Research Article

Two-Step Relaxation Newton Method for Nonsymmetric Algebraic Riccati Equations Arising from Transport Theory

Shulin Wu¹ and Chengming Huang²

¹ School of Science, Sichuan University of Science and Engineering, Zigong, Sichuan 643000, China
² School of Mathematics and Statistics, Huazhong University of Science and Technology, Wuhan 430074, China

Correspondence should be addressed to Shulin Wu, wushulin@163.com

Received 26 February 2009; Accepted 20 August 2009

Recommended by Alois Steindl

We propose a new idea to construct an effective algorithm to compute the minimal positive solution of the nonsymmetric algebraic Riccati equations arising from transport theory. For a class of these equations, an important feature is that the minimal positive solution can be obtained by computing the minimal positive solution of a couple of fixed-point equations with vector form. Based on the fixed-point vector equations, we introduce a new algorithm, namely, two-step relaxation Newton, derived by combining two different relaxation Newton methods to compute the minimal positive solution. The monotone convergence of the solution sequence generated by this new algorithm is established. Numerical results are given to show the advantages of the new algorithm for the nonsymmetric algebraic Riccati equations in vector form.

Copyright © 2009 S. Wu and C. Huang. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

1. Introduction

In this paper, we are interested in iteratively solving the algebraic Riccati equation arising from transport theory (see, e.g., [1–5] and references therein):

\[ XCX - XE - AX + B = 0, \]  

(1.1)

where \( A, B, C, E \in \mathbb{R}^{n \times n} \) are given by

\[ A = \Delta - eq^T, \quad B = ee^T, \quad C = qq^T, \quad E = D - qe^T. \]  

(1.2)
Here $e = (1, 1, \ldots, 1)^T$ and $q = (q_1, q_2, \ldots, q_n)^T$ with $q_i = c_i/2\omega_i$ and

$$
\begin{align*}
\Delta &= \text{diag}(\delta_1, \delta_2, \ldots, \delta_n), \\
\delta_i &= \frac{1}{c\omega_i(1 + \alpha)}, \\
D &= \text{diag}(d_1, d_2, \ldots, d_n), \\
d_i &= \frac{1}{c\omega_i(1 - \alpha)}.
\end{align*}
$$

The parameters $c$ and $\alpha$ satisfy $0 < c \leq 1$, $0 \leq \alpha < 1$, and $\{c_i\}_{i=1}^n$ and $\{\omega_i\}_{i=1}^n$ are sets of the composite Gauss-Legendre weights and nodes, respectively, on the interval $[0, 1]$ satisfying

$$
0 < \omega_n < \omega_{n-1} < \cdots < \omega_1, \quad c_i > 0 \quad (i = 1, 2, \ldots, n), \quad \sum_{i=1}^n c_i = 1; \quad (1.4)
$$

see, for example, [4] for details. Clearly, it holds that

$$
0 < \delta_1 < \delta_2 < \cdots < \delta_n, \quad 0 < d_1 < d_2 < \cdots < d_n,
$$

$$
d_i = \delta_i \quad \text{for} \; \alpha = 0, \quad d_i > \delta_i \quad \text{for} \; \alpha \neq 0, \quad i = 1, 2, \ldots, n. \quad (1.5)
$$

It has been shown that problem (1.1) has positive solution in the sense of component-wise; see [2, 4] for details. Since only the minimal positive solution is physically meaningful, the research in the field of iteration methods centers around the computation of the minimal positive solution of (1.1); see, for example, [3, 5–12]. For the discussion on more general nonsymmetric algebraic Riccati equations arising in real world, we refer the interested reader to [13–17].

In the seminal paper by Lu [9], it was shown that the solution $X$ of (1.1) can be written as

$$
X = T \circ \left(uv^T\right) = \left(uv^T\right) \circ T, \quad (1.6)
$$

where $\circ$ denotes the Hadamard product, $T$ is the matrix with elements $T_{i,j} = 1/(\delta_i + d_j)$, and $u, v$ are two vectors satisfying the following vector equations:

$$
f(u, v) = \begin{pmatrix} u - [u \circ (Pv) + e] \\ v - [v \circ (Qu) + e] \end{pmatrix} = 0, \quad (1.7)
$$

where $P = [P_{i,j}] = q_j/(\delta_i + d_j)$ and $Q = [Q_{i,j}] = q_j/(\delta_j + d_i)$.

Let $w = (u^T, v^T)^T \in \mathbb{R}^{2n}$ with $w_i = u_i$ and $w_{n+i} = v_i$, $i = 1, 2, \ldots, n$, and let $g(w) = (g_1(w), g_2(w), \ldots, g_{2n}(w))^T$ with

$$
g_i(w) = \begin{cases} 
\sum_{j=1}^n P_{i,j}w_{n+j} & \text{for } 1 \leq i \leq n, \\
\sum_{j=1}^n Q_{i-n,j}w_{j} & \text{for } n + 1 \leq i \leq 2n. 
\end{cases} \quad (1.8)
$$
Then the vector equations (1.7) can be uniformly rewritten as

$$w = w \circ g(w) + e.$$  \hspace{1cm} (1.9)

Based on the vector equations (1.7), Lu [9] investigated the simple iteration (SI) method to compute the minimal positive solution as follows:

$$u_{k+1} = u_k \circ (Pv_k) + e,$$
$$v_{k+1} = v_k \circ (Qu_k) + e,$$
$$v_0 = 0, \quad u_0 = 0.$$  \hspace{1cm} (1.10)

It was shown that the solution sequence \{u_k, v_k\} generated by (1.10) converges monotonically to the minimal positive solution of (1.7). We note that at each step the SI method costs about $4n^2$ flops (for the definition of flops, see, e.g., [18]), while the Gauss-Jacobi (GJ) scheme defined by Juang [2]

$$X_{k+1} = T \circ (X_k q + e)\left(X_k^T q + e\right)^T$$  \hspace{1cm} (1.11)

costs about $6n^2$ flops. Hence, the SI method is more efficient than the GJ method.

In (1.10), we note that before computing $v_{k+1}$, we have obtained $u_{k+1}$, which should be a better approximation to $u$ than $u_k$. Based on this consideration, Bao et al. [7] constructed the modified simple iteration method as

$$u_{k+1} = u_k \circ (Pv_k) + e,$$
$$v_{k+1} = v_k \circ (Qu_{k+1}) + e,$$
$$v_0 = 0, \quad u_0 = 0.$$  \hspace{1cm} (1.12)

It was shown theoretically and numerically that the modified simple iteration procedure (1.12) is more efficient than the original one (1.10).

Recently, the so-called nonlinear splitting iteration methods defined as

$$u_{k+1} = u_{k+1} \circ [Pv_k] + e,$$
$$v_{k+1} = v_{k+1} \circ [Qu_k] + e,$$
$$u_{k+1} = u_{k+1} \circ [Pv_k] + e,$$
$$v_{k+1} = v_{k+1} \circ [Qu_k] + e$$  \hspace{1cm} (1.13a)

were investigated by Bai et al. [6] and independently by Lin [19]. In [6], methods (1.13a) and (1.13b) were called nonlinear block Jacobi (NB) iteration method and nonlinear block Gauss-Seidel (NBGS) iteration method, respectively, and it was shown that the solution sequence \{u_k, v_k\} generated by the NB method and the NBGS method converges monotonically to the
minimal positive solution of (1.7). Moreover, numerical experiments given in [6, 19] show that the convergence speed of these two methods is higher than that of the SI method (1.10).

In this paper, based on the fixed-point equations (1.7), we present an accelerated version of the NBJ scheme, namely, two-step relaxation Newton method, to compute the minimal positive solution. The construction of the new method is based on the relaxation Newton method introduced by Wu et al. [20, 21]. At each iteration, the new algorithm is composed of two-steps: we use the NBJ method as a simple relaxation Newton method to obtain a coarse approximation of the solution of (1.7), and then with this coarse approximation at hand we use a relative complicated relaxation Newton method to get a finer approximation. It is shown that the solution sequence generated by this method converges monotonically to the minimal positive solution of (1.7). We also prove that the new method is more efficient than the NBJ method and its two-step version.

The remainder of this paper is organized as follows. In Section 2, we introduce the relaxation Newton method and its two-step version. Section 3 is concerned with the monotone convergence of the new method. In Section 4, we test some numerical experiments to compare the performance of the new method with the SI method and the NBJ method in the sense of iteration number and CPU time. In the end of the work of this paper, we have constructed another two-step relaxation Newton algorithm which performs much better than the SI, NBJ methods; unfortunately, at the moment we cannot theoretically prove the convergence of this method to the minimal positive solution of the vector equations (1.7). Therefore, we will just report in Section 4 the numerical results of this method.

2. The Two-Step Relaxation Newton Method

In this section, we focus on introducing the basic idea of the relaxation Newton method investigated by [20, 21] and its two-step version for the following general nonlinear equations:

\[ \mathcal{F}(x) = 0. \]  

(2.1)

2.1. The Relaxation Newton Algorithm

The key idea of the relaxation Newton algorithm is to choose some splitting function \( F: \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n \) which is minimally assumed to satisfy a consistency condition

\[ F(x, x) = \mathcal{F}(x) \]  

(2.2)

for any \( x \in \mathbb{R}^n \). Then with an initial guess \( x_0 \) of the unknown solution \( x^* \), we start with the previous approximation \( x_k \) to compute the next approximation \( x_{k+1} \) by solving the following problem:

\[ F(x_k, x_{k+1}) = 0 \]  

(2.3)

with some conventional method, such as the classical Newton’s method, quasi-Newton methods, and Conjugate-Gradient method. Obviously, the generated sequence \( \{x_k\}_0^\infty \) will upon convergence approach to some value \( x^* \) which satisfies \( F(x^*, x^*) = 0 \), that is, \( \mathcal{F}(x) = 0 \).
for $k = 0, 1, 2, \ldots$  
with a given initial approximation $\tilde{x}_0$ of $x_{k+1}$;  
for $m = 0, 1, \ldots, M$  
$F_2(x_k, \tilde{x}_m) \Delta \tilde{x}_m = -F(x_k, \tilde{x}_m)$,  
$\tilde{x}_{m+1} = \tilde{x}_m + \Delta \tilde{x}_m$,  
end  
$x_{k+1} = \Delta \tilde{x}_M$
end

Algorithm 1: The relaxation Newton method.

Wu et. al. [20, 21] used the classical Newton’s method to solve (2.3), which explains the name: relaxation Newton; the deduced algorithm written compactly is shown in Algorithm 1. In Algorithm 1 and what follows

$$F_2(x, y) = \frac{\partial F(x, z)}{\partial z} \bigg|_{z=y}.$$  

(2.4)

If we set $\tilde{x}_0 = x_k$ and $M = 1$ in the relaxation algorithm shown in Algorithm 1, by the consistency condition (2.2), the method can be written as

$$F_2(x_k, x_k) \Delta x_k = -\mathcal{F}(x_k), \quad x_{k+1} = x_k + \Delta x_k, \quad k = 0, 1, \ldots.$$  

(2.5)

With a special choice of $F$, the Jacobi matrix $F_2(x, x)$ will be a diagonal or block diagonal matrix and invertible in $\mathbb{R}^n$, and thus iteration method (2.5) can be processed stably and simultaneously with less storage compared with the classical Newton’s method. We will see in the next section how to select the splitting function $F$ such that the Jacobi matrix $F_2(x, x)$ is a diagonal or block diagonal matrix.

### 2.2. The Two-Step Relaxation Newton Method

Now, suppose that we have two splitting functions $F(x, y), G(x, y, z)$ and an initial approximation $x_0$ of the solution of (2.1) at hand. We start with the previous approximation $x_k$ to compute a mid-approximation $x_{k+1/2}$ by solving the equations

$$F(x_k, x_{k+1/2}) = 0,$$  

(2.6a)

and then with $x_k, x_{k+1/2}$ at hand we compute $x_{k+1}$ by solving the equations

$$G(x_k, x_{k+1/2}, x_{k+1}) = 0.$$  

(2.6b)

The splitting functions $F$ and $G$ are assumed to satisfy the consistency conditions

$$F(x, x) = \mathcal{F}(x), \quad G(x, x, x) = \mathcal{F}(x), \quad \forall x \in \mathbb{R}^n.$$  

(2.7)
Similar to the relaxation Newton algorithm described above, we use Newton’s method with a single iteration to solve equations (2.6a), (2.6b), and the deduced iteration scheme is

\[ F_2(x_k, x_k)(x_{k+1/2} - x_k) = -f(x_k), \tag{2.8a} \]
\[ G_3(x_k, x_{k+1/2}, x_{k+1/2})(x_k - x_{k+1/2}) = -G(x_k, x_{k+1/2}, x_{k+1/2}), \tag{2.8b} \]

where \( F_2 \) is defined by (2.4), and the Jacobi matrix \( G_3(x, y, z) \) of the function \( G \) is defined by

\[ G_3(x, y, z) = \frac{\partial G(x, y, s)}{\partial s} \bigg|_{s=z}. \tag{2.8c} \]

Throughout this paper, the iteration scheme (2.8a), (2.8b), and (2.8c) is called two-step relaxation Newton (denoted by “TSRN”) algorithm.

Specially, for the vector equations (1.7) we consider in this paper the following splitting functions:

\[ F(w_k, w_{k+1/2}) = \begin{pmatrix} u_{k+1/2} - u_{k+1/2} \circ (Pv_k) - e \\ v_{k+1/2} - v_{k+1/2} \circ (Qu_k) - e \end{pmatrix}, \tag{2.9a} \]
\[ G(w_k, w_{k+1/2}, w_{k+1}) = \begin{pmatrix} u_{k+1} - u_{k+1} \circ (Pv_{k+1}) - u_{k+1/2} \circ [\Phi(v_{k+1} - v_{k+1/2})] - e \\ v_{k+1} - v_{k+1} \circ (Qu_{k+1}) - v_{k+1/2} \circ [\Psi(u_{k+1} - u_{k+1/2})] - e \end{pmatrix}, \tag{2.9b} \]

here and hereafter \( w_{k+1/2} = (u_{k+1/2}^T, v_{k+1/2}^T) \) and \( w_k = (u_k^T, v_k^T) \) for all \( k = 0, 1, 2 \ldots \); \( \Phi \) and \( \Psi \) are diagonal matrices and their diagonal elements are determined as

\[ \Phi_{ii} = \begin{cases} P_{ii} & \text{if } i \text{ is odd}, \\ Q_{ii} & \text{if } i \text{ is even}, \\ 0 & \text{if } i \text{ is even}, \end{cases} \quad \Psi_{ii} = \begin{cases} Q_{ii} & \text{if } i \text{ is even}, \\ 0 & \text{if } i \text{ is odd}. \end{cases} \tag{2.10} \]

Generally speaking, the splitting function \( G \) is a three variables function, but here we consider a special case—only the second and third variables are involved; see (2.9b). In the end of Section 4, we will see another TSRN algorithm where the function \( G \) is defined with 3 arguments (see (4.5b)).

Define

\[ D_k = \text{diag}(e - Pv_k), \quad \Lambda_k = \text{diag}(e - Qu_k), \quad \tilde{D}_k = \text{diag}(e - Pv_{k+1/2}), \]
\[ \tilde{\Lambda}_k = \text{diag}(e - Qu_{k+1/2}), \quad A_k = \text{diag}(u_{k+1/2} \circ [\Phi e]), \quad B_k = \text{diag}(v_{k+1/2} \circ [\Psi e]). \tag{2.11} \]

Then it is clear that

\[ F_2(w_k, w_k) = \begin{bmatrix} D_k \\ \Lambda_k \end{bmatrix}, \quad G_3(w_k, w_{k+1/2}, w_{k+1/2}) = \begin{bmatrix} \tilde{D}_k & -A_k \\ -B_k & \tilde{\Lambda}_k \end{bmatrix}. \tag{2.12} \]
Therefore, it follows by substituting the function $f$ defined in (1.7) and the splitting functions $F, G$ defined in (2.9a), (2.9b) into (2.8a), (2.8b), (2.8c) that

$$F_2(w_k, w_k) \begin{pmatrix} u_{k+1/2} \\ v_{k+1/2} \end{pmatrix} = \begin{pmatrix} e \\ e \end{pmatrix},$$

(2.13a)

$$G_3(w_k, w_{k+1/2}, w_{k+1/2}) \begin{pmatrix} u_{k+1} - u_{k+1/2} \\ v_{k+1} - v_{k+1/2} \end{pmatrix} = -f(u_{k+1/2}, v_{k+1/2}).$$

(2.13b)

We note that with the special splitting functions $F$ and $G$ defined in (2.9a), (2.9b), equations (2.13a), (2.13b) implies

$$F(w_k, w_{k+1/2}) = 0, \quad G(w_k, w_{k+1/2}, w_{k+1}) = 0, \quad k = 0, 1, 2, \ldots,$$

(2.14)

while for general splitting functions this is not always true. For diagonal matrices $\tilde{D}_k, \tilde{\Lambda}_k, A_k$ and $B_k$, routine calculation yields

$$[G_3(w_k, w_{k+1/2}, w_{k+1/2})]^{-1} = \begin{bmatrix} \tilde{D}_k \tilde{\Lambda}_k - A_k B_k & -[\tilde{D}_k \tilde{\Lambda}_k - A_k B_k]^{-1} \tilde{\Lambda}_k & A_k \\ B_k - \tilde{D}_k & \tilde{\Lambda}_k & A_k \end{bmatrix}.\quad (2.15)$$

Furthermore, by (2.10) we know that $A_k B_k$ must be zero matrix for all $k = 0, 1, 2 \ldots$; hence

$$[G_3(w_k, w_{k+1/2}, w_{k+1/2})]^{-1} = \begin{bmatrix} \tilde{D}_k^{-1} & \tilde{D}_k \tilde{\Lambda}_k & A_k \\ \tilde{D}_k \tilde{\Lambda}_k & A_k & \tilde{\Lambda}_k^{-1} \end{bmatrix}.\quad (2.16)$$

From (2.9a), (2.9b), (2.10), and (2.16) we obtain the algorithm for implementing the TSRN method as follows.

**Algorithm 1 (Two-step relaxation Newton iteration).** Starting with the initial value $(u_0, v_0)$, the solution $(u_k, v_k)$ is defined, for $k = 0, 1, 2, \ldots$, by

1. computing explicitly in the following elementwise fashion:

$$u_{k+1/2,i} = \frac{1}{1 - [Pv_k]_i}, \quad v_{k+1/2,i} = \frac{1}{1 - [Qu_k]_i}, \quad i = 1, 2, \ldots, n;$$

(2.17)

2. performing for $i = 1$ to $n$:

   if $i$ is odd

   $$u_{k+1,i} = \frac{1}{1 - [Pv_{k+1/2}]_i} \left(1 - \Phi_{ii} u_{k+1/2}, v_{k+1/2,j} + \frac{\Phi_{ii} u_{k+1/2} + v_{k+1/2,j}}{1 - [Qu_{k+1/2}]_i}\right),$$

   (2.18)

   $$v_{k+1,i} = \frac{1}{1 - [Qu_{k+1/2}]_i},$$
$$u_{k+1,i} = \frac{1}{1 - [Pu_{k+1/2}]_i}$$

$$v_{k+1,i} = \frac{1}{1 - [Qu_{k+1/2}]_i} \left( 1 - \Psi_{ii} u_{k+1/2,i} v_{k+1/2,i} + \frac{\Psi_{ii} v_{k+1/2,i}}{1 - [Pu_{k+1/2}]_i} \right);$$

(2.19)

(3) if $\|f(u_{k+1}, v_{k+1})\|_\infty \leq \text{Tol}$, stop iteration; else set $u_k = u_{k+1}, v_k = v_{k+1}$ and go to the first step in Algorithm 1.

Clearly, compared with the Newton-type methods investigated by Lu [10] and Lin et al. [8], no LU-factorization is needed at each iteration of the TSRN method.

### 3. Convergence Analysis

For our proof of the monotone convergence of the TSRN method, we need the following results proved by Lu [9].

**Lemma 3.1** (see [9]). With the function $g$ defined in (1.8), one has

1. $u^* > e$, $v^* > e$,
2. $\max_{1 \leq i \leq 2n} \{g_i(e)\} < 1$.

The first conclusion can be obtained straightforwardly from (1.7).

**Theorem 3.2.** Let $(u_0, v_0) = (0, 0)$ be the initial point of the TSRN method. The solution sequence \{$(u_k, v_k)^{\infty}_{k=1}\}$ generated by the TSRN method is strictly and monotonically increasing and converges to the minimal positive solution $(u^*, v^*)$ of the vector equations (1.7); particularly, it holds that

1. $0 \leq u_k < u_{k+1/2} < u_{k+1} < u^*$ and $0 \leq v_k < v_{k+1/2} < v_{k+1} < v^*$, $k = 0, 1, 2, \ldots$;
2. $\lim_{k \to \infty} u_k = u^*$ and $\lim_{k \to \infty} v_k = v^*$.

**Proof.** We first prove the first conclusion of Theorem 3.2 by induction rule and this is completed in two-steps: first, we prove the correctness of this conclusion for $k = 0$, and then under the induction assumption with index $k = j$, we prove the case for $k = j + 1$.

(1) It is easy to get $(u_{1/2}, v_{1/2}) = (e, e)$. Hence, from the first conclusion of Lemma 3.1, it holds $u_{1/2} < u^*$ and $v_{1/2} < v^*$. By the second conclusion of Lemma 3.1, we get $e - Pu_{1/2} > 0$ and $e - Qu_{1/2} > 0$. Therefore, from (2.16) we know $[G_3(w_0, w_{1/2}, w_{1/2})]^{-1} > 0$. Since

$$u_{1/2} \circ (e - Pu_0) - e = 0, \quad v_{1/2} \circ (e - Qu_0) - e = 0, \quad u_{1/2} > u_0, \quad v_{1/2} > v_0, \quad (3.1)$$

we have $u_{1/2} \circ (e - Pu_{1/2}) - e < 0$ and $v_{1/2} \circ (e - Qu_{1/2}) - e < 0$, that is, $f(u_{1/2}, v_{1/2}) < 0$. Hence, it follows from (2.13b) that $w_1 = w_{1/2} - [G_3(w_0, w_{1/2}, w_{1/2})]^{-1}f(u_{1/2}, v_{1/2}) > w_{1/2}$. We note that for any $\tilde{w}, w, w_-, w_+ \in \mathbb{R}^{2n}$ it holds

$$G(\tilde{w}, w, w_-) = G(\tilde{w}, w, w_+) + G_3(\tilde{w}, w, w_+)(w_- - w_+). \quad (3.2)$$
Hence, \( G(w_0, w_{1/2}, w^*) + G_3(w_0, w_{1/2}, w^*)(w_1 - w^*) = G(w_0, w_{1/2}, w) = 0 \), that is,

\[
    w_1 - w^* = -[G_3(w_0, w_{1/2}, w^*)]^{-1}G(w_0, w_{1/2}, w^*),
\]

where \( w^* = \left( \begin{array}{c} u^* \\ v^* \end{array} \right) \). It is easy to verify \( G_3(w_0, w_{1/2}, w^*) = G_3(w_0, w_{1/2}, w_{1/2}) \) and this implies \( [G_3(w_0, w_{1/2}, w^*)]^{-1} > 0 \). Consider

\[
    u^* \circ (e - P v^*) = e,
    v^* \circ (e - Q u^*) = e,
\]

\[
    u^* \circ (e - P v_{1/2}) - u_{1/2} \circ \Phi(v^* - v_{1/2}) - u^* \circ (e - P v^*) \\
    = u^* \circ [P(v^* - v_{1/2})] - u_{1/2} \circ \Phi(v^* - v_{1/2}),
\]

\[
    v^* \circ (e - Q u_{1/2}) - v_{1/2} \circ \Psi(u^* - u_{1/2}) - v^* \circ (e - Q u^*) \\
    = v^* \circ [Q(u^* - u_{1/2})] - v_{1/2} \circ \Psi(u^* - u_{1/2}),
\]

and \( u_{1/2} < u^*, v_{1/2} < v^* \), and we know that the right hand of (3.5) is positive, and this implies \( G(w_0, w_{1/2}, w^*) > 0 \). Therefore, from (3.3) we get \( w_1 < w^* \). We have proved that the first conclusion of Theorem 3.2 is correct for \( k = 0 \).

(2) Assume that the first conclusion of Theorem 3.2 is correct for \( k = j, j \geq 0 \) and we will prove that it is also correct for \( k = j + 1 \). To this end, we note that under the induction assumption it holds

\[
    e > e - Qu_j > e - Qu_{j+1/2} > e - Qu_{j+1} > e - Qu^* > 0,
    e > e - Pv_j > e - Pv_{j+1/2} > e - Pv_{j+1} > e - P v^* > 0,
\]

where the inequalities \( e - Qu^* > 0 \) and \( e - P v^* > 0 \) follow directly from equality (3.4). Consider

\[
    u_{j+1} \circ (e - P v_{j+1/2}) - u_{j+1/2} \circ [\Phi(v_{j+1} - v_{j+1/2})] - u_{j+1} \circ (e - P v_{j+1}) \\
    = u_{j+1} \circ [P(v_{j+1} - v_{j+1/2})] - u_{j+1/2} \circ [\Phi(v_{j+1} - v_{j+1/2})] > 0,
\]

\[
    v_{j+1} \circ (e - Q u_{j+1/2}) - v_{j+1/2} \circ [\Psi(u_{j+1} - u_{j+1/2})] - v_{j+1} \circ (e - Q u_{j+1}) \\
    = v_{j+1} \circ [Q(u_{j+1} - u_{j+1/2})] - v_{j+1/2} \circ [\Psi(u_{j+1} - u_{j+1/2})] > 0,
\]

and since \( w_{j+1/2} < w_{j+1} \), we get

\[
    u_{j+1/2} \circ (e - P v_{j+1}) = e = u_{j+1} \circ (e - P v_{j+1/2}) - u_{j+1/2} \circ [\Phi(v_{j+1} - v_{j+1/2})] \\
    > u_{j+1} \circ (e - P v_{j+1}),
\]

\[
    v_{j+1/2} \circ (e - Q u_{j+1}) = e = v_{j+1} \circ (e - Q u_{j+1/2}) - v_{j+1/2} \circ [\Psi(u_{j+1} - u_{j+1/2})] \\
    > u_{j+1} \circ (e - Q u_{j+1}),
\]
and this coupled with (3.6) gives

\[ w_{j+1} < w_{j+1/2}. \]  

(3.9)

From (3.6) and \( w_{j+1} < w^* \), we have

\[ u_{j+1/2} \circ (e - P v_{j+1}) = e = u^* \circ (e - P v^*) < u^* \circ (e - P v_{j+1}) \implies u_{j+1/2} < u^*, \]

(3.10)

\[ v_{j+1/2} \circ (e - Q u_{j+1}) = e = v^* \circ (e - Q v^*) < v^* \circ (e - Q u_{j+1}) \implies v_{j+1/2} < v^*, \]

that is, \( w_{j+1/2} < w^* \); this coupled with (3.6) implies \( e - P v_{j+1} > 0 \) and \( e - Q u_{j+1} > 0 \). Hence, we arrive at

\[ [G_3(w_{j+1}, w_{j+1/2}, w_{j+1/2})]^{-1} > 0. \]

(3.11)

From (3.9) we have

\[ u_{j+1/2} \circ (e - P v_{j+1/2}) - e < u_{j+1/2} \circ (e - P v_{j+1}) - e = 0 \]

\[ v_{j+1/2} \circ (e - Q u_{j+1/2}) - e < v_{j+1/2} \circ (e - Q u_{j+1}) - e = 0 \implies f(u_{j+1/2}, v_{j+1/2}) < 0; \]

(3.12)

this coupled with (3.11) gives

\[ w_{j+2} = w_{j+1/2} - [G_3(w_{j+1}, w_{j+1/2}, w_{j+1/2})]^{-1} f(u_{j+1/2}, v_{j+1/2}) > w_{j+1/2}. \]

(3.13)

Next, we prove \( w_{j+2} < w^* \). To this end, from (3.2) and (2.14) we have

\[ G(w_{j+1}, w_{j+1/2}, w^*) + G_3(w_{j+1}, w_{j+1/2}, w^*)(w_{j+2} - w^*) = G(w_{j+1}, w_{j+1/2}, w_{j+2}) = 0, \]

(3.14)

and this implies

\[ w_{j+2} - w^* = -[G_3(w_{j+1}, w_{j+1/2}, w^*)]^{-1} G(w_{j+1}, w_{j+1/2}, w^*). \]

(3.15)

From (3.11), we get

\[ [G_3(w_{j+1}, w_{j+1/2}, w^*)]^{-1} = [G_3(w_{j+1}, w_{j+1/2}, w_{j+1/2})]^{-1} > 0, \]

(3.16)
where the first equality can be easily verified from (2.9b). Consider (3.4) and the following relations:

\[
\begin{align*}
    u^* \circ (e P\nu_{j+1/2}^*) - u_{j+1/2}^* \circ \Phi(\nu^* - \nu_{j+1/2}^*) - u^* \circ (e \circ \nu^*) & = u^* \circ [P(\nu^* - \nu_{j+1/2}^*)] - u_{j+1/2}^* \circ \Phi(\nu^* - \nu_{j+1/2}^*) > 0, \\
    \nu^* \circ (e \circ Q\nu_{j+1/2}^*) - \nu_{j+1/2}^* \circ \Psi(\nu^* - \nu_{j+1/2}^*) - \nu^* \circ (e \circ Q\nu^*) & = \nu^* \circ [Q(\nu^* - \nu_{j+1/2}^*)] - \nu_{j+1/2}^* \circ \Psi(\nu^* - \nu_{j+1/2}^*) > 0,
\end{align*}
\]

(3.17)

and since \(u_{j+1/2}^* < u^*, \nu_{j+1/2}^* < \nu^*,\) we have \(G(w_{j+1}, w_{j+1/2}, \nu^*) > 0;\) this coupled with (3.15) and (3.16) gives

\[
w_{j+2} < \nu^*.
\]

(3.18)

By (3.9), (3.13) and (3.18) we have proved the validity of the first conclusion of Theorem 3.2 for \(k = j + 1.\) Therefore, by induction rule we have completed the proof of the first conclusion of Theorem 3.2.

From the first conclusion of Theorem 3.2, it is obvious that there exist positive vectors \(\tilde{u}^*\) and \(\tilde{v}^*\) such that

\[
\lim_{k \to +\infty} u_k = \lim_{k \to +\infty} u_{k+1/2} = \tilde{u}^*, \quad \lim_{k \to +\infty} \nu_k = \lim_{k \to +\infty} \nu_{k+1/2} = \tilde{v}^*.
\]

(3.19)

Therefore, it follows from (2.9a), (2.9b), and (2.14) that

\[
\tilde{u}^* = \tilde{u}^* \circ (P\tilde{v}^*) + e, \\
\tilde{v}^* = \tilde{v}^* \circ (Q\tilde{u}^*) + e.
\]

(3.20)

This implies that \((\tilde{u}^*, \tilde{v}^*)\) is a positive solution of the vector equations (1.7). According to the minimal property of \((u^*, v^*),\) it must hold \(\tilde{u}^* = u^*\) and \(\tilde{v}^* = v^*.\)

To finish this section, we compare the efficiency of the TSRN method with the NBJ method. To this end, we rewrite the original NBJ iteration scheme (1.13a) into the following form:

\[
\begin{align*}
    \mathcal{U}_{k+1/2} &= \mathcal{U}_{k+1/2} \circ [P\mathcal{U}_k] + e, \\
    \mathcal{U}_{k+1/2} &= \mathcal{U}_{k+1/2} \circ [Q\mathcal{U}_k] + e, \\
    \mathcal{U}_{k+1} &= \mathcal{U}_{k+1} \circ [P\mathcal{U}_{k+1/2}] + e, \\
    \mathcal{U}_{k+1} &= \mathcal{U}_{k+1} \circ [Q\mathcal{U}_{k+1/2}] + e.
\end{align*}
\]

(3.21)

Clearly, the TSRN method will reduce into the two-step version NBJ method (3.21) if \(\Psi = \Phi = 0\) in (2.9b).

The following theorem indicates that the TSRN method is more efficient than the NBJ method.
Theorem 3.3. Let both the NBJ method and the TSRN method start with the initial value \((0, 0)\), and \(\{u_k, v_k\}, \{\U_k, \V_k\}\) be the sequences generated by the TSRN method and the NBJ method, respectively. Then it holds that

\[
\begin{align*}
\U_{k+1/2} &< u_{k+1/2}, \\
\V_{k+1/2} &< v_{k+1/2}, \\
\U_{k+1} &< u_{k+1}, \\
\V_{k+1} &< v_{k+1}
\end{align*}
\] (3.22)

for \(k \geq 2\).

Proof. It is easy to get \((u_{1/2}, v_{1/2}) = (\U_{1/2}, \V_{1/2}) = (e, e)\). Hence, we have

\[
\begin{align*}
(u_1 - \U_1) \circ [e - Pe] &= u_{1/2} \circ [\Phi(v_1 - v_{1/2})], \\
(v_1 - \V_1) \circ [e - Qe] &= v_{1/2} \circ [\Psi(u_1 - u_{1/2})].
\end{align*}
\] (3.23)

This coupled with (2.10), the second conclusion of Lemma 3.1, and the first conclusion of Theorem 3.2 gives

\[
\begin{align*}
u_{i,j} &> \U_{i,j}, \quad \text{if } i \text{ is odd,} \\
u_{i,j} &= \U_{i,j}, \quad \text{if } i \text{ is even,} \\
v_{i,j} &> \V_{i,j}, \quad \text{if } i \text{ is even,} \\
v_{i,j} &= \V_{i,j}, \quad \text{if } i \text{ is odd.}
\end{align*}
\] (3.24)

Since \(P, Q > 0\), it follows from (3.24) that

\[
\begin{align*}
u_{2+1/2} &\circ [e - Pv_1] = e = \U_{2+1/2} \circ [e - P\U_1] > \U_{2+1/2} \circ [e - Pv_1], \\
v_{2+1/2} &\circ [e - Qu_1] = e = \V_{2+1/2} \circ [e - Q\V_1] > \V_{2+1/2} \circ [e - Qu_1],
\end{align*}
\] (3.25)

where in the last inequality we used the relation (3.6) and the fact established by [6]—the sequence \(\{\U_k, \V_k\}\) is strictly and monotonically increasing and converges to the minimal positive solution \((u^*, v^*)\). From (3.25), we have \(u_{2+1/2} > \U_{2+1/2}\) and \(v_{2+1/2} > \V_{2+1/2}\).

Consider

\[
\begin{align*}
u_3 &\circ [e - P\nu_{2+1/2}] - u_{2+1/2} \circ [\Phi(v_3 - v_{2+1/2})] = e = \U_3 \circ [e - P\U_{2+1/2}] > \U_3 \circ [e - P\nu_{2+1/2}], \\
v_3 &\circ [e - Qu_{2+1/2}] - v_{2+1/2} \circ [\Psi(u_3 - u_{2+1/2})] = e = \V_3 \circ [e - Q\V_{2+1/2}] > \V_3 \circ [e - Qu_{2+1/2}],
\end{align*}
\] (3.26)
In this section, we compute the minimal positive solution of the vector equations (1.7) by the TSRN method, in order to compare numerically the feasibility and effectiveness of this algorithm with the SI method (1.10) and the NBJ method (1.13a) in the sense of number of iteration (denoted as “IT”) and elapsed CPU time in seconds (denoted as “CPU”). Clearly, these methods are very suitable to be performed in parallel environment. We remark that the Newton type methods coupled with LU-factorization [6, 10], NBGS method (1.13b), and the modified simple iteration (1.12) are sequential methods and not suitable to be implemented in parallel environment.

In all experiments, the constants \( c_i \) and \( \omega_i \), \( i = 1, 2, \ldots, n \), are given by the composite Gauss-Legendre quadrature formula on the interval [0, 1]. More precisely, the interval [0, 1] is first divided into subintervals of equal length, and the composite Gauss-Legendre quadrature formula with 4 nodes is then applied to each subinterval (see [14]). In our implementations, all iterations start with initial value \((0, 0)\) and are terminated once the current residual error defined as

\[
\text{ERR}_k = \max \{ \| u_k - u_k \circ (Pv_k) - e \|_\infty, \| v_k - v_k \circ (Qu_k) - e \|_\infty \} \tag{4.1}
\]

satisfies \( \text{ERR}_k \leq 10^{-13} \). All codes are performed in MATLAB (version 7.0) on an Intel (R) Pentium (R) Dual E2110 @ 1.4 GHz PC with memory 1 GB.

To make a fair comparison, we rewrite equivalently the SI method (1.10) into two-step fashion as follows:

\[
\begin{align*}
    u_{k+1/2} & = u_k \circ [Pv_k] + e, \\
v_{k+1/2} & = v_k \circ [Qu_k] + e, \\
u_{k+1} & = u_{k+1/2} \circ [Pv_{k+1/2}] + e, \\
v_{k+1} & = v_{k+1/2} \circ [Qu_{k+1/2}] + e,
\end{align*}
\tag{4.2}
\]

and the vectors \( u_{k+1/2}, v_{k+1/2} \) and \( u_{k+1}, v_{k+1} \) are explicitly computed in elementwise as

\[
\begin{align*}
    u_{k+1/2,i} & = u_{k,i}[Pv_k]_i + 1, \\
v_{k+1/2,i} & = v_{k,i}[Qu_k]_i + 1, \\
u_{k+1,i} & = u_{k+1/2,i}[Pv_{k+1/2}]_i + 1, \\
v_{k+1,i} & = v_{k+1/2,i}[Qu_{k+1/2}]_i + 1, \quad i = 1, 2, \ldots
\end{align*}
\tag{4.3}
\]

Remark 3.4. Since the NBJ method is more feasible than the NBGS method in parallel environment, in this paper we only focus on comparing the efficiency between the TSRN and the NBJ methods.

4. Numerical Results

In this section, we compute the minimal positive solution of the vector equations (1.7) by the TSRN method, in order to compare numerically the feasibility and effectiveness of this algorithm with the SI method (1.10) and the NBJ method (1.13a) in the sense of number of iteration (denoted as “IT”) and elapsed CPU time in seconds (denoted as “CPU”). Clearly, these methods are very suitable to be performed in parallel environment. We remark that the Newton type methods coupled with LU-factorization [6, 10], NBGS method (1.13b), and the modified simple iteration (1.12) are sequential methods and not suitable to be implemented in parallel environment.

In all experiments, the constants \( c_i \) and \( \omega_i \), \( i = 1, 2, \ldots, n \), are given by the composite Gauss-Legendre quadrature formula on the interval [0, 1]. More precisely, the interval [0, 1] is first divided into subintervals of equal length, and the composite Gauss-Legendre quadrature formula with 4 nodes is then applied to each subinterval (see [14]). In our implementations, all iterations start with initial value \((0, 0)\) and are terminated once the current residual error defined as

\[
\text{ERR}_k = \max \{ \| u_k - u_k \circ (Pv_k) - e \|_\infty, \| v_k - v_k \circ (Qu_k) - e \|_\infty \} \tag{4.1}
\]

satisfies \( \text{ERR}_k \leq 10^{-13} \). All codes are performed in MATLAB (version 7.0) on an Intel (R) Pentium (R) Dual E2110 @ 1.4 GHz PC with memory 1 GB.

To make a fair comparison, we rewrite equivalently the SI method (1.10) into two-step fashion as follows:

\[
\begin{align*}
    u_{k+1/2} & = u_k \circ [Pv_k] + e, \\
v_{k+1/2} & = v_k \circ [Qu_k] + e, \\
u_{k+1} & = u_{k+1/2} \circ [Pv_{k+1/2}] + e, \\
v_{k+1} & = v_{k+1/2} \circ [Qu_{k+1/2}] + e,
\end{align*}
\tag{4.2}
\]

and the vectors \( u_{k+1/2}, v_{k+1/2} \) and \( u_{k+1}, v_{k+1} \) are explicitly computed in elementwise as

\[
\begin{align*}
    u_{k+1/2,i} & = u_{k,i}[Pv_k]_i + 1, \\
v_{k+1/2,i} & = v_{k,i}[Qu_k]_i + 1, \\
u_{k+1,i} & = u_{k+1/2,i}[Pv_{k+1/2}]_i + 1, \\
v_{k+1,i} & = v_{k+1/2,i}[Qu_{k+1/2}]_i + 1, \quad i = 1, 2, \ldots
\end{align*}
\tag{4.3}
\]

RemarK 3.4. Since the NBJ method is more feasible than the NBGS method in parallel environment, in this paper we only focus on comparing the efficiency between the TSRN and the NBJ methods.
Table 1: Numerical results for $n = 32$ and different $(\alpha, c)$.

<table>
<thead>
<tr>
<th>Method</th>
<th>$(0.1, 0.9)$</th>
<th>$(0.001, 0.995)$</th>
<th>$(10^{-5}, 1 - 2 \times 10^{-5})$</th>
<th>$(10^{-7}, 1 - 10^{-7})$</th>
<th>$(10^{-12}, 1 - 10^{-12})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>SI</td>
<td>37</td>
<td>181</td>
<td>2377</td>
<td>24405</td>
<td>71486</td>
</tr>
<tr>
<td>IT</td>
<td>20</td>
<td>84</td>
<td>1040</td>
<td>10609</td>
<td>31092</td>
</tr>
<tr>
<td>TSRN</td>
<td>20</td>
<td>81</td>
<td>1029</td>
<td>10499</td>
<td>30662</td>
</tr>
<tr>
<td></td>
<td>SI</td>
<td>0.018</td>
<td>0.035</td>
<td>0.29</td>
<td>2.7</td>
</tr>
<tr>
<td>CPU</td>
<td>NBJ</td>
<td>0.001</td>
<td>0.026</td>
<td>0.210</td>
<td>1.551</td>
</tr>
<tr>
<td></td>
<td>TSRN</td>
<td>0.001</td>
<td>0.015</td>
<td>0.110</td>
<td>1.270</td>
</tr>
</tbody>
</table>

Table 2: Numerical results for $(\alpha, c) = (10^{-7}, 1)$ and different $n$.

<table>
<thead>
<tr>
<th>Method</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>512</th>
<th>1024</th>
</tr>
</thead>
<tbody>
<tr>
<td>SI</td>
<td>70317</td>
<td>71076</td>
<td>71494</td>
<td>71697</td>
<td>71796</td>
<td>71899</td>
<td>71931</td>
<td>72014</td>
</tr>
<tr>
<td>IT</td>
<td>32090</td>
<td>31401</td>
<td>31123</td>
<td>30932</td>
<td>30724</td>
<td>30743</td>
<td>30742</td>
<td>30771</td>
</tr>
<tr>
<td>TSRN</td>
<td>30945</td>
<td>30713</td>
<td>30664</td>
<td>30643</td>
<td>30637</td>
<td>30639</td>
<td>30678</td>
<td>30715</td>
</tr>
<tr>
<td></td>
<td>SI</td>
<td>5.503</td>
<td>5.971</td>
<td>7.450</td>
<td>12.540</td>
<td>26.587</td>
<td>108.811</td>
<td>466.51</td>
</tr>
<tr>
<td>CPU</td>
<td>NBJ</td>
<td>2.687</td>
<td>2.865</td>
<td>3.441</td>
<td>6.437</td>
<td>11.501</td>
<td>47.355</td>
<td>203.578</td>
</tr>
<tr>
<td></td>
<td>TSRN</td>
<td>2.392</td>
<td>2.525</td>
<td>3.237</td>
<td>5.608</td>
<td>10.568</td>
<td>46.458</td>
<td>201.795</td>
</tr>
</tbody>
</table>

The NBJ method is also performed in two-step fashion as defined in (3.21) and the vectors are computed in elementwise as:

\[
\begin{align*}
    u_{k+1/2,i} &= \frac{1}{1 - [Pv_{k}]_i}, \\
    v_{k+1/2,i} &= \frac{1}{1 - [Qu_k]_i}, \\
    u_{k+1,i} &= \frac{1}{1 - [Pv_{k+1/2}]_i}, \\
    v_{k+1,i} &= \frac{1}{1 - [Qu_{k+1/2}]_i}, \quad i = 1, 2, \ldots.
\end{align*}
\]

Example 4.1. In Table 1, for the fixed problem size $n = 32$ but different pairs of $(\alpha, c)$, and in Table 2 for the fixed $(\alpha, c) = (10^{-7}, 1)$ but different problem size $n$, we list ITs and CPUs for the SI, NBJ and TSRN methods, respectively.

We see clearly in Table 1 that with fixed problem size, the TSRN method converges faster than the SI and NBJ methods, and as $\alpha$ converges to 0 and $c$ converges to 1, the advantages of the TSRN method are more significant. Moreover, for each pair $(\alpha, c)$, the TSRN method needs less CPU time than the SI and NBJ methods, even though the former needs a little more flops at each iteration. When the parameters $\alpha$ and $c$ are fixed, as the problem size $n$ becomes large, the performance of the TSRN method and the NBJ method becomes close, as shown in Table 2.
Example 4.2. In the end of the work of this paper, we have constructed another two-step relaxation Newton algorithm (denoted as TSRN* for the moment) with the splitting functions \( F \) and \( G \) defined as

\[
F(w_{k+1/2}, w_k) = \left( u_{k+1/2} - u_{k+1/2} \circ [Pv_k] - e \right) / v_{k+1/2} - v_{k+1/2} \circ [Qw_k] - e, \tag{4.5a}
\]

\[
G(w_k, w_{k+1/2}, w_{k+1}) = \left( u_{k+1} - u_{k+1} \circ [Pv_k] - (u_{k+1} \circ [P(v_{k+1/2} - v_k)] + Q(u_{k+1/2} - u_k)) - e \right) / (v_{k+1} - v_{k+1} \circ [Qw_k] - (v_{k+1} \circ [Q(u_{k+1/2} - u_k)]) + P(v_{k+1/2} - v_k)) - e. \tag{4.5b}
\]

With these two splitting functions, we list in Table 3, for fixed problem size \( n = 256 \) but different \((a, c)\), and in Table 4 for fixed \((a, c) = (5 \times 10^{-8}, 1 - 3 \times 10^{-9})\) but different problem size, ITs and CPUs for the NBJ, TSRN and TSRN* methods (from the numerical results given in the previous example we believe that the NBJ and TSRN, methods converge faster than the SI method, and thus at the moment we omit the comparison of the TSRN* method with the SI method).

For each problem size \( n \) and \((a, c)\) tested in Tables 3 and 4, it holds that

\[
\max \{ ||U_{TSRN} - U_{TSRN*}||_\infty, ||V_{TSRN} - V_{TSRN*}||_\infty \} \leq 10^{-14}, \tag{4.6}
\]

where \(U_{TSRN}, V_{TSRN}\) and \(U_{TSRN*}, V_{TSRN*}\) are the converged vectors generated by the TSRN and TSRN* methods, respectively. Therefore, the TSRN* method really converges to the minimal positive solution of the vector equations (1.7). Moreover, from the numerical results listed.

### Table 3: Numerical results for \( n = 256 \) and different \((a, c)\).

<table>
<thead>
<tr>
<th>Method</th>
<th>((0.01, 0.99))</th>
<th>((10^{-6}, 1 - 3 \times 10^{-6}))</th>
<th>((10^{-7}, 1 - 10^{-7}))</th>
<th>((10^{-8}, 1 - 10^{-8}))</th>
<th>((10^{-9}, 1 - 10^{-9}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>NBJ</td>
<td>61</td>
<td>2417</td>
<td>10514</td>
<td>29465</td>
<td>29536</td>
</tr>
<tr>
<td>IT</td>
<td>60</td>
<td>2411</td>
<td>10489</td>
<td>29339</td>
<td>29310</td>
</tr>
<tr>
<td>TSRN*</td>
<td>46</td>
<td>1976</td>
<td>8612</td>
<td>24106</td>
<td>24114</td>
</tr>
<tr>
<td>TSRN</td>
<td>0.109</td>
<td>3.783</td>
<td>16.058</td>
<td>45.401</td>
<td>47.972</td>
</tr>
<tr>
<td>CPU</td>
<td>0.098</td>
<td>3.581</td>
<td>15.121</td>
<td>43.948</td>
<td>44.770</td>
</tr>
<tr>
<td>TSRN*</td>
<td>0.065</td>
<td>2.981</td>
<td>13.242</td>
<td>36.725</td>
<td>36.845</td>
</tr>
</tbody>
</table>

### Table 4: Numerical results for \((a, c) = (5 \times 10^{-8}, 1 - 3 \times 10^{-9})\) and different \( n \).

<table>
<thead>
<tr>
<th>Method</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>384</th>
<th>960</th>
</tr>
</thead>
<tbody>
<tr>
<td>NBJ</td>
<td>28552</td>
<td>27834</td>
<td>27476</td>
<td>27329</td>
<td>27276</td>
<td>27275</td>
<td>27234</td>
<td>27211</td>
</tr>
<tr>
<td>IT</td>
<td>27409</td>
<td>27205</td>
<td>27146</td>
<td>27130</td>
<td>27133</td>
<td>27143</td>
<td>27158</td>
<td>27196</td>
</tr>
<tr>
<td>TSRN*</td>
<td>23071</td>
<td>22659</td>
<td>22466</td>
<td>22363</td>
<td>22333</td>
<td>22235</td>
<td>22232</td>
<td>22342</td>
</tr>
<tr>
<td>TSRN</td>
<td>2.278</td>
<td>2.827</td>
<td>3.107</td>
<td>5.081</td>
<td>12.481</td>
<td>42.595</td>
<td>95.803</td>
<td>480.821</td>
</tr>
<tr>
<td>CPU</td>
<td>2.115</td>
<td>2.447</td>
<td>2.997</td>
<td>4.758</td>
<td>10.921</td>
<td>40.984</td>
<td>92.919</td>
<td>472.829</td>
</tr>
<tr>
<td>TSRN*</td>
<td>1.778</td>
<td>1.971</td>
<td>2.503</td>
<td>3.958</td>
<td>8.750</td>
<td>33.970</td>
<td>76.889</td>
<td>441.720</td>
</tr>
</tbody>
</table>
in Tables 3 and 4, we see clearly that the TSRN∗ method performs much better than the TSRN and NBJ methods. Unfortunately, at the moment we cannot prove theoretically the convergence of the TSRN∗ method to the minimal positive solution of the vector equations (1.7).

Acknowledgments

This work was supported by the NSF of China (no. 10971077) and by the program for NCET, the State Education Ministry of China.

References

