.H.; Van Loan, C.F. *Matrix Computations*, 4th ed.; Johns Hopkins University Press: Baltimore, MD, USA, 2012; Volume 3.
37. Davis, T.A.; Hu, Y. The University of Florida sparse matrix collection. *ACM Trans. Math. Softw. (TOMS)* **2011**, *38*, 1–25. [CrossRef]
38. Gu, X.M.; Carpentieri, B.; Huang, T.Z.; Meng, J. Block variants of the COCG and COCR methods for solving complex symmetric linear systems with multiple right-hand sides. In *Numerical Mathematics and Advanced Applications ENUMATH 2015*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 305–313.
39. Jing, Y.F.; Huang, T.Z.; Zhang, Y.; Li, L.; Cheng, G.H.; Ren, Z.G.; Duan, Y.; Sogabe, T.; Carpentieri, B. Lanczos-type variants of the COCR method for complex nonsymmetric linear systems. *J. Comput. Phys.* **2009**, *228*, 6376–6394. [CrossRef]
40. Zhong, H.X.; Gu, X.M.; Zhang, S.L. A Breakdown-Free Block COCG Method for Complex Symmetric Linear Systems with Multiple Right-Hand Sides. *Symmetry* **2019**, *11*, 1302. [CrossRef]