Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2011, Article ID 545470, 12 pages
doi:10.1155/2011/545470

Research Article

A Polynomial Preconditioner for the CMRH Algorithm

Jiangzhou Lai,[1] Linzhang Lu,[1,2] and Shiji Xu[1]

[1] *School of Mathematical Sciences, Xiamen University, Xiamen 361005, China*
[2] *School of Mathematics and Computer Science, Guizhou Normal University, Guiyang 550001, China*

Correspondence should be addressed to Linzhang Lu, lzlu@xmu.edu.cn

Received 4 June 2010; Revised 21 December 2010; Accepted 24 January 2011

Academic Editor: P. Liatsis

Many large and sparse linear systems can be solved efficiently by restarted GMRES and CMRH methods Sadok 1999. The CMRH(m) method is less expensive and requires slightly less storage than GMRES(m). But like GMRES, the restarted CMRH method may not converge. In order to remedy this defect, this paper presents a polynomial preconditioner for CMRH-based algorithm. Numerical experiments are given to show that the polynomial preconditioner is quite simple and easily constructed and the preconditioned CMRH(m) with the polynomial preconditioner has better performance than CMRH(m).

1. Introduction

In many practical applications, we have to solve a large and sparse, nonsymmetric linear system of equations

$$Ax = b, \tag{1.1}$$

where $A \in \mathcal{R}^{n\times n}$ is nonsingular, and $x, b \in \mathcal{R}^n$. The linear system (1.1) can be solved efficiently by iterative methods, especially those based on the Krylov subspace

$$K_k(r_0, A) = \operatorname{span}\left\{r_0, Ar_0, \ldots, A^{k-1}r_0\right\}. \tag{1.2}$$

Although the GMRES method [1] and CMRH method [2] are Krylov-type algorithms, they construct different Krylov subspace bases in different ways; the GMRES method uses

the Arnoldi process, while CMRH uses the Hessenberg process. As the number of iterations increases, the methods become impractical because of growth of memory and computational requirement as k increases, so they must be remedied by restarting; thus, the GMRES(m) and CMRH(m) algorithms were developed. It is concluded in [2] that the CMRH(m) algorithm only requires half the arithmetical operations per iteration and slightly less storage than GMRES(m). However, like GMRES(m), the CMRH(m) method may not converge in some cases. In order to remedy this defect, van Gijzen [3] presented a polynomial preconditioner to improve the GMRES(m) method. The main idea is finding the low-degree polynomials $p_m(x)$ satisfying $p_m(A) \approx A^{-1}$ and transferring the original linear system of (1.1) into

$$p_m(A)Ax = p_m(A)b, \tag{1.3}$$

then applying the iterative method based on the following Krylov subspace:

$$K^k(p_m(A)A; r_0) = \operatorname{span}\left\{r_0, p_m(A)Ar_0, \ldots, \left(p_m(A)A\right)^{k-1} r_0\right\} \tag{1.4}$$

to the new linear system (1.3), obviously, $K^k(p_m(A)A; r_0) \in K^{(k+1)(m-1)+1}(A; r_0)$. This method becomes very effective when large computation of inner product is needed. In this paper, we apply the same technique to the CMRH(m) algorithm. Our method is efficient since the CMRH process itself is utilized to construct the polynomial preconditioner. Numerical experiments are given to show that the CMRH(m) method with a polynomial preconditioner has better performance, in the sense of fewer iterations, than the original CMRH(m) method.

The rest of the paper is arranged as follows. The first part of Section 2 reviews the CMRH(m) method. The construction of the polynomial preconditioner is described in the second part of Section 2, and in the third part, the PCMRH(m) method is developed. We present some numerical experiments to show that the polynomial preconditioner is effective for the CMRH(m) method in Section 3. Section 4 is a simple conclusion.

Throughout the paper, all vectors and matrices are assumed to be real. For a vector v, $\|v\|$ denotes the Euclidean norm $\|v\| = \sqrt{v^T v}$, and $\|v\|_\infty$ denotes the maximum norm $\|v\|_\infty = \max_{i=1,\ldots n}|v_i|$, where v_i is the ith component of a vector v, and $|v_i|$ denotes the absolute value(modulus) of a real(complex) number v_i. For a matrix A, $\|A\|$ denotes the 2-norm. Furthermore, we denote by $e_k^{(n)}$ the kth canonical vector of R^n,

$$e_k^{(n)} = (0, \ldots, 1, 0, \ldots, 0)^T, \tag{1.5}$$

and $I_k^{(n)}$ the first kth columns of an n-dimensional identity matrix, and we denote by $H^\dagger$ the pseudoinverse of a matrix H.

2. PCMRH(m) Algorithm

2.1. CMRH(m) Algorithm

The principle of CMRH [2] method is to use the Hessenberg process with pivoting to construct a basis of the Krylov subspace $K_k(A, r_0)$: $l_1, l_2, \ldots, l_k$. Precisely, the Hessenberg

process produces an $n \times k$ unit lower trapezoidal matrix $L_k = (l_1, \ldots, l_k)$ and a $(k+1) \times k$ upper Hessenberg matrix $\overline{H}_k \in \mathcal{R}^{(k+1)\times k}$, such that

$$AL_k = L_{k+1}\overline{H}_k. \tag{2.1}$$

Then, we find a correction vector $z \in K_k(A, r_0)$, such that

$$\min_{z\in K_k(A,r_0)} \|b - A(x_0 + z)\| = \min_{z\in K_k(A,r_0)} \|r_0 - Az\|. \tag{2.2}$$

Since $K_k(A, r_0) = \operatorname{span}\{l_1, l_2, \ldots, l_k\}$, we let $z = L_k y_k$ and have

$$\begin{aligned} \min_{z\in K_k(A,r_0)} \|b - A(x_0 + z_k)\| &= \min_{y_k\in R^k} \left\|r_0 - AL_k y_k\right\| \\ &= \min_{y_k\in R^k} \left\|r_0 - L_{k+1}\overline{H}_k y_k\right\| \\ &= \min_{y_k\in R^k} \left\|\left\|r_0\right\| e_1^{(k+1)} - \overline{H}_k y_k\right\|. \end{aligned} \tag{2.3}$$

Therefore, a least-squares solution could be obtained from the Krylov subspace by minimizing $\|\|r_0\|e_1^{(k+1)} - \overline{H}_k y_k\|$ over y, just as in the GMRES method, except that the Hessenberg process is replaced by the Arnoldi process; this is the basic idea of the CMRH method. Since the Hessenberg process will be breakdown if $(l_j)_j$ is zero, Sadok [2] uses a pivoting strategy such as in the Gaussian elimination method to avoid such a breakdown, and then the approximating solution can be computing by

$$x_k = x_0 + L_k H^{\dagger}(r_0)_{i_0} e_1^{k+1}, \tag{2.4}$$

where i_0 satisfies $|(r_0)_{i_0}| = \|r_0\|_\infty$, $L_k = (l_1, \ldots, l_k)$ is an $n \times k$ unit lower trapezoidal matrix, and $H^{\dagger}$ denotes the pseudoinverse of a matrix H for more details, see (Algorithm 1) [2].

The notation "$\leftrightarrow$" means as "swap contents": $\alpha \leftrightarrow \beta \Leftrightarrow \gamma = \alpha; \alpha = \beta; \beta = \gamma$. As k increases, the CMRH method(Hessenberg process with pivot) becomes impractical because of the growth of memory and computational requirement, so it must be remedied by restarting, thus the CMRH(m) algorithm is developed, which is described in (Algorithm 2) [2].

The main body of Algorithm 2 is the Hessenberg process with pivoting [2], in the last, we get $x_m = x_0 + L_m y_m$, where $y_m \in \mathcal{R}^m$ minimizes $\|\overline{H}_m y - (r_0)_{i_0} e_1^{(m+1)}\|$; this is a little different from the GMRES method because of the pivot strategy used in the Hessenberg process.

2.2. Construction of the Polynomial Preconditioner

The main idea of the polynomial preconditioned method is to construct a polynomial $s(t)$ satisfying $s(A) \approx A^{-1}$ and then solve the linear system of equations $s(A)Ax = s(A)b$ instead of solving the linear system (1.1). Generally, we do not need to compute $s(A)A$ really, since what we need is only that the matrix $s(A)A$ approaches the unit matrix. In practice, if

set $p = (1, \ldots, n)^T$.
Determine i_0 so $|(v)_{i_0}| = \|v\|_\infty$; $l_1 = \dfrac{v_0}{(v)_{i_0}}$; $p(1) \leftrightarrow p(i_0)$ (interchange $p(1)$ and $p(i_0)$);
iterate:
for $k = 1, 2, \ldots, n$
  $u = Al_k$
  for $j = 1, 2, \ldots, k$
    $c = u_{p(j)}$; $h_{j,k} = c$; $u_{p(j)} = 0$
    for $l = j + 1, \ldots, n$
      $u_{p(l)} = u_{p(l)} - c * (l_j)_{p(l)}$
    end
  end
  if $k < n$
    determine i_0 so $|(u)_{i_0}| = \|u\|_\infty$; $p(k+1) \leftrightarrow p(i_0)$
    $h_{k+1,k} = u_{i_0}$; $l_{k+1} = u/(u)_{i_0}$
  end
end

Algorithm 1: Hessenberg process with pivoting [2].

a polynomial q_n is constructed, such that $x_{k+1} = x_0 + q_k(A)r_0$ is an approximation of the solution of the linear system (1.1), then when $x_0 = 0$, the following relationship holds:

$$A^{-1}b \approx x_{k+1} = q_k(A)b. \tag{2.5}$$

That is, $q_k(A)$ is a polynomial preconditioner.

In the following, we utilize the residual polynomial produced from the CMRH process to construct a polynomial preconditioner.

Let K_k be an $n \times k$ matrix formed by Krylov vectors, that is,

$$K_k = \left(r_0, Ar_0, \ldots, A^{k-1}r_0\right). \tag{2.6}$$

From (2.1), we get

$$l_{k+1} = h_{k+1,k}^{-1}(Al_k - L_k h_k), \tag{2.7}$$

where $h_k = (h_{1k}, \ldots, h_{kk})^T$ is the kth column of $\overline{H}_k$. Since the columns of L_k span the same space as the columns of K_k, the following relationship holds:

$$L_k = K_k C_k, \tag{2.8}$$

Input: m: the dimension of a Krylov subspace and the approximation precision ε
start: $x_0 = 0, r_0 = b - Ax_0$, and set $p(i) = i, i = 1, \ldots, n$
Determine i_0 *so* $|(r_0)_{i_0}| = \|r_0\|_\infty$; $l_1 = \dfrac{r_0}{(r_0)_{i_0}}$; $p(1) \leftrightarrow p(i_0)$ (interchange $p(1)$ and $p(i_0)$);
iterate:
for $k = 1, 2, \ldots, m$
  $u = Al_k$
  for $j = 1, 2, \ldots, k$
    $c = u_{p(j)}$; $h_{j,k} = c$; $u_{p(j)} = 0$
    for $l = j + 1, \ldots, n$
      $u_{p(l)} = u_{p(l)} - c * (l_j)_{p(l)}$
    end
  end
  if $k < n$
    determine i_0 *so* $|(u_0)_{i_0}| = \|u_0\|_\infty$; $p(k+1) \leftrightarrow p(i_0)$
    $h_{k+1,k} = u_{i_0}$; $l_{k+1} = u/(u)_{i_0}$
  end
  if (an estimate of) $\|b - Ax_k\|$ is small enough *or* $k = n$ then
    $x_k = x_0 + L_k y_k$, where y_k minimize $\|\overline{H}_k y - (r_0)_{i_0} e_1^{(k+1)}\|$
    stop iteration
  end
end
$x_m = x_0 + L_m y_m$, where $y_m \in \mathcal{R}^m$ minimizes $\|\overline{H}_m y - (r_0)_{i_0} e_1^{(m+1)}\|$.
$x_0 := x_m$, go to start

Algorithm 2: CMRH(m) [2].

where C_k is an upper triangular matrix. Thus,

$$L_k h_k = K_k C_k h_k = \begin{pmatrix} K_k & A^k r_0 \end{pmatrix} \begin{pmatrix} C_k h_k \\ 0 \end{pmatrix},$$

$$Al_k = AK_k \begin{pmatrix} c_{1k} \\ \vdots \\ c_{kk} \end{pmatrix} = \begin{pmatrix} Ar_0 & A^2 r_0 & \cdots & A^k r_0 \end{pmatrix} \begin{pmatrix} c_{1k} \\ \vdots \\ c_{kk} \end{pmatrix} = K_{k+1} \begin{pmatrix} 0 \\ c_{1k} \\ \vdots \\ c_{kk} \end{pmatrix}. \tag{2.9}$$

Hence, from (2.7), we have

$$l_{k+1} = h_{k+1,k}^{-1} K_{k+1} \left(\begin{pmatrix} 0 \\ c_{1k} \\ \vdots \\ c_{kk} \end{pmatrix} - \begin{pmatrix} C_k h_k \\ 0 \end{pmatrix} \right). \tag{2.10}$$

On the other hand, from $L_{k+1} = K_{k+1}C_{k+1}$, we have $l_{k+1} = K_{k+1}\begin{pmatrix} c_{1,k+1} \\ \vdots \\ c_{k+1,k+1} \end{pmatrix}$. Since the columns of K_{k+1} are linear independent, we obtain

$$\begin{pmatrix} c_{1,k+1} \\ c_{2,k+1} \\ \vdots \\ c_{k+1,k+1} \end{pmatrix} = h_{k+1,k}^{-1} \begin{pmatrix} 0 \\ c_{1k} \\ \vdots \\ c_{kk} \end{pmatrix} - h_{k+1,k}^{-1} \begin{pmatrix} C_k h_k \\ 0 \end{pmatrix}. \tag{2.11}$$

Thus, from the CMRH algorithm, we can obtain the approximate solution

$$x_{k+1} = x_0 + L_{k+1}y = x_0 + K_{k+1}C_{k+1}y. \tag{2.12}$$

Since $L_{k+1}y = x_0 + K_{k+1}C_{k+1}y$, we know that the entries of $C_{k+1}y$ are the coefficients of $q_k(z)$, that is, if $C_{k+1}y = (\alpha_0, \alpha_1, \ldots, \alpha_k)^T$, then $q_k(z) = \alpha_0 + \alpha_1 z + \cdots + \alpha_k z^k$. Note that $p_{k+1}(z) = 1 - zq_k(z)$ is the residual polynomial, so we have $q_k(z) = (1 - p_{k+1}(z))/z$ and $q_k(A) \approx A^{-1}$ [4]; therefore, $q_k(A)$ becomes a preconditioner.

In the next subsection, we present the preconditioned CMRH(m) with the polynomial preconditioner described above.

2.3. PCMRH(m) Algorithm

The polynomial preconditioned CMRH(m) (PCMRH(m)) method is given in the following, which is composed of two parts, in the first part, the polynomial preconditioner is constructed, and in the second part, the CMRH(m) method is applied to the new linear systems (Algorithm 3).

3. Numerical Experiments

In this section, we give some numerical examples to compare the performance of PCMRH(m) with CMRH(m). All codes were written in Matlab 6.5 and run on AMD Athlon(tm) 64 X2 Dual Core Processor 2.20 GHz equipped with 1.5 G of RAM. In these numerical experiments, all the matrices are from Sadok's paper [2] we always take the initial vector $x_0 = (0, 0, \ldots, 0)^T$, $b = (1, 1, \ldots, 1)^T$, and $m = 20$. We use the inequality $\text{norm}(r)/\text{norm}(b) \leq 10^{-10}$ as stopping criteria. In the figures, solid lines(-) denote the CMRH(m) method, and dots(-.) denote the PCMRH(m) method. The x-axis denotes the number of iterations, and y-axis denotes the norm of the residual r: $\text{norm}(r)$.

Input: the dimension of Krylov subspace m; the prescribed stopping criterion ε;
the degree of the polynomial preconditioner kk; the initial approximate solution x_0;
the residual vector $r_0 = b - Ax_0$.

1. The procedure of constructing polynomial preconditioner q_{kk}

(1) Let $p(i) = i,\ i = 1, 2, \ldots, n$;
determine i_0 *so* $|(r_0)|_{i_0} = \|r_0\|_\infty$; $l_1 = \dfrac{r_0}{(r_0)_{i_0}}$; $p(1) \leftrightarrow p(i_0)$,
$C(1,1) = \dfrac{1}{(r_0)_{i_0}}$

(2) for $k = 1, 2, \ldots, kk$ *do*
$u = Al_k$
for $j = 1, 2, \ldots, k$ *do*
$c = u_{p(j)}, h_{j,k} = c, u_{p(j)} = 0$
for $l = j + 1, \ldots, n$ *do*
$u_{p(l)} = u_{p(l)} - c \times (l_j)_{p(l)}$
end
if $k < n$
compute i_0 *so* $|(u_0)|_{i_0} = \|u_0\|_\infty$; $p(k+1) \leftrightarrow p(i_0)$
$h_{k+1,k} = u_{i_0}$; $l_{k+1} = \dfrac{u}{(u)_{i_0}}$
let $h_j = \begin{pmatrix} h_{1,j} \\ \vdots \\ h_{j,j} \end{pmatrix}$, $C_j = \begin{pmatrix} c_{11} & \cdots & c_{1j} \\ & \ddots & \vdots \\ 0 & 0 & c_{jj} \end{pmatrix}$
compute
$$\begin{pmatrix} c_{1,j+1} \\ c_{2,j+1} \\ \vdots \\ c_{j+1,j+1} \end{pmatrix} = h_{j+1,j}^{-1} \begin{pmatrix} 0 \\ c_{1j} \\ \vdots \\ c_{jj} \end{pmatrix} - h_{j+1,j}^{-1} \begin{pmatrix} C_j h_j \\ 0 \end{pmatrix}$$
end
end

(3) Form the approximate solution $x_k = x_0 + L_k y_k$, where $y_k \in C^k$ *minimizes*
$$F(y) = \|\overline{H}_k y - (r_0)_{i_0} e_1\|.$$

(4) Form the polynomial preconditioner $q_{kk}(z)$
$$C_k y_k = (\alpha_0 \cdots \alpha_{k-1})^T;$$
$$q_{kk}(x) = \alpha_0 + \alpha_1 x + \cdots + \alpha_{k-1} x^{k-1}.$$

2. Using the CMRH(m) algorithm for $q_{kk}(A)Ax = q_{kk}(A)b$.

Algorithm 3: PCMRH(m).

Example 3.1. The first example is taken from Brown [5]. The $n \times n$ matrix

$$A = \begin{pmatrix} \varepsilon & 1 & & & \\ -1 & \varepsilon & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & \varepsilon & 1 \\ & & & -1 & \varepsilon \end{pmatrix} \tag{3.1}$$

Table 1

kk	No. of restarts	Time (sec)	Residual norm	Convergence
1	1000	2.7500	0.0011	No
2	171	0.4690	$9.4299e-11$	Yes
3	289	0.7810	$9.4596e-11$	Yes
4	177	0.4840	$9.9338e-11$	Yes
5	236	0.6560	$9.6960e-11$	Yes
6	50	0.1250	$9.6988e-11$	Yes
7	72	0.2030	$6.8890e-11$	Yes
8	60	0.1720	$8.5579e-11$	Yes
9	75	0.2190	$7.7776e-11$	Yes
10	34	0.1090	$5.8616e-11$	Yes
11	40	0.1250	$5.5309e-11$	Yes
12	32	0.1090	$6.1764e-11$	Yes
13	28	0.0780	$8.4069e-11$	Yes
14	23	0.0780	$3.6570e-11$	Yes
15	23	0.0940	$7.6101e-11$	Yes
16	17	0.0780	$4.8057e-11$	Yes
17	18	0.0780	$8.7496e-12$	Yes
18	11	0.0620	$7.8930e-12$	Yes
19	7	0.0470	$3.5654e-12$	Yes
20	14	0.0620	$1.3220e-11$	Yes

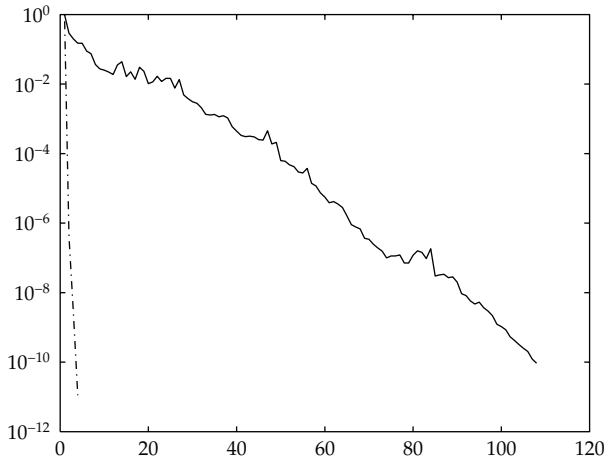

Figure 1: $\varepsilon = 0.1$, $n = 40$.

is used in three experiments, whose results are listed in Figures 1 and 2 and Table 1, respectively. In Figure 1, $\varepsilon = 0.1$, $n = 40$, and the degree of the preconditioned polynomial: $kk = 20$. In Figure 2, $\varepsilon = 0.01$, $n = 40$, and $kk = 20$. In Table 1, $\varepsilon = 0.01$, $n = 100$ and kk ranges from 1 to 20.

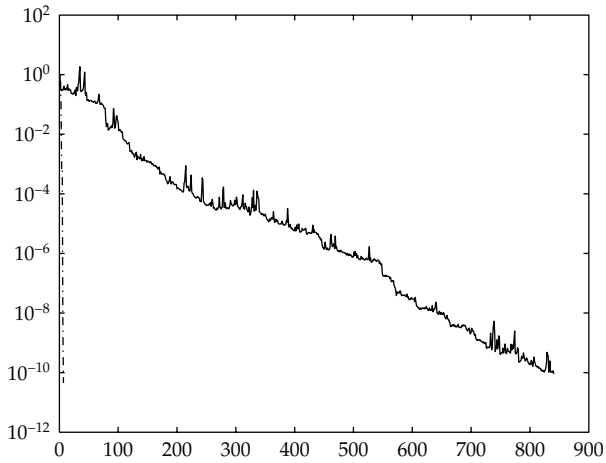

Figure 2: $\varepsilon = 0.01$, $n = 40$.

From Figures 1 and 2, we can see that the PCMRH(m) method converges much faster than CMRH(m), when $\varepsilon = 0.1$, $n = 40$, the CMRH(20) method needs 107 restarts to make sure that the residual norm is $9.2925e-011$, while PCMRH(20) can get $1.1143e-011$ only in 3 restarts. When $\varepsilon = 0.01$, $n = 40$, the CMRH(20) method needs 840 restarts to make sure that the residual norm is $9.0075e-011$, while PCMRH(20) can get $4.4771e-011$ only by 6 restarts. The polynomial preconditioner is well done.

Let kk range from 1 to 20 and list PCMRH(20) in Table 1. We see that only when $kk = 1$, the PCMRH(20) method does not converge, and when $kk = 19$, PCMRH(20) has the best performance.

Example 3.2. The matrix

$$A = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 & 1 \\ a_1 & 1 & 1 & \dots & 1 & 1 \\ a_1 & a_2 & 1 & \dots & 1 & 1 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ a_1 & a_2 & a_3 & \dots & a_{n-1} & 1 \end{pmatrix} \quad \text{with } a_i = 1 + i\varepsilon \tag{3.2}$$

is taken from [6]. In this experiment, we let $\varepsilon = 10^{-2}$, $n = 100$, and $kk = 2$.

From Figure 3, we can also see that the PCMRH(20) method converges much faster than CMRH(20). The CMRH(20) method needs 317 restarts to make sure the residual norm is $9.9221e-011$, while PCMRH(20) reaches $9.0007e-011$ only in 37 restarts. So, we can conclude that the polynomial preconditioner makes CMRH(m) algorithm more efficient.

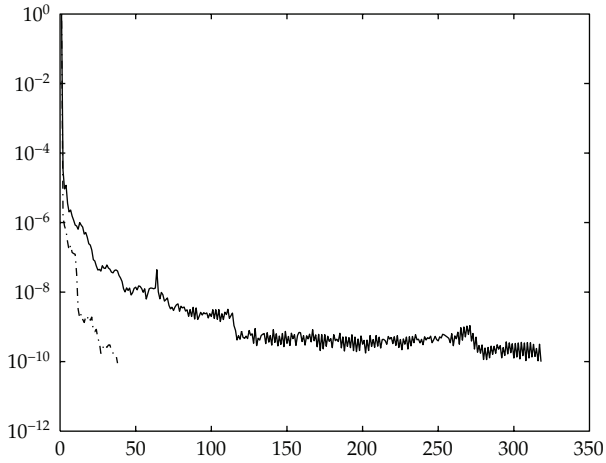

Figure 3: $\varepsilon = 10^{-2}$, $n = 100$.

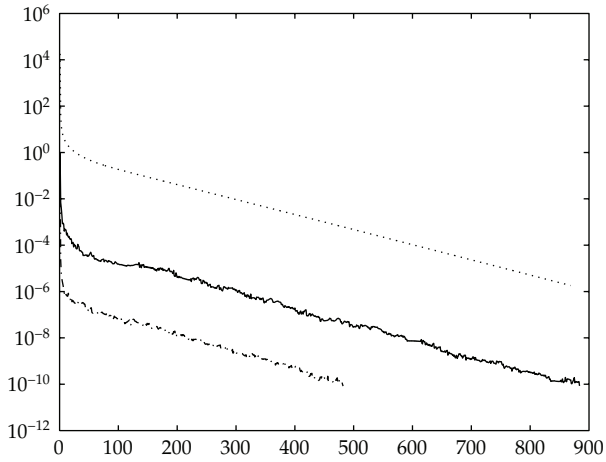

Figure 4: $A = SDS^{-1}$.

Example 3.3. Let $A = SDS^{-1} \in \mathcal{R}^{1000\times 1000}$, where S is a bidiagonal matrix of the form

$$S = \begin{pmatrix} 1 & 0.9 & & & \\ & 1 & 0.9 & & \\ & & \ddots & \ddots & \\ & 0 & & \ddots & 0.9 \\ & & & & 1 \end{pmatrix} \in \mathcal{R}^{1000\times 1000}, \tag{3.3}$$

and $D = \operatorname{diag}(-10, -9, \ldots, -1, 1, 2, \ldots, 990) \in \mathcal{R}^{1000\times 1000}$.

In this experiment, we compare the PCMRH(m) method with CMRH(m) and GMRES(m). In Figure 4, we use the dot line to denote the GMRES(m) method.

Table 2

Algorithm	No. of restarts	Time (sec)	Residual norm	Convergence
GMRES(20)	1000	368.0310	$1.8110e-6$	No
CMRH(20)	883	100.2500	$8.4761e-11$	Yes
PCMRH(20) ($kk = 3$)	481	57.2650	$8.2944e-11$	Yes

Table 3

kk	No. of restarts	Time (sec)	Residual norm	Convergence
1	518	60.4840	$9.4131e-11$	Yes
2	499	59.3440	$9.6438e-11$	Yes
3	481	57.2650	$8.2944e-11$	Yes
4	625	75.1720	$9.7851e-11$	Yes
5	639	77.7970	$9.6175e-11$	Yes
6	934	113.6710	$9.7319e-11$	Yes
7	680	84.7190	$9.9634e-11$	Yes
8	649	82.4840	$9.8637e-11$	Yes
9	747	94.9220	$9.3972e-11$	Yes
10	824	104.3750	$6.7137e-11$	Yes
11	940	118.8590	$8.5893e-11$	Yes
12	585	79.5790	$9.8813e-11$	Yes
13	690	93.1400	$8.4271e-11$	Yes
14	640	89.2820	$9.6628e-11$	Yes
15	599	84.9530	$9.8641e-11$	Yes
16	755	104.5470	$9.8972e-11$	Yes
17	553	81.8440	$9.0119e-11$	Yes
18	730	104.3910	$9.3932e-11$	Yes
19	611	90.2970	$8.6073e-11$	Yes
20	660	97.0460	$9.8196e-11$	Yes

It can be seen from Figure 4 that the GMRES(20) method is stagnate since some eigenvalues of A are negative; it uses 1000 restarts or about 368 seconds to make the residual norm reach $1.8110e-006$. Meanwhile, the CMRH(20) method uses 883 restarts or about 100 seconds to get the residual norm be $8.4761e-011$; however, the PCMRH(20) method with $kk = 3$ can reach $8.2944e-011$ only by 481 restarts or about 57 seconds. These results are also listed in Table 2 and show that the polynomial preconditioner is effective.

Table 3 lists the performance of PCMRH(20). It is seen that the PCMRH(20) method is convergent when the value of kk ranges from 1 to 20, but it can not conclude that the number of restarts and the time needed by PCMRH(20) decrease(increase) as kk increases(decreases). In fact, when $kk = 3$, the PCMRH(20) method performs the best.

4. Conclusion

To accelerate the convergence, we take advantage of the CMRH process to construct a polynomial preconditioner for CMRH(m) algorithm. Numerical experiments show that the preconditioned CMRH(m) algorithm is more efficient than the CMRH(m) algorithm

without the polynomial preconditioner. However, the degree of the polynomial used as preconditioner should be small, otherwise it possibly reduces the preconditioner's effectiveness.

Acknowledgment

This work is supported by the National Natural Science Foundation of China no. 10961010.

References

[1] Y. Saad and M. H. Schultz, "GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on Scientific and Statistical Computing*, vol. 7, no. 3, pp. 856–869, 1986.

[2] H. Sadok, "CMRH: a new method for solving nonsymmetric linear systems based on the Hessenberg reduction algorithm," *Numerical Algorithms*, vol. 20, no. 4, pp. 303–321, 1999.

[3] M. B. van Gijzen, "A polynomial preconditioner for the GMRES algorithm," *Journal of Computational and Applied Mathematics*, vol. 59, no. 1, pp. 91–107, 1995.

[4] P. E. Saylor and D. C. Smolarski, "Implementation of an adaptive algorithm for Richardson's method," *Linear Algebra and its Applications*, vol. 154–156, pp. 615–646, 1991.

[5] P. N. Brown, "A theoretical comparison of the Arnoldi and GMRES algorithms," *SIAM Journal on Scientific and Statistical Computing*, vol. 12, no. 1, pp. 58–78, 1991.

[6] R. T. Gregory and D. L. Karney, *A Collection of Matrices for Testing Computational Algorithms*, John Wiley & Sons, New York, NY, USA, 1969.