mathematics

Article

On C-To-R-Based Iteration Methods for a Class of Complex Symmetric Weakly Nonlinear Equations

Min-Li Zeng [1,*] and Guo-Feng Zhang [2]

1 School of Mathematics and Finance, Putian University, Putian 351100, China
2 School of Mathematics and Statistic, Lanzhou University, Lanzhou 730000, China; gf_zhang@lzu.edu.cn
* Correspondence: zengml12@lzu.edu.cn

Received: 31 December 2019; Accepted: 4 February 2020; Published: 6 February 2020

Abstract: To avoid solving the complex systems, we first rewrite the complex-valued nonlinear system to real-valued form (C-to-R) equivalently. Then, based on separable property of the linear and the nonlinear terms, we present a C-to-R-based Picard iteration method and a nonlinear C-to-R-based splitting (NC-to-R) iteration method for solving a class of large sparse and complex symmetric weakly nonlinear equations. At each inner process iterative step of the new methods, one only needs to solve the real subsystems with the same symmetric positive and definite coefficient matrix. Therefore, the computational workloads and computational storage will be saved in actual implements. The conditions for guaranteeing the local convergence are studied in detail. The quasi-optimal parameters are also proposed for both the C-to-R-based Picard iteration method and the NC-to-R iteration method. Numerical experiments are performed to show the efficiency of the new methods.

Keywords: weakly nonlinear equations; C-to-R preconditioner; local convergence; Picard iteration; complex symmetric matrix.

MSC: 65F10; 65F50; 65N22

1. Introduction

We consider the iterative solutions of nonlinear system of equations in the following form,

$$Au = \phi(u), \quad \text{or} \quad F(u) = Au - \phi(u) = 0, \tag{1}$$

where $A = W + \mathbf{i}T \in \mathbb{C}^{n\times n}$ is a large, sparse, complex symmetric matrix, with $W \in \mathbb{R}^{n\times n}$ and $T \in \mathbb{R}^{n\times n}$ being the real parts and the imaginary parts of the coefficient matrix A, respectively. Here, we assume that W and T are both symmetric positive and semidefinite (SPSD) and at least one of them being symmetric positive and definite (SPD). The right hand vector function $\phi : \mathbb{D} \subset \mathbb{C}^n \to \mathbb{C}^n$ is a continuously differential function defined on the open convex domain $\mathbb{D}$ in the n-dimensional $\mathbb{C}^n$. $u \in \mathbb{C}^n$ is an unknown vector. When the linear term Au is strongly dominant over the nonlinear term $\phi(u)$ in certain norm [1], we say that the system of nonlinear Equation (1) is weakly nonlinear. Here, and in the sequence, we assume that the Jacobian matrix of the nonlinear function $\phi(u)$ at the solution point $u^\star \in \mathbb{D}$, denoted as $\phi'(u^\star)$, is the non-Hermitian and negative semidefinite.

Weakly nonlinear equations of the form (1) arise in many areas of scientific computing and engineering applications, e.g., nonlinear ordinary and partial differential equations, nonlinear integral and integro differential equations, nonlinear optimization and variational problems, saddle point problems from image processing, and so on. For more details, see [2–8].

By substituting $u = x + \mathbf{i}y$, where $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^n$, we can rewrite the system of nonlinear Equations (1) as

$$(Wx - Ty) + \mathbf{i}(Tx + Wy) = R(u) + \mathbf{i}I(u),$$

where $R(u) = \text{real}(\phi(u)) \in \mathbb{R}^n$ and $I(u) = \text{imag}(\phi(u)) \in \mathbb{R}^n$ are the real parts and imaginary parts of $\phi(u)$, respectively. Here, we reformulate the complex nonlinear system (1) as a block two-by-two real form in the following,

$$C\begin{pmatrix} x \\ y \end{pmatrix} := \begin{pmatrix} W & -T \\ T & W \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} R(u) \\ I(u) \end{pmatrix} \triangleq \Phi(u). \tag{2}$$

When $R(u)$ and $I(u)$ are both constant vectors, the system (2) reduces to a linear system with structural block two-by-two coefficient matrix. To solve the linear system (2) efficiently, many methods have been proposed. Particularly, when the block matrices W and T are both SPSD and at least one of them is SPD, some classical iterative methods can be found in existed references, for example, the block preconditioned methods [9,10], the additive block diagonal preconditioned method [11]. There are also sequences of preconditioners based on the modified Hermitian and skew-Hermitian splitting (MHSS), such as, the preconditioned MHSS (PMHSS) iteration method [12,13]. Some other methods such as the preconditioned GSOR (PGSOR) iteration method [14], the complex-value to real-value (C-to-R) preconditioner [15], and so on, also attract a lot of researchers' interest. For more efficient methods, we refer to the works in [16–18].

When Au is strong dominant than the nonlinear term $\phi(u)$, and $R(u)$ and $I(u)$ are dependent on the variable vector u, then we say the system (2) is weakly nonlinear. The most classic and important solvers for the system of nonlinear Equations (1) is the Newton method [3,6,19], which can be described as

$$u^{(k+1)} = u^{(k)} - F'(u^{(k)})^{-1}F(u^{(k)}), \quad k = 0, 1, 2, \cdots.$$

It can be seen from the Newton method that the dominant task in implementations is to solve the following equation at each iteration step,

$$F'(u^{(k)})s^{(k)} = -F(u^{(k)}), \quad \text{with} \quad u^{(k+1)} = u^{(k)} + s^{(k)},$$

and to recompute the Jacobian matrix $F'(u^{(k)})$ at every iteration step. When the Jacobian matrix $F'(u^{(k)})$ is large and sparse, we usually use either the splitting relaxation form [6] or the Krylov subspace method form [5,20,21] to compute an approximation to update the vector $s^{(k)}$. However, those methods are all heavy in both computational workload and computational storage.

To improve the efficiency of the Newton iteration method, Bai and Guo [4] use the Hermitian and skew-Hermitian splitting (HSS) method to solve approximately the Newton Equations (2), called the Newton-HSS method and then Guo and Duff [22] analyze the Kantorovich-type semilocal convergence. Many efficient methods based on the Hermitian and skew-Hermitian splitting (HSS) have come forth since then. For example, Bai and Yang [1] present the nonlinear HSS-like iteration method based on the Hermitian and skew-Hermitian (HS) splitting of the non-Hermitian coefficient matrix of the linear term Au. Some variants of the HSS-based methods for nonlinear equations can be found in references, e.g., the lopsided preconditioned modified HSS (LPMHSS) iteration method [23], the Newton-MHSS method [24], the accelerated Newton-GPSS iteration method [25], the preconditioned modified Newton-MHSS method [26], the modified Newton-SHSS method [27], the modified Newton-DPMHSS method [28], and so on. See [4,29–32] for more details.

In this paper, we will concentrate on the efficient methods of the real value equivalent nonlinear system (2). By utilizing the C-to-R preconditioning technique proposed in [16], we will first construct a C-to-R-based Picard iteration method. This method is actually the inexact Picard method with the C-to-R iterative method as inner process iteration. Therefore, the convergence results based on Picard iteration method can be used directly. To further improve the efficiency, we then introduce a nonlinear

C-to-R splitting iteration method for solving the weakly nonlinear equations. The local convergence for both of the two new methods are analyzed in detail. The way to determine the theoretical optimal parameters is also studied.

The organization of this paper is outlined as follows. In Section 2, we firstly give the C-to-R-based Picard iteration method. Then, we give the convergence results and the choice of the quasi-optimal parameters. In Section 3, we construct the nonlinear C-to-R-based splitting iteration method for solving the nonlinear system (2) and then we give a detailed theoretically analysis about the convergence properties. Theoretical optimal parameters are also proposed subsequently. Numerical experiments are given in Section 4 to illustrate the feasibility and effectiveness of the new methods. Finally, a brief conclusion and some remarks are drawn in Section 5 to end this work.

Throughout this paper, we use $\rho(B)$ to denote the spectral radius of the matrix B. Denote $u = x + \mathbf{i}y \in \mathbb{C}^n$ with $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^n$ being its real parts and imaginary parts, respectively.

2. The C-To-R-Based Picard Iteration Method

In this section, we will propose the C-to-R-based Picard iteration method. To begin with, according to [16], we review the C-to-R preconditioner for the block two-by-two coefficient matrix in Equation (2) as

$$B(\alpha) = \begin{pmatrix} \alpha^2 W + 2\alpha T & -T \\ T & W \end{pmatrix}, \tag{3}$$

where α is a positive constant. We know that the implementing of the preconditioner $B(\alpha)$ at each iterative step needs one to solve the generalized residual linear equations

$$B(\alpha)\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix}.$$

Or equivalently,

$$\begin{cases} \alpha(\alpha W + T)x + T(\alpha x - y) = f, \\ T(\alpha x - y) + (\alpha W + T)y = \alpha g, \end{cases}$$

i.e.,

$$\begin{cases} (\alpha W + T)(\alpha x - y) = f - \alpha g, \\ \alpha(\alpha W + T)x = f - T(\alpha x - y). \end{cases}$$

Therefore, we can summarize the implementation as the algorithm 1 for solving the above equations.

Algorithm 1 (The C-to-R iteration method)**.** *Let α be a given positive constant. Use the following steps to solve the generalized residual equation.*

```
Step 1.  solve $(\alpha W + T)z = f - \alpha g$ to obtain $z$.
Step 2.  solve $(\alpha W + T)x = \frac{1}{\alpha}(f - Tz)$ to obtain $x$.
Step 3.  compute $y = \alpha x - z$.
```

The preconditioner $B(\alpha)$ can be seen as a splitting matrix from the following matrix splitting,

$$\begin{pmatrix} W & -T \\ T & W \end{pmatrix} = \begin{pmatrix} \alpha^2 W + 2\alpha T & -T \\ T & W \end{pmatrix} - \begin{pmatrix} (\alpha^2 - 1)W + 2\alpha T & 0 \\ 0 & 0 \end{pmatrix}. \tag{4}$$

On the other hand, because Au is strong dominant over the nonlinear term $\phi(u)$, then the Picard iteration method can be used based on the separable property of the linear term and nonlinear term, i.e.,

$$Au^{(k+1)} = \phi(u^{(k)}). \tag{5}$$

Or equivalently, the iteration of the real value form can be described as

$$C\begin{pmatrix} x^{(k+1)} \\ y^{(k+1)} \end{pmatrix} := \begin{pmatrix} W & -T \\ T & W \end{pmatrix}\begin{pmatrix} x^{(k+1)} \\ y^{(k+1)} \end{pmatrix} = \begin{pmatrix} R(u^{(k)}) \\ I(u^{(k)}) \end{pmatrix} = \Phi(u^{(k)}), \tag{6}$$

where $u^{(k)} = x^{(k)} + \mathbf{i}y^{(k)} \in \mathbb{C}^n$ with $x^{(k)}, y^{(k)} \in \mathbb{R}^n$. Therefore, to improve the efficiency, we can use the splitting iterative method as inner process iteration based on (6) to obtain the solution approximately at each Picard iteration process. We describe the C-to-R-based Picard iteration method in the Algorithm 2.

Algorithm 2 (The C-to-R-based Picard iteration method)**.** *Let $\phi \subset \mathbb{C}^n \to \mathbb{C}^n$ be a continuously differentiable function and $A = W + \boldsymbol{i}T \in \mathbb{C}^{n\times n}$ be a large, sparse complex symmetric matrix, where $W \in \mathbb{R}^{n\times n}$ and $T \in \mathbb{R}^{n\times n}$ are the real parts and the imaginary parts of A, respectively. Given an initial guess $u^{(0)} \in \mathbb{C}^n$. For $k = 0, 1, 2, \cdots$, until $\{u^{(k)}\}$ converges, compute the next iterate $\{u^{(k+1)}\}$ according to the following steps.*

Step 1. *set $\tilde{u}^{(k,0)} := u^{(k)}$.*

Step 2. *for a given positive constant α and $l = 0, 1, 2, \ldots, l_k - 1$, set $x^{(k,l)} = real(\tilde{u}^{(k,l)})$, $y^{(k,l)} = imag(\tilde{u}^{(k,l)})$, then solve the following subsystem*

$$\begin{pmatrix} \alpha^2 W + 2\alpha T & -T \\ T & W \end{pmatrix}\begin{pmatrix} x^{(k,l+1)} \\ y^{(k,l+1)} \end{pmatrix} = \begin{pmatrix} (\alpha^2-1)W + 2\alpha T & 0 \\ 0 & 0 \end{pmatrix}\begin{pmatrix} x^{(k,l)} \\ y^{(k,l)} \end{pmatrix} + \begin{pmatrix} R(u^{(k,l)}) \\ I(u^{(k,l)}) \end{pmatrix}$$

to obtain $x^{(k,l+1)}$ and $y^{(k,l+1)}$. Set $\tilde{u}^{(k,l+1)} = x^{(k,l+1)} + \boldsymbol{i}y^{(k,l+1)}$.

Step 3. *set $u^{(k+1)} = \tilde{u}^{(k,l_k)}$.*

We can also describe the Algorithm 2 in a detailed implementing steps as the Algorithm 3.

Algorithm 3 (The detailed implementing process of the C-to-R-based Picard iteration method)**.** *Given an initial guess $u^{(0)} \in \mathbb{C}^n$ and a sequence positive integers $\{l_k\}_{k=0}^{\infty}$, use the following iteration steps to compute $u^{(k+1)}$ for $k = 0, 1, 2, \ldots$, until $u^{(k)}$ satisfies the stopping criterion:*

```
Step 1. set $r^{(k)} = \phi(u^{(k)}) - Au^{(k)}$.
Step 2. given an initial guess $\tilde{z}^{(k,0)} \in \mathbb{C}^n$. For $l = 0, 1, 2, \ldots, l_k - 1$,
Step 3. compute $\tilde{r}^{(k,l)} = r^{(k)} - A\tilde{z}^{(k,l)}$, set $\tilde{f}^{(k,l)} = real(\tilde{r}^{(k,l)})$, $\tilde{g}^{(k,l)} = imag(\tilde{r}^{(k,l)})$.
Step 4. solve $(\alpha W + T)z^{(k,l)} = \tilde{f}^{(k,l)} - \alpha\tilde{g}^{(k,l)}$ to obtain $z^{(k,l)}$.
Step 5. solve $(\alpha W + T)x^{(k,l)} = \frac{1}{\alpha}(\tilde{f}^{(k,l)} - T\tilde{z}^{(k,l)})$ to obtain $x^{(k,l)}$.
Step 6. compute $y^{(k,l)} = \alpha x^{(k,l)} - z^{(k,l)}$.
Step 7. set $\tilde{z}^{(k,l+1)} = \tilde{z}^{(k,l)} + x^{(k,l)} + \boldsymbol{i}y^{(k,l)}$.
        If $\tilde{z}^{(k,l+1)}$ satisfies the inner stopping criterion, go to Step 8.
        If $\tilde{z}^{(k,l+1)}$ does not meet the inner stopping criterion, return to Step 3.
Step 8. set $u^{(k+1)} = u^{(k)} + \tilde{z}^{(k,l_k)}$.
```

From Algorithm 3, we find that the main workload is to solve the linear subsystem with the coefficient matrix being $\alpha W + T$ both in Step 4 and Step 5. Therefore, we can solve the corresponding system exactly by sparse Cholesky decomposition or inexactly by symmetric Krylov subspace methods (e.g., the conjugate gradient method).

Next, we denote

$$L(\alpha) = \begin{pmatrix} \alpha^2 W + 2\alpha T & -T \\ T & W \end{pmatrix}^{-1}\begin{pmatrix} (\alpha^2-1)W + 2\alpha T & 0 \\ 0 & 0 \end{pmatrix}, \tag{7}$$

then the local convergence results for the C-to-R-based Picard iteration method can be summarized in the following theorem.

Theorem 1. *Denoted by*

$$\theta(\alpha) = \|L(\alpha)\|, \quad \mu = \|A^{-1}\| \quad \text{and} \quad \nu = \|A^{-1}\phi'(u^\star)\|.$$

Then, for any initial guess $u^{(0)} \in \mathbb{C}^n$ and any sequence of positive integers l_k, $k = 0, 1, 2, \ldots$, if $\nu < 1$ and $l_0 \geq \lfloor \frac{\ln(\frac{1-\nu}{1+\nu})}{\ln \theta(\alpha)} \rfloor$, the iteration sequence generated by the C-to-R-based Picard method converges to the exact solution $u^\star$ and it holds

$$\limsup_{k\to\infty} \|u^{(k)} - u^\star\|^{\frac{1}{k}} \leq \nu + (1+\nu)\theta(\alpha)^{l^*},$$

where $l^* = \inf\{\lim_{k\to\infty} l_k\}$. Particularly, let $\lim_{k\to\infty} l_k$, it follows

$$\limsup_{k\to\infty} \|u^{(k)} - u^\star\|^{\frac{1}{k}} \leq \nu, \tag{8}$$

or equivalently, the convergence rate of the C-to-R-based Picard method is R-rate, with the R-factor being ν.

Proof. The results can be obtained immediately from the results in [1]. □

Theorem 1 shows that the convergence rate of the C-to-R-based Picard iteration method depends on the quantities of $\theta(\alpha)$ and ν. The weakly nonlinear property leads to the dominant of the quantities of $\theta(\alpha)$. Therefore, we can obtain the quasi-optimal parameter by minimizing the R-factor $\theta(\alpha) = \|L(\alpha)\|$. In other words, we need to find α such that the eigenvalues of the following matrix cluster around 1.

$$\begin{pmatrix} \alpha^2 W + 2\alpha T & -T \\ T & W \end{pmatrix}^{-1} \begin{pmatrix} W & -T \\ T & W \end{pmatrix}.$$

Therefore, we can use the results in [33,34] to obtain the quasi-optimal parameter α in the following theorem.

Theorem 2. *Let α be a given positive constant and assume that the conditions of Theorem 1 are satisfied. Then, the optimal parameter α that minimizes $\theta(\alpha)$ is $\alpha = \frac{\sqrt[4]{8}}{2}$.*

Remark 1. *Because there exists an extra term $\phi'(u^\star)$, then we use the above optimal parameter as a quasi-optimal parameter. In actual implementation, we use this value as a suggestion. The true optimal parameter may vary dependent on the weakly nonlinear term.*

3. The Nonlinear C-to-R-Based Splitting Iteration Method

In this section, we will further introduce the NC-to-R method for solving the block two-by-two nonlinear Equations (2) by making use of the nonlinear fixed-point equation

$$\begin{pmatrix} \alpha^2 W + 2\alpha T & -T \\ T & W \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} (\alpha^2 - 1)W + 2\alpha T & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} R(u) \\ I(u) \end{pmatrix}. \tag{9}$$

The NC-to-R method can be described as the algorithm 4.

Algorithm 4 (The nonlinear C-to-R splitting iteration method). *Let $\phi \subset \mathbb{C}^n \to \mathbb{C}^n$ be a continuously differentiable function and $A = W + \boldsymbol{i}T \in \mathbb{C}^{n\times n}$ be a large, sparse complex matrix, where $W \in \mathbb{R}^{n\times n}$ and $T \in \mathbb{R}^{n\times n}$ are the real parts and the imaginary parts of A, respectively. Given an initial guess $u^{(0)} \in \mathbb{C}^n$.*

For $k = 0, 1, 2, \cdots$, until $\{u^{(k)}\}$ converges, compute the next iterate $\{u^{(k+1)}\}$ by solving the following subsystems.

$$\begin{pmatrix} \alpha^2 W + 2\alpha T & -T \\ T & W \end{pmatrix} \begin{pmatrix} x^{(k+1)} \\ y^{(k+1)} \end{pmatrix} = \begin{pmatrix} (\alpha^2 - 1)W + 2\alpha T & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x^{(k)} \\ y^{(k)} \end{pmatrix} + \begin{pmatrix} R(u^{(k)}) \\ I(u^{(k)}) \end{pmatrix}$$

to obtain $u^{(k+1)} = x^{(k+1)} + \boldsymbol{i} y^{(k+1)}$.

The detailed implementing process of the NC-to-R method can be carried out in the Algorithm 5.

Algorithm 5 (The detailed implementing process of the NC-to-R method)**.** *Given an initial guess $u^{(0)} \in \mathbb{C}^n$. For $k = 0, 1, 2, \cdots$, until $\{u^{(k)}\}$ converges, compute the next iterate $\{u^{(k+1)}\}$ according to the following steps.*

```
Step 1. set $x^{(k)} = real(u^{(k)})$, $y^{(k)} = imag(u^{(k)})$ and compute $\tilde{R}(u^{(k)}) = R(u^{(k)}) + (\alpha^2 - 1)Wx^{(k)} + 2\alpha T x^{(k)}$.
Step 2. solve $(\alpha W + T)z = \tilde{R}(u^{(k)}) - \alpha I(x^{(k)})$ to obtain $z$.
Step 3. solve $(\alpha W + T)\tilde{z}_1 = \frac{1}{\alpha}(\tilde{R}(u^{(k)}) - Tz)$ to obtain $\tilde{z}_1$.
Step 4. compute $\tilde{z}_2 = \alpha \tilde{z}_1 - z$.
Step 5. set $u^{(k+1)} = \tilde{z}_1 + \boldsymbol{i}\tilde{z}_2$.
```

Next, we will focus on the convergence analysis for the NC-to-R method. By utilizing the Ostrowski Theorem (Theorem 10.1.3 in [6]), we can establish the local convergence theory for the NC-to-R method in the following theorem.

Theorem 3. *Assume $\phi : \mathbb{D} \subset \mathbb{C}^n \to \mathbb{C}^n$ is F-differentiable at a point $u^\star \in \mathbb{D}$ such that $Au^\star = \phi(u^\star)$. Denoted by*

$$\widetilde{T}(\alpha; u^\star) = \begin{pmatrix} \alpha^2 W + 2\alpha T & -T \\ T & W \end{pmatrix}^{-1} \left[\begin{pmatrix} (\alpha^2 - 1)W + 2\alpha T & 0 \\ 0 & 0 \end{pmatrix} + \Phi'(u^\star) \right],$$

where

$$\Phi(u^\star) = \begin{pmatrix} R(u^\star) \\ I(u^\star) \end{pmatrix}$$

is defined in (2). If $\rho(\widetilde{T}(\alpha; u^\star)) < 1$, then $u^\star \in \mathbb{D}$ is a point of attraction of the NC-to-R iteration method.

Proof. By making use of (9), we can rewrite the nonlinear system (1) as

$$w = \Psi(w),$$

where

$$\Psi(w) = \begin{pmatrix} \alpha^2 W + 2\alpha T & -T \\ T & W \end{pmatrix}^{-1} \left[\begin{pmatrix} (\alpha^2 - 1)W + 2\alpha T & 0 \\ 0 & 0 \end{pmatrix} w + \Phi(w) \right],$$

$$\Phi(w) = \begin{pmatrix} R(u) \\ I(u) \end{pmatrix} \quad \text{and} \quad w = \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \mathrm{real}(u) \\ \mathrm{imag}(u) \end{pmatrix}.$$

Then, the NC-to-R method can be expressed as

$$w^{(k+1)} = \Psi(w^{(k)}), \quad k = 0, 1, 2, \ldots.$$

After a few simple algebra computations, we have

$$\Psi'(u^\star) = \begin{pmatrix} \alpha^2 W + 2\alpha T & -T \\ T & W \end{pmatrix}^{-1} \left[\begin{pmatrix} (\alpha^2-1)W + 2\alpha T & 0 \\ 0 & 0 \end{pmatrix} + \Phi'(u^\star) \right] = \widetilde{T}(\alpha; u^\star).$$

As $\phi : \mathbb{D} \subset \mathbb{C}^n \to \mathbb{C}^n$ is F-differentiable at a point $u^\star \in \mathbb{D}$, then the real parts and the imaginary parts are also F-differentiable at the point $u^\star$. Therefore, Φ and Ψ are F-differentiable at the point $u^\star$ [35]. Therefore, by making use of the Ostrowski Theorem (Theorem 10.1.3 in [6]), we can conclude that if $\rho(\widetilde{T}(\alpha; u^\star)) < 1$, then $u^\star$ is a point of attraction of the NC-to-R method. □

At the end of this section, we will give a strategy to determine the optimal iterative parameter α by following the some strategy proposed in [36].

We use tr(.) to denote the trace of a matrix in the following. First, we do a partition of the matrix $\Phi'(u^\star)$ as

$$\Phi'(u^\star) = \begin{pmatrix} D_{11}(u^\star) & D_{12}(u^\star) \\ D_{21}(u^\star) & D_{22}(u^\star) \end{pmatrix},$$

then the conjugate transpose of $\Phi'(u^\star)$ can be expressed as

$$(\Phi'(u^\star))^* = \begin{pmatrix} (D_{11}(u^\star))^* & (D_{21}(u^\star))^* \\ (D_{12}(u^\star))^* & (D_{22}(u^\star))^* \end{pmatrix},$$

where $D_{11}(.)$, $D_{12}(.)$, $D_{21}(.)$, and $D_{22}(.)$ have the same size as the matrix W and T.

The following theorem gives a strategy to choose the optimal parameter for the NC-to-R method.

Theorem 4. *Assume that the conditions of Theorem 3 are satisfied. Denoted by*

$$\begin{aligned} \delta &= tr(\Phi'(u^\star)(\Phi'(u^\star))^*) \\ &= tr(D_{11}(u^\star)(D_{11}(u^\star))^*) + tr(D_{12}(u^\star)(D_{12}(u^\star))^*) + tr(D_{21}(u^\star)(D_{21}(u^\star))^*) + tr(D_{22}(u^\star)(D_{22}(u^\star))^*), \end{aligned}$$

$$\eta = tr(WD_{11}(u^\star)) + tr((D_{11}(u^\star)^* W), \quad \xi = tr(TD_{11}(u^\star)) + tr((D_{11}(u^\star)^* T),$$

$$a = tr(W^2), \quad b = tr(WT) + tr(TW), \quad and \quad c = tr(T^2).$$

Then, the optimal parameter α_{opt} for the NC-to-R method satisfies

$$h'(\alpha_{opt}) = 0 \quad and \quad h''(\alpha_{opt}) > 0,$$

where

$$h(\alpha) = a\alpha^4 + 2b\alpha^3 + (-2a + 4c + \eta)\alpha^2 + 2(\xi - b)\alpha + (a + \delta - \eta).$$

Here, $h'(\alpha)$ and $h''(\alpha)$ are the first derivative and second derivative of $h(\alpha)$ with respect to the variable α, respectively.

Proof. As it is known, if α is such a value such that $B(\alpha)$ is close to C, then $B(\alpha)^{-1}(C - B(\alpha))$ should be zero approximately. Therefore, $C - B(\alpha)$ should be approaching to zeros. Furthermore, the square norm $\|C - B(\alpha) + \Phi'(u^\star)\|_F^2$ could be reach the minimal with respect to α.

By direct algebra computations, it follows

$$
\begin{aligned}
&\|C-B(\alpha)+\Phi'(u^\star)\|_F^2 \\
&\quad = \mathrm{tr}\left(\left[\begin{pmatrix}(\alpha^2-1)W+2\alpha T & 0\\ 0 & 0\end{pmatrix}+\Phi'(u^\star)\right]\left[\begin{pmatrix}(\alpha^2-1)W+2\alpha T & 0\\ 0 & 0\end{pmatrix}+\Phi'(u^\star)\right]^*\right) \\
&\quad = \mathrm{tr}\left(\begin{pmatrix}(\alpha^2-1)W+2\alpha T & 0\\ 0 & 0\end{pmatrix}\begin{pmatrix}(\alpha^2-1)W+2\alpha T & 0\\ 0 & 0\end{pmatrix}^*\right) \\
&\qquad + \mathrm{tr}\left(+\begin{pmatrix}(\alpha^2-1)W+2\alpha T & 0\\ 0 & 0\end{pmatrix}\begin{pmatrix}D_{11}(u^\star) & D_{12}(u^\star)\\ D_{21}(u^\star) & D_{22}(u^\star)\end{pmatrix}^*\right) \\
&\qquad + \mathrm{tr}\left(\begin{pmatrix}D_{11}(u^\star) & D_{12}(u^\star)\\ D_{21}(u^\star) & D_{22}(u^\star)\end{pmatrix}\begin{pmatrix}(\alpha^2-1)W+2\alpha T & 0\\ 0 & 0\end{pmatrix}^*\right) \\
&\qquad + \mathrm{tr}\left(\begin{pmatrix}D_{11}(u^\star) & D_{12}(u^\star)\\ D_{21}(u^\star) & D_{22}(u^\star)\end{pmatrix}\begin{pmatrix}D_{11}(u^\star) & D_{12}(u^\star)\\ D_{21}(u^\star) & D_{22}(u^\star)\end{pmatrix}^*\right).
\end{aligned}
$$

Then, by using the notation in the theorem and after some simple algebra computations, we can obtain

$$\|C-B(\alpha)+\Phi'(u^\star)\|_F^2 = a\alpha^4+2b\alpha^3+(-2a+4c+\eta)\alpha^2+2(\xi-b)\alpha+(a+\delta-\eta)=h(\alpha).$$

Therefore, by taking the first derivative of $h(\alpha)$ to be zero, it follows $h'(\alpha_{opt})=0$. Then the solution that satisfies the second derivative $h''(\alpha)>0$ is the exact optimal parameter α_{opt} that we need. $\square$

4. Numerical Experiments

In this section, we will testify the effectiveness of the C-to-R-based iteration methods by numerical experiments. All the tests are performed in MATLAB R2017a [version 9.2.0.538062] in double precision, on a personal computer with 2.40 GHz central processing unit (Intel(R) Core(TM) 2 Duo CPU), 4.00 GB memory and Windows 64-bit operating system. In our calculations, the stopping criterion for the proposed methods is the current relative residual satisfying $\frac{\|r_k\|_2}{\|r_0\|}<10^{-6}$, where r_k is the residual at the k-th iteration with $u^{(k)}$ being the k-th approximate solution of Equation (1). We use the zero vector as the initial guess. To show the advantages of the new methods, we compare the C-to-R-based methods with the methods listed in Table 1, which shows the abbreviations and the corresponding full description. All the parameter choices in our experiments are listed in Table 2 and we classify the cases as follows.

Table 1. Abbreviations and the corresponding description of the proposed tested methods.

Abbreviation	Description
N-G	Newton method using the GMRES method as inner iteration process
N-H	Newton method using the HSS method as inner iteration process
HSS-N-G	Newton method using the HSS preconditioned GMRES method as inner iteration process
P-G	Picard method using the GMRES method as inner iteration process
P-H	Picard method using the HSS method as inner iteration process
HSS-P-G	Picard method using the HSS preconditioned GMRES method as inner iteration process
N-HSS	nonlinear HSS-like method
P-C	Picard method using the C-to-R ietrative method as inner iteration process
CP-P-G	Picard method using the C-to-R preconditioned GMRES method as inner iteration process
N-C	nonlinear C-to-R-based iteration method

In addition, we set the stopping criterion for the inner iteration process of all the methods to be

$$\frac{\|\phi'(u^{(k)})s^{(k,l_k)} + \phi(u^{(k)})\|_2}{\|\phi(u^{(k)})\|_2} \le \eta_k,$$

where l_k is the inner iteration steps number, η_k is the prescribed inner tolerance. Here, we fix the η_k simply by $\eta = 0.1$ for all k.

Example 1. *Consider the following time dependent nonlinear equation [1,37]:*

$$\begin{cases} u_t - (\beta_1 + \boldsymbol{i}\gamma_1)(u_{xx} + u_{yy}) + \varrho u = -(\beta_2 + \boldsymbol{i}\gamma_2)\sin(\sqrt{1 + u_x^2 + u_y^2}), & \text{for} \quad (x,y) \in (0,1] \times \Omega \\ u(0,x,y) = u_0(x,y), & \text{for} \quad (x,y) \in \Omega \\ u(t,x,y) = 0, & \text{for} \quad (x,y) \in (0,1] \times \partial\Omega, \end{cases}$$

where $\Omega = (0,1) \times (0,1)$, with $\partial\Omega$ being its boundary. ϱ is a positive constant that measures the magnitude of the reaction term.

By applying the centered finite element different scheme with the space step size $h = \frac{1}{N+1}$ and the implicit scheme with the temporal step size $\triangle t = h$, we can obtain the following nonlinear equations

$$Au = \phi(u),$$

where

$$A = h(1 + \varrho\triangle t)I_n + (\beta_1 + \mathbf{i}\gamma_1)\frac{\triangle t}{h}(A_N \otimes I_N + I_N \otimes A_N), \quad A_N = \text{tridiag}(-1, 2, -1) \in \mathbb{R}^{N \times N},$$
$$\phi(u) = (\beta_2 + \mathbf{i}\gamma_2)h\triangle t \cdot \sin(\sqrt{1 + u_x^2 + u_y^2}).$$

Here $n = N^2$ and $\otimes$ denoted the Kronecker product symbol. I_n and I_N are identity matrices of size $n \times n$ and $N \times N$, respectively.

Table 2. Cases with respect different choices of parameters.

$\varrho \setminus (\beta_1 + \mathbf{i}\gamma_1, \beta_2 + \mathbf{i}\gamma_2) =$	$(1 + \mathbf{i}, 1 + \mathbf{i})$	$(0.5 + \mathbf{i}, 1 + 0.5\mathbf{i})$
1	**Case 1.1**	**Case 2.1**
10	**Case 1.2**	**Case 2.2**
100	**Case 1.3**	**Case 2.3**

Our numerical results are presented for problem sizes $N = 16, 32, 64, 128, 256$, and 512 (i.e., $n = 16^2, 32^2, 64^2, 128^2, 256^2$, and 512^2). First, we search the optimal parameter that minimizes the inner iteration count number by varying the parameter from 0.1 to 1 by step size 0.1. If the iteration number decreases when α decreases, then we will expand the searching area with extra $[0.01, 0.1]$ by step size 0.01, or further $[0.001, 0.01]$ by step size 0.001. Therefore, in Table 3, we give the experimental optimal parameters for all the proposed methods with respect to different cases and mesh grids.

Table 3. The experimental optimal parameters with respect to different cases and mesh grids.

	$N \times N$:	**16 × 16**	**32 × 32**	**64 × 64**	**128 × 128**	**256 × 256**	**512 × 512**
Case 1.1	N-H	0.6	0.4	0.2	0.2	0.2	0.1
	HSS-N-G	0.6	0.5	1	1	1	1
	P-H	0.7	0.4	0.2	0.08	0.01	0.01
	HSS-P-G	0.3	0.3	0.3	0.3	0.3	0.3
	N-HSS	0.1	0.1	0.1	0.1	0.1	0.1
	P-C	0.6	0.7	0.7	0.7	0.8	0.8
	CP-P-G	0.6	0.7	0.8	0.7	0.7	0.7
	N-C	0.001	0.4	0.1	0.5	1	1.3
Case 1.2	N-H	0.5	0.4	0.2	0.2	0.1	0.1
	HSS-N-G	1	1	1	1	1	1
	P-H	0.8	0.8	0.3	0.2	0.1	0.1
	HSS-P-G	0.8	0.8	0.3	0.2	0.1	0.1
	N-HSS	0.1	0.2	0.2	0.3	0.4	0.4
	P-C	0.7	0.7	0.7	0.7	0.8	0.8
	CP-P-G	0.6	0.8	0.8	0.8	0.8	0.8
	N-C	0.001	0.04	0.1	0.5	1	1.3
Case 1.3	N-H	1.4	1	0.3	0.3	0.2	0.2
	HSS-N-G	0.9	0.9	0.8	0.8	0.8	0.7
	P-H	1.2	0.7	4	0.2	0.1	0.1
	HSS-P-G	1.2	0.7	0.4	0.2	0.1	0.1
	N-HSS	0.1	0.1	0.05	0.02	0.01	0.008
	P-C	0.8	0.8	0.8	0.8	0.8	0.8
	CP-P-G	0.8	0.8	0.8	0.8	0.8	0.8
	N-C	0.001	0.02	0.07	0.2	0.5	0.8
Case 2.1	N-H	0.5	0.3	0.2	0.2	0.1	0.1
	HSS-N-G	0.8	0.3	0.2	0.1	0.1	0.008
	P-H	0.5	0.3	0.2	0.1	0.1	0.08
	HSS-P-G	0.5	0.3	0.2	0.1	0.1	0.08
	N-HSS	0.001	0.02	0.02	0.03	0.04	0.04
	P-C	0.5	0.6	0.7	0.7	0.8	0.8
	CP-P-G	0.5	0.6	0.7	0.7	0.8	0.8
	N-C	0.001	0.05	0.2	0.4	0.7	0.5
Case 2.2	N-H	0.5	0.3	0.2	0.2	0.1	0.1
	HSS-N-G	0.5	0.3	0.2	0.2	0.1	0.1
	P-H	0.5	0.3	0.2	0.1	0.1	0.08
	HSS-P-G	0.5	0.3	0.2	0.1	0.1	0.08
	N-HSS	0.04	0.07	0.07	0.08	0.08	0.09
	P-C	0.6	0.6	0.7	0.7	0.8	0.8
	CP-P-G	0.6	0.7	0.8	0.8	0.8	0.9
	N-C	0.001	0.04	0.2	0.4	0.6	0.8
Case 2.3	N-H	0.9	0.5	0.3	0.2	0.2	0.1
	HSS-N-G	0.8	0.8	0.3	0.3	0.2	0.2
	P-H	0.9	0.9	0.3	0.2	0.2	0.1
	HSS-P-G	0.9	0.9	0.3	0.2	0.2	0.1
	N-HSS	0.02	0.05	0.05	0.06	0.07	0.07
	P-C	0.8	0.8	0.8	0.8	0.8	0.8
	CP-P-G	0.8	0.8	0.9	0.9	0.9	0.9
	N-C	0.001	0.02	0.08	0.2	0.5	0.6

The numerical results along with the it_out (i.e., the outer iteration counts), IT (i.e., the total inner iteration counts running through the corresponding method), and CPU (i.e., the elapsed cpu time in seconds) are shown in Tables 4–9. If the total inner iteration count number exceeds 500, or the elapsed CPU time is over 500 in second, or the computational storage is out of memory (especially the Newton-based methods), then we will denote the corresponding results as "-" in the tables.

From Tables 4–9, we find that the outer iteration counts of the Newton-based methods are less than the outer iteration counts of the Picard-based methods. However, the Newton-based methods occupy more CPU time than the Picard-based methods.

Table 4. Iteration counts and elapsed CPU time for **Case 1.1**.

	$N \times N$:	**16 × 16**	**32 × 32**	**64 × 64**	**128 × 128**	**256 × 256**	**512 × 512**
N-G	it_out	7	8	8	–	–	–
	IT	94	194	327	–	–	–
	CPU	0.078	0.359	8.823	–	–	–
N-H	it_out	6	8	9	–	–	–
	IT	48	74	135	–	–	–
	CPU	0.041	0.631	4.531	–	–	–
HSS-N-G	it_out	8	9	10	–	–	–
	IT	31	48	98	–	–	–
	CPU	0.419	3.406	36.891	–	–	–
P-G	it_out	7	8	8	8	–	–
	IT	76	146	240	379	–	–
	CPU	0.046	0.139	3.578	20.891	–	–
P-H	it_out	6	8	9	8	8	–
	IT	41	80	145	327	401	–
	CPU	0.004	0.101	1.124	25.135	231.103	–
HSS-P-G	it_out	8	10	10	10	–	–
	IT	30	44	55	87	–	–
	CPU	0.344	3.309	9.219	75.641	–	–
N-HSS	IT	27	28	29	32	33	33
	CPU	0.016	0.047	0.735	4.203	35.741	302.214
P-C	it_out	6	7	8	7	6	5
	IT	12	14	16	17	17	15
	CPU	0.001	0.016	0.111	3.031	6.516	20.406
CP-P-G	it_out	6	8	8	8	–	–
	IT	8	12	12	12	–	–
	CPU	0.001	0.088	0.484	2.931	–	–
N-C	IT	17	7	9	7	7	6
	CPU	0.001	0.016	0.047	0.406	1.047	4.516

Table 5. Iteration counts and elapsed CPU time for **Case 1.2**.

	$N \times N$:	**16 × 16**	**32 × 32**	**64 × 64**	**128 × 128**	**256 × 256**	**512 × 512**
N-G	it_out	6	7	8	–	–	–
	IT	78	166	314	–	–	–
	CPU	0.047	0.319	8.406	–	–	–
N-H	it_out	6	8	9	–	–	–
	IT	56	75	140	–	–	–
	CPU	0.051	0.638	4.559	–	–	–
HSS-N-G	it_out	7	8	9	–	–	–
	IT	31	56	88	–	–	–
	CPU	0.406	6.75	32.422	–	–	–
P-G	it_out	6	7	8	8	–	–
	IT	64	129	235	365	–	–
	CPU	0.034	0.129	3.422	19.406	–	–

Table 5. *Cont.*

	$N \times N$:	16×16	32×32	64×64	128×128	256×256	512×512
P-H	it_out	6	7	9	9	8	–
	IT	37	111	136	145	218	–
	CPU	0.003	0.162	0.903	18.344	142.541	–
HSS-P-G	it_out	7	9	10	10	–	–
	IT	24	48	53	67	–	–
	CPU	0.281	3.297	9.215	70.516	–	–
N-HSS	IT	26	27	28	34	33	33
	CPU	0.016	0.047	0.735	4.203	35.741	302.214
P-C	it_out	6	7	9	7	6	6
	IT	12	14	18	18	17	18
	CPU	0.001	0.016	0.128	3.037	6.516	24.194
CP-P-G	it_out	6	7	8	8	–	–
	IT	7	12	12	12	–	–
	CPU	0.001	0.086	0.484	2.931	–	–
N-C	IT	17	8	8	6	7	7
	CPU	0.001	0.021	0.041	0.401	1.047	4.844

Table 6. Iteration counts and elapsed CPU time for **Case 1.3**.

	$N \times N$:	16×16	32×32	64×64	128×128	256×256	512×512
N-G	it_out	4	5	6	–	–	–
	IT	46	101	199	–	–	–
	CPU	0.031	0.219	6.531	–	–	–
N-H	it_out	6	6	7	–	–	–
	IT	25	53	84	–	–	–
	CPU	0.023	0.441	2.917	–	–	–
HSS-N-G	it_out	4	5	7	–	–	–
	IT	21	33	60	–	–	–
	CPU	0.313	2.406	25.813	–	–	–
P-G	it_out	6	6	6	8	–	–
	IT	42	78	138	304	–	–
	CPU	0.016	0.078	1.641	16.172	–	–
P-H	it_out	6	6	7	9	9	–
	IT	26	38	85	139	203	–
	CPU	0.002	0.045	0.564	18.078	133.11	–
HSS-P-G	it_out	6	7	8	10	–	–
	IT	16	23	36	54	–	–
	CPU	0.263	1.906	8.859	68.609	–	–
N-HSS	IT	27	27	27	27	28	28
	CPU	0.016	0.047	0.734	3.984	31.406	277.66
P-C	it_out	5	5	6	7	7	7
	IT	14	14	17	20	20	20
	CPU	0.001	0.016	0.093	3.127	7.364	24.969
CP-P-G	it_out	5	5	6	7	–	–
	IT	9	10	12	12	–	–
	CPU	0.001	0.069	0.444	2.916	–	–
N-C	IT	17	8	6	8	7	7
	CPU	0.001	0.021	0.036	0.414	1.047	4.844

Table 7. Iteration counts and elapsed CPU time for **Case 2.1**.

	$N \times N$:	**16 × 16**	**32 × 32**	**64 × 64**	**128 × 128**	**256 × 256**	**512 × 512**
N-G	it_out	7	8	8	–	–	–
	IT	93	186	314	–	–	–
	CPU	0.076	0.338	8.408	–	–	–
N-H	it_out	6	7	9	–	–	–
	IT	36	59	116	–	–	–
	CPU	0.036	0.538	4.156	–	–	–
HSS-N-G	it_out	4	8	8	–	–	–
	IT	17	34	46	–	–	–
	CPU	0.308	5.672	23.408	–	–	–
P-G	it_out	6	7	8	8	–	–
	IT	66	130	242	379	–	–
	CPU	0.036	0.125	3.579	20.891	–	–
P-H	it_out	6	7	9	9	9	–
	IT	32	53	107	122	163	–
	CPU	0.003	0.072	0.812	16.984	91.38	–
HSS-P-G	it_out	7	9	10	10	–	–
	IT	23	33	46	73	–	–
	CPU	0.274	2.063	9.063	71.75	–	–
N-HSS	IT	26	27	28	28	29	29
	CPU	0.016	0.047	0.735	3.985	34.399	278.93
P-C	it_out	6	7	8	7	6	5
	IT	12	14	16	17	17	15
	CPU	0.001	0.016	0.111	3.031	6.516	20.406
CP-P-G	it_out	6	8	8	8	–	–
	IT	9	12	12	12	–	–
	CPU	0.001	0.088	0.484	2.931	–	–
N-C	IT	17	6	9	7	7	5
	CPU	0.001	0.016	0.047	0.406	1.047	4.297

Table 8. Iteration counts and elapsed CPU time for **Case 2.2**.

	$N \times N$:	**16 × 16**	**32 × 32**	**64 × 64**	**128 × 128**	**256 × 256**	**512 × 512**
N-G	it_out	6	7	8	–	–	–
	IT	77	161	302	–	–	–
	CPU	0.043	0.266	7.516	–	–	–
N-H	it_out	6	7	9	–	–	–
	IT	31	50	103	–	–	–
	CPU	0.031	0.438	3.735	–	–	–
HSS-N-G	it_out	7	8	8	–	–	–
	IT	24	34	46	–	–	–
	CPU	0.328	5.359	23.406	–	–	–
P-G	it_out	6	7	8	8	–	–
	IT	65	127	235	365	–	–
	CPU	0.036	0.118	3.469	19.406	–	–
P-H	it_out	6	7	9	9	9	–
	IT	32	53	107	122	153	–
	CPU	0.003	0.072	0.812	16.984	88.38	–

Table 8. *Cont.*

	$N \times N$:	**16 × 16**	**32 × 32**	**64 × 64**	**128 × 128**	**256 × 256**	**512 × 512**
HSS-P-G	it_out	7	9	9	10	–	–
	IT	22	32	39	68	–	–
	CPU	0.266	2.052	9.891	70.953	–	–
N-HSS	IT	26	27	27	29	29	30
	CPU	0.016	0.047	0.734	3.993	34.399	279.93
P-C	it_out	6	7	9	6	7	7
	IT	12	14	18	18	21	21
	CPU	0.001	0.016	0.128	3.072	7.406	24.766
CP-P-G	it_out	6	7	8	8	–	–
	IT	9	10	12	12	–	–
	CPU	0.001	0.063	0.484	2.931	–	–
N-C	IT	17	6	10	6	8	7
	CPU	0.001	0.016	0.052	0.401	1.063	4.844

Table 9. Iteration counts and elapsed CPU time for **Case 2.3**.

	$N \times N$:	**16 × 16**	**32 × 32**	**64 × 64**	**128 × 128**	**256 × 256**	**512 × 512**
N-G	it_out	4	4	5	–	–	–
	IT	45	81	168	–	–	–
	CPU	0.029	0.188	4.75	–	–	–
N-H	it_out	5	6	7	–	–	–
	IT	18	29	62	–	–	–
	CPU	0.016	0.259	2.453	–	–	–
HSS-N-G	it_out	4	4	5	–	–	–
	IT	17	26	33	–	–	–
	CPU	0.308	2.641	19.203	–	–	–
P-G	it_out	6	6	6	7	–	–
	IT	41	72	133	270	–	–
	CPU	0.016	0.072	1.613	15.234	–	–
P-H	it_out	5	6	7	8	8	–
	IT	18	48	62	128	191	–
	CPU	0.002	0.068	0.632	17.953	113.11	–
HSS-P-G	it_out	6	6	7	8	–	–
	IT	13	22	27	45	–	–
	CPU	0.242	1.603	7.703	65.250	–	–
N-HSS	IT	26	27	27	28	27	28
	CPU	0.016	0.047	0.734	3.985	30.813	277.36
P-C	it_out	5	6	6	6	7	7
	IT	14	17	17	18	21	21
	CPU	0.001	0.018	0.109	3.072	7.406	24.766
CP-P-G	it_out	5	4	6	6	–	–
	IT	9	8	9	10	–	–
	CPU	0.001	0.063	0.431	2.663	–	–
N-C	IT	17	7	7	8	8	8
	CPU	0.001	0.016	0.047	0.414	1.103	4.984

Besides, we find that the HSS-based iteration methods are more efficient than the Krylov subspace-based methods in CPU time. Further, the N-HSS method can solve the proposed problem efficiently with the optimal experimental parameters shown in Table 3, keeping the steady iteration count numbers while the mesh grid increases.

However, we also find that as the mesh grid increases, the CPU time of the N-HSS method increases rapidly.

The good news is that the C-to-R-based iterative methods (e.g., P-C and N-C) need the least iteration counts and CPU time. In particular, the NC-to-R method not only keeps a steady iteration count number with respect to different mesh grids, but also increases very slowly in CPU time as the mesh grid increases.

Therefore, we can draw a conclusion that the C-to-R-based iterative methods are the first choice and best choice among all the proposed methods for solving this class of complex symmetric weakly nonlinear equations.

5. Concluding Remarks

In this paper, we focus on the numerical methods for solving a class of weakly nonlinear complex symmetric equations. First, we rewrite the original system as a real-valued form. Then, we propose a C-to-R-based Picard iteration method. This method is actually an inexact Picard iteration method. Therefore, the local convergence can be obtained by making use of some existed results. To further improve the efficiency, we construct a nonlinear C-to-R-based splitting iteration method. The convergence results and the theoretically optimal parameters are analyzed in detail. To illustrate the feasibility and the efficiency of the new methods, we perform some numerical experiments to compare with some classical methods. The numerical results show that our new methods are the most efficient method among all the proposed methods.

Author Contributions: M.-L.Z. deduced the theory, implemented the algorithms with the numerical examples, reviewed the manuscript, and wrote the paper; G.-F.Z. provided some innovative advice. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Natural Science Foundation of China (Nos. 11901324, 11771193, and 11626136), the Natural Science Foundation of Fujian Province (No. 2016J05016), and the Scientific Research Project of Putian University (No. 2015061).

Acknowledgments: The authors would like to thank the referees for the comments and constructive suggestions, which were valuable in improving the quality of the manuscript.

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

References

1. Bai, Z.Z.; Yang, X. On HSS-based iteration methods for weakly nonlinear systems. *Appl. Numer. Math.* **2009**, *59*, 2923–2936. [CrossRef]
2. Bai, Z.Z. A class of two-stage iterative methods for systems of weakly nonlinear equations. *Numer. Algorithms* **1997**, *14*, 295. [CrossRef]
3. Bai, Z.Z. On the convergence of parallel chaotic nonlinear multisplitting Newton-type methods. *J. Comput. Appl. Math.* **1997**, *80*, 317–334. [CrossRef]
4. Bai, Z.Z.; Guo, X.P. On Newton-HSS methods for systems of nonlinear equations with positive-definite Jacobian matrices. *J. Comput. Math.* **2010**, *28*, 235–260.
5. Kelley, C.T. *Iterative Methods for Linear and Nonlinear Equations*; SIAM: Philadelphia, PA, USA, 1995; Volume 16.
6. Ortega, J.M.; Rheinboldt, W.C. *Iterative Solution of Nonlinear Equations in Several Variables*; SIAM: Philadelphia, PA, USA, 1970; Volume 30.
7. Aidara, S. Anticipated backward doubly stochastic differential equations with non-Liphschitz coefficients. *Appl. Math. Nonlinear Sci.* **2019**, *4*, 9–20. [CrossRef]
8. Hassan, S.S.; Reddy, M.P.; Rout, R.K. Dynamics of the modified n-degree Lorenz system. *Appl. Math. Nonlinear Sci.* **2019**, *4*, 315–330. [CrossRef]
9. Bai, Z.Z. Block preconditioners for elliptic PDE-constrained optimization problems. *Computing* **2011**, *91*, 379–395. [CrossRef]

10. Bai, Z.Z. Structured preconditioners for nonsingular matrices of block two-by-two structures. *Math. Comput.* **2006**, *75*, 791–815. [CrossRef]
11. Bai, Z.Z.; Chen, F.; Wang, Z.Q. Additive block diagonal preconditioning for block two-by-two linear systems of skew-Hamiltonian coefficient matrices. *Numer. Algorithms* **2013**, *62*, 655–675. [CrossRef]
12. 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]
13. Bai, Z.Z.; Benzi, M.; Chen, F. On preconditioned MHSS iteration methods for complex symmetric linear systems. *Numer. Algorithms* **2011**, *56*, 297–317. [CrossRef]
14. Hezari, D.; Edalatpour, V.; Salkuyeh, D.K. Preconditioned GSOR iterative method for a class of complex symmetric system of linear equations. *Numer. Linear Algebra Appl.* **2015**, *22*, 761–776. [CrossRef]
15. Axelsson, O.; Kucherov, A. Real valued iterative methods for solving complex symmetric linear systems. *Numer. Linear Algebra Appl.* **2000**, *7*, 197–218. [CrossRef]
16. Axelsson, O.; Neytcheva, M.; Ahmad, B. A comparison of iterative methods to solve complex valued linear algebraic systems. *Numer. Algorithms* **2014**, *66*, 811–841. [CrossRef]
17. Axelsson, O.; Lukáš, D. Preconditioning methods for eddy-current optimally controlled time-harmonic electromagnetic problems. *J. Numer. Math.* **2019**, *27*, 1–21. [CrossRef]
18. Liao, L.D.; Zhang, G.F. Efficient preconditioner and iterative method for large complex symmetric linear algebraic systems. *East Asian J. Appl. Math.* **2017**, *7*, 530–547. [CrossRef]
19. Sherman, A.H. On Newton-iterative methods for the solution of systems of nonlinear equations. *SIAM J. Numer. Anal.* **1978**, *15*, 755–771. [CrossRef]
20. Axelsson, O. A generalized conjugate gradient, least square method. *Numer. Math.* **1987**, *51*, 209–227. [CrossRef]
21. Saad, Y. *Iterative Methods for Sparse Linear Systems*; SIAM: Philadelphia, PA, USA, 2003; Volume 82.
22. Guo, X.P.; Duff, I.S. Semilocal and global convergence of the Newton–HSS method for systems of nonlinear equations. *Numer. Linear Algebra Appl.* **2011**, *18*, 299–315. [CrossRef]
23. Li, C.X.; Wu, S.L. On LPMHSS-based iteration methods for a class of weakly nonlinear systems. *Comput. Appl. Math.* **2018**, *37*, 1232–1249. [CrossRef]
24. Yang, A.L.; Wu, Y.J. Newton-MHSS methods for solving systems of nonlinear equations with complex symmetric Jacobian matrices. *Numer. Algebra Control Optim.* **2012**, *2*, 839. [CrossRef]
25. Li, X.; Wu, Y.J. Accelerated Newton-GPSS methods for systems of nonlinear equations. *J. Comput. Anal. Appl.* **2014**, *17*, 245–254.
26. Zhong, H.X.; Chen, G.L.; Guo, X.P. On preconditioned modified Newton-MHSS method for systems of nonlinear equations with complex symmetric Jacobian matrices. *Numer. Algorithms* **2015**, *69*, 553–567. [CrossRef]
27. Xie, F.; Wu, Q.B.; Dai, P.F. Modified Newton–SHSS method for a class of systems of nonlinear equations. *Comput. Appl. Math.* **2019**, *38*, 19–37. [CrossRef]
28. Chen, M.H.; Dou, W.; Wu, Q.B. DPMHSS-based iteration methods for solving weakly nonlinear systems with complex coefficient matrices. *Appl. Numer. Math.* **2019**, *146*, 328–341. [CrossRef]
29. An, H.B.; Bai, Z.Z. A globally convergent Newton-GMRES method for large sparse systems of nonlinear equations. *Appl. Numer. Math.* **2007**, *57*, 235–252. [CrossRef]
30. Zeng, M.L.; Zhang, G.F. A class of preconditioned TGHSS-based iteration methods for weakly nonlinear systems. *East Asian J. Appl. Math.* **2016**, *6*, 367–383. [CrossRef]
31. Wang, J.; Guo, X.P.; Zhong, H.X. MN-DPMHSS iteration method for systems of nonlinear equations with block two-by-two complex Jacobian matrices. *Numer. Algorithms* **2018**, *77*, 167–184. [CrossRef]
32. Amiri, A.; Darvishi, M.T.; Cordero, A.; Torregrosa, J.R. An efficient iterative method based on two-stage splitting methods to solve weakly nonlinear systems. *Mathematics* **2019**, *7*, 815–831. [CrossRef]
33. Liao, L.D.; Zhang, G.F.; Li, R.X. Optimizing and improving of the C-to-R method for solving complex symmetric linear systems. *Appl. Math. Lett.* **2018**, *82*, 79–84. [CrossRef]
34. Axelsson, O.; Liang, Z.Z. Parameter modified versions of preconditioning and iterative inner product free refinement methods for two-by-two block matrices. *Linear Algebra Its Appl.* **2019**, *582*, 403–429. [CrossRef]
35. Schwartz, J.T.; Karcher, H. *Nonlinear Functional Analysis*; CRC Press: Boca Raton, FL, USA, 1969.

36. Huang, Y.M. A practical formula for computing optimal parameters in the HSS iteration methods. *J. Comput. Appl. Math.* **2014**, *255*, 142–149. [CrossRef]
37. Xie, F.; Lin, R.F.; Wu, Q.B. Modified Newton-DSS method for solving a class of systems of nonlinear equations with complex symmetric Jacobian matrices. *Numer. Algorithms* **2019**, 1–25. [CrossRef]