Journal of Computational Mathematics
Vol.40, No.5, 2022, 715–731.

http://www.global-sci.org/jcm
doi:10.4208/jcm.2101-m2019-0243

HEAVY BALL FLEXIBLE GMRES METHOD FOR NONSYMMETRIC LINEAR SYSTEMS*

Mei Yang[1)]
School of Mathematical Sciences, Shanghai Jiao Tong University, Shanghai 200240, China
Email: ymsjtu@sjtu.edu.cn
Ren-Cang Li
Department of Mathematics, University of Texas at Arlington, Arlington, TX, USA
Email: rcli@uta.edu

Abstract

Flexible GMRES (FGMRES) is a variant of preconditioned GMRES, which changes preconditioners at every Arnoldi step. GMRES often has to be restarted in order to save storage and reduce orthogonalization cost in the Arnoldi process. Like restarted GMRES, FGMRES may also have to be restarted for the same reason. A major disadvantage of restarting is the loss of convergence speed. In this paper, we present a heavy ball flexible GMRES method, aiming to recoup some of the loss in convergence speed in the restarted flexible GMRES while keep the benefit of limiting memory usage and controlling orthogonalization cost. Numerical tests often demonstrate superior performance of the proposed heavy ball FGMRES to the restarted FGMRES.

Mathematics subject classification: 65F10.
Key words: GMRES, Flexible GMRES, Heavy ball GMRES, Preconditioner, Linear system.

1. Introduction

The Generalized Minimal Residual (GMRES) method [12] is a well-established Krylov subspace method for solving a large and sparse nonsymmetric linear system of equations

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

where $A \in \mathbb{C}^{n\times n}, b \in \mathbb{C}^n$, and $x \in \mathbb{C}^n$ is the unknown. Given an initial approximation x_0, the k-th GMRES approximation x_k is sought so that the k-th residual $r_k = b - Ax_k$ satisfies

$$\|r_k\|_2 = \min_{z\in\mathcal{K}_k(A,r_0)} \|b - A(x_0 + z)\|_2, \tag{1.2}$$

where $r_0 = b - Ax_0$, $\|\cdot\|_2$ is the Euclidian norm, and $\mathcal{K}_k(A, r_0)$ is the k-th Krylov subspace of A on r_0 defined by

$$\mathcal{K}_k(A, r_0) = \text{span}\left\{r_0, Ar_0, A^2r_0, \cdots, A^{k-1}r_0\right\}. \tag{1.3}$$

Algorithmically, GMRES first builds an orthonormal basis of $\mathcal{K}_k(A, r_0)$ via the Arnoldi process and, along the way, A is projected onto the Krylov subspace to turn (1.2) into a much smaller $(k + 1) \times k$ least squares problem.

* Received October 30, 2019 / Revised version received June 12, 2020 / Accepted January 19, 2021 / Published online March 8, 2021 /
1) Corresponding author

Ideally, it is hoped that for a modest k relative to n, $\|r_k\|_2$ is sufficient tiny so that x_k can be regarded as a sufficiently accurate approximation to the exact solution of (1.1). But that is not always the case. When that happens, GMRES can become very expensive because of the heavy burden in memory for storing orthonormal basis vectors and for generating them in the Arnoldi process. A popular and the simplest remedy is the so-called restarted GMRES (REGMRES) [6] which sets an upper bound $k_{\max}$ on k and starts over as soon as k reaches the upper bound $k_{\max}$ but $\|r_{k_{\max}}\|_2$ is not yet tiny enough, using the latest approximation $x_{k_{\max}}$ as the initial guess for the next GMRES run. Doing so effectively put the memory requirement and orthogonalization cost under control, but not without tradeoff, which is slow convergence and potentially increases overall computational time in solving (1.1). In recognizing this tradeoff, researchers have made efforts address the issue.

The loss in convergence speed by REGMRES is due to its control on the largest possible number of Arnoldi steps that one GMRES run can use. Besides the use of the latest approximation as the initial guess for the next GMRES run, REGMRES completely throws away the Krylov subspaces built thus far. To partially compensate the throw-away, two main types of improvements are discussed, which include hybrid iterative methods and acceleration techniques. Our discussion mainly focuses on acceleration techniques. One natural idea to accelerate GMRES is to augment the Krylov subspace of REGMRES according to spectral information at the restart, named augmented Krylov subspace techniques, such as GMRES-E [8] , GMRES-IR [9] and GMRES-DR [10]. This kind of methods keeps the form of Krylov subspace. Another technique is to approximate the search space with non-Krylov subspace, i.e., approximation space. In [1], Baker, Jessup and Manteuffel presented the Loose GMRES (LGMRES) method. At the ℓ-th restart of LGMRES, the Krylov subspace $\mathcal{K}_k(A, r_0^{(\ell)})$ is generated and augmented with the t most recent error vectors, which are defined to be the differences between every two sequential solutions.

In [7], from the optimization perspective, Imakura, Li and Zhang proposed two comparable methods, the locally optimal GMRES (LOGMRES) and the heavy ball GMRES methods (HBGMRES). LOGMRES augments the search space by adding the most previous solution vector. The latter one incorporates the idea in the heavy ball method from optimization [11, p.65] into REGMRES by adding a new vector – the difference of approximations from the previous two cycles of HBGMRES, i.e., the most previous error vector in LGMRES, to the next searching space. Numerical experiments show that HBGMRES often converges significantly faster than REGRMES. However, there are cases where the gain of HBGMRES over REGMRES is not so significant.

The preconditioning technique is often very effective in enhancing the performance and reliability of Krylov subspace methods, provided a reasonably good preconditioner can be found. Instead of (1.1), its right preconditioned linear system takes the form

$$(AM^{-1})y = b, \quad Mx = y. \tag{1.4}$$

The matrix M is called a preconditioner and it may exist in form in such a way that linear system $Mz = c$ is cheap to solve. However, it is usually hard to find a suitable preconditioner M for the problem at hand. Saad [13] proposed an inner-outer iteration method called the flexible GMRES method (FGMRES), in which GMRES is used as the outer iteration. In the inner iteration some linear system $Az = c$ is approximately solved and thus each inner iteration can be viewed as applying some preconditioner M, not known explicitly but implicitly in the action $M^{-1}c \approx A^{-1}c$. GMRESR [15], another similar inner-outer iteration, uses GCR [3] instead

of GMRES as its outer iteration. If using the same solver for the inner iteration, FGMRES and GMRESR often yield solutions of comparable accuracy [16]. As it's pointed by Fokkema, Sleigpen, and van der Vorst in [4] that the distinction between preconditioning and acceleration is not so clear. Both FGMRES and GMRESR can also be seen as the approximation space technique mentioned before.

As k increases, orthogonalization cost and memory for FGMRES usage also increase for computing and storing basis vectors. For that reason, FGMRES may also need to be restarted for difficult linear systems, too. Just like REGMRES, restarting FGMRESS can cause loss in convergence speed. In this paper, we design a heavy ball FGMRES which introduces the idea of the heavy ball method to the restarted FGMRES. Numerical tests show that it's able to salvage the lost convergence speed while still keep the benefit of the restarted FGMRES in limiting memory usage and controlling orthogonalization cost.

The rest of this paper is organized as follows. Section 2 outlines GMRES, flexible GMRES and the heavy ball GMRES method. Section 3 presents the framework of the heavy ball FGMRES method and its implementation details. Section 4 presents numerical results on eight test problems from the SuiteSparse Matrix Collection and Matrix Market. Finally, we give our concluding remarks in Section 5.

Notation. $\mathbb{C}^{n\times m}$ is the set of all $n\times m$ complex matrices, $\mathbb{C}^n = \mathbb{C}^{n\times 1}$. I_n or I (if its size is clear from the context) is the $n\times n$ identity matrix, and e_j is its j-th column. The superscript "H" takes the conjugate transpose of a matrix or vector. Denote by $i:j$ the set of integers from i to j inclusive. For a vector u and a matrix X, $u_{(i)}$ is u's i-th entry, $X_{(i,j)}$ is X's (i,j)-th entry; X's submatrices $X_{(k:\ell,i:j)}$, $X_{(k:\ell,:)}$ and $X_{(:,i:j)}$ consist of intersections of row k to row ℓ and column i to column j, row k to row ℓ, and column i to column j, respectively, and

$$\|u\|_2 = \sqrt{\sum_i |u_{(i)}|^2}, \quad \|X\|_2 := \max_{u\neq 0} \frac{\|Xu\|_2}{\|u\|_2}.$$

2. Preliminaries

2.1. GMRES

Given an initial guess x_0, the corresponding residual is $r_0 = b - Ax_0$. The k-step GMRES method first builds a basis matrix $V_{k+1} = [v_1, v_2, \ldots, v_{k+1}]$ of the Krylov subspace $\mathcal{K}_{k+1}(A, r_0)$ via the Arnoldi process, and then seeks the best solution x_k that minimizes $\|b - Ax\|_2$ over $x_0 + \mathcal{K}_k(A, r_0)$. Specifically,

$$x_k = x_0 + \operatorname*{argmin}_{z\in\mathcal{K}_k(A,r_0)} \|r_0 - Az\|_2.$$

When with a right preconditioner M as in (1.4), the associated Arnoldi process is for AM^{-1} on r_0, and the resulting preconditioned GMRES is outlined in Algorithm 2.1, which will reduce to the plain GMRES upon simply setting $M = I$. In the generic situation, i.e., **break** at Line 12 is not invoked, the Arnoldi process in Algorithm 2.1 can be compactly written as

$$AZ_k = AM^{-1}V_k = V_{k+1}\check{H}_k,$$

where $\check{H}_k$ is the matrix $\check{H}$ defined in Algorithm 2.1 and $Z_k = [z_1, z_2, \ldots, z_k]$ also defined there. The approximate solution x_k returned by the algorithm is the best one from $x_0 + \operatorname{span}\{Z_k\} =$

$x_0 + \mathcal{K}_k(AM^{-1}, r_0)$ in the sense that

$$x_k = x_0 + \underset{z \in \mathrm{span}\{Z_k\}}{\mathrm{argmin}} \|r_0 - Az\|_2. \tag{2.1}$$

Any $z \in \mathrm{span}\{M^{-1}V_k\}$ can be written as $z = M^{-1}V_k y = Z_k y$. Therefore,

$$r_0 - Az = r_0 - AZ_k y = \beta V_{k+1} e_1 - V_{k+1}\check{H}_k y = V_{k+1}(\beta e_1 - \check{H}_k y),$$

yielding $\|r_0 - Az\|_2 = \|\beta e_1 - \check{H}_k y\|_2$, which combined with (2.1) lead to

$$x_k = x_0 + Z_k y_k \quad \text{with} \quad y_k = \underset{y \in \mathbb{C}^k}{\mathrm{argmin}} \|\beta e_1 - \check{H}_k y\|_2. \tag{2.2}$$

That explains Lines 17 and 18 of Algorithm 2.1. In what follows, by the k-step GMRES we mean Algorithm 2.1 with $M = I$. REGMRES(k) is the algorithm of repeatedly running the k-step GMRES with the initial guess of the current k-step GMRES being the computed solution of the very previous k-step GMRES, assuming the initial guess for the first k-step GMRES run is given. Each k-step GMRES run is termed a *cycle*.

Algorithm 2.1. k-step Preconditioned GMRES for $Ax = b$

Input: $A \in \mathbb{C}^{n\times n}$, preconditioner $M \in \mathbb{C}^{n\times n}$, $x_0 \in \mathbb{C}^n$, integer k;
Output: An approximate solution x_k to $Ax = b$.
1: $r_0 = b - Ax_0 \in \mathbb{C}^n$, $\beta = \|r_0\|_2$ and $v_1 = r_0/\beta$;
2: $V_{(:,1)} = v_1$, $\check{H} = 0_{(k+1)\times k}$ (the $(k+1) \times k$ zero matrix);
3: **for** $j = 1, 2, \ldots, k$ **do**
4: $\quad z_j = M^{-1}V_{(:,j)}$;
5: $\quad f = Az_j$;
6: $\quad$ **for** $i = 1, 2, \ldots, j$ **do**
7: $\quad\quad \check{H}_{(i,j)} = V_{(:,i)}^{\mathrm{H}} f, f = f - V_{(:,i)}\check{H}_{(i,j)}$
8: $\quad$ **end for**
9: $\quad \check{H}_{(j+1,j)} = \|f\|_2$;
10: $\quad$ **if** $\check{H}_{(j+1,j)} > 0$ **then**
11: $\quad\quad V_{(:,j+1)} = f/\check{H}_{(j+1,j)}$;
12: $\quad$ **else**
13: $\quad\quad$ reset $k = j$, $\check{H} = \check{H}_{(1:j,1:j)}$;
14: $\quad\quad$ **break**;
15: $\quad$ **end if**
16: **end for**
17: compute $y_k = \underset{y}{\mathrm{argmin}} \|\beta e_1 - \check{H}y\|_2$;
18: **return** $x_k = x_0 + Z_k y_k$, where $Z_k = [z_1, \ldots, z_k]$.

2.2. Flexible GMRES

Let us now focus on Line 4 of Algorithm 2.1 on the preconditioner M. In practice, it may be possible that M^{-1} is explicitly constructed but more often M^{-1} is implicitly constructed, e.g., an incomplete LU decomposition [14], in such a way, that any linear system

$$Mz = c \tag{2.3}$$

takes little effort to solve. Even more generally, it can be some iterative linear system solver, where M implicitly exists to resemble A in some way and varies with the right-hand side c. This is exactly what the flexible GMRES (FGMRES) [13] does: iteratively solve $Az_j = V_{(:,j)}$ for z_j approximately. So effectively, FGMRES is an inner-outer iterative scheme.

With the goal of developing a general purpose linear system solver, we will further restrict ourselves to the use of the m-step plain GMRES for $Az_j = V_{(:,j)}$. In other words, in the rest of this paper, the inner iteration of our FGMRES is always the m-step plain GMRES. In doing so, we actually use a degree $(m-1)$ polynomial $P_m(A)$ [5] to approximate A^{-1}.

Consequently, each solving implicitly determines a preconditioner M that differs from one solving to another. To distinguish them, we add the loop-index to each preconditioner, namely, M_j. Different from what we had in the previous subsection, now $Z_k = [M_1^{-1}v_1, \cdots, M_k^{-1}v_k]$, where $v_j = V_{(:,j)}$ and M_j^{-1} is unknown but $M_j^{-1}v_j$ is known. We still have (2.1) and (2.2). We outline FGMRES as in Algorithm 2.2.

Algorithm 2.2. FGMRES(m,k): k-step Flexible GMRES(m)

Input: $A \in \mathbb{C}^{n\times n}$, $x_0 \in \mathbb{C}^n$, integers k and m;
Output: An approximate solution x_k to $Ax = b$.
1: Replace Line 4 of Algorithm 2.1 by: solve $Az_j = V_{(:,j)}$ approximately by the m-step GMRES.

Numerical tests show that FGMRES often improves GMRES and sometimes the improvement can be rather significant. But it is still a possibility that FGMRES may need a large k to deliver a sufficiently accurate approximate solution x_k. When that's the case, it will need a lot of memory space to store $V_{(:,j)}$ and z_j and require heavy costs to orthogonalize $V_{(:,j)}$. In order to deal with this problem, one can also restart FGMRES as in the restarted GMRES, i.e., repeat the k-step FGMRES(m) with the current initial guess being the solution of the previous k-step FGMRES(m). We denote it by REFGMRES(m,k) and call each k-step FGMRES(m) a *cycle*. The framework of REFGMRES(m,k) is presented in Algorithm 2.3.

Algorithm 2.3. REFGMRES(m,k)

Input: $A \in \mathbb{C}^{n\times n}$, $x_0 \in \mathbb{C}^n$, integers k and m;
Output: An approximate solution $x_k^{(\ell)}$ to $Ax = b$.
1: $x_0^{(1)} = x_0$, $r_0^{(1)} = b - Ax_0^{(1)}$;
2: **for** $\ell = 1, 2, \ldots,$ **do**
3: **if** $\|r_0^{(\ell)}\|_2 \le \texttt{tol} \times (\|A\|_1\|x_0^{(\ell)}\|_2 + \|b\|_2)$ **then**
4: **break**;
5: **else**
6: call Algorithm 2.2 with input $x_0^{(\ell)}$, k and m, and let $x_k^{(\ell)}$ be the returned approximation;
7: $x_0^{(\ell+1)} = x_k^{(\ell)}$;
8: $r_0^{(\ell+1)} = b - Ax_0^{(\ell+1)}$;
9: **end if**
10: **end for**
11: **return** $x_k^{(\ell)}$ as the computed solution to $Ax = b$.

2.3. Heavy Ball GMRES

It is well known that REGMRES may encounter slow convergence, partly because at every restart, REGMRES completely ignores all the Krylov subspaces built in the previous cycles for the purpose of cost control in memory and flops. Imakura, Li, and Zhang [7] proposed the locally optimal GMRES (LOGMRES) and the heavy ball GMRES (HBGMRES) methods, designed to recoup some information from previous cycles back for the purpose of accelerating the convergence of REGMRES(k). As mentioned in the introduction, HBGMRES is motivated by the optimization method, i.e., the heavy ball method, and it is a special case of Loose GMRES [1]. In this part, we focus on the work by Imakura, Li, and Zhang.

The so-called heavy ball method [11, p. 65] is an optimization method

$$x^{(\ell+1)} = \underset{s,t}{\operatorname{argmin}} f\left(x^{(\ell)} + t\nabla f\big(x^{(\ell)}\big) + s\big(x^{(\ell)} - x^{(\ell-1)}\big)\right)$$

for minimizing a differentiable function $f(x)$. Its name is drawn from the motion of a "heavy ball" in a potential field under the force of friction. The heavy ball method is a multi-step method. It brings in information of the previous cycles by just including the difference of last two approximate solutions. The heavy ball GMRES is an application of the heavy ball method to the linear system solving. In every HBGMRES cycle, the search space is expanded to $\mathcal{K}_k(A, r_0^{(\ell)}) + \operatorname{span}\{x_0^{(\ell)} - x_0^{(\ell-1)}\}$. In [7], numerical experiments demonstrated that HBGMRES often converges much faster than REGMRES. More detailed efficiency discussion of HBGMRES can be found in [7] and [1].

2.4. A Brief Comparison among Variants of GMRES

In what follows, we will conduct a brief comparison among the methods we have mentioned so far in an effort to justify our focus on improving the restarted FGMRES in the next section.

In Table 2.1, we estimate the numbers of flops for GMRES and its variants. MV represents the number of flops by one matrix-vector multiplication with $A \in \mathbb{C}^{n\times n}$, which is taken to be twice the number of nonzero entries in A. GMRESR and FGMRES use the same space information both in their inner and outer iterations except that they are implemented differently in their respective outer iterations. Because our focus is on Krylov subspace methods that are easy to implement and use, as easy as GMRES itself, GMRES variants like GMRES-E, GMRES-IR, and GMRES-DR are, as a result, excluded in our comparison. In the table, only three major actions are considered in the total flops: matrix-vector multiplications, orthogonalization, and solutions of the reduced least squares problems. In the table and in the rest of this paper, REGMRES(k) stands for the restart of the k-step GMRES. The parameter k in HBGMRES(k) indicates the order of Krylov subspace in each cycle of HBGMRES(k), while m in GMRESR(m) and FGMRES(m) is the Arnoldi steps in each inner iteration, i.e., indicating Line 4 of Algorithm 2.1 is done by solving $Az_j = V_{(:,j)}$ approximately by the m-step GMRES.

Table 2.1: Total Flops of GMRES Variants.

ℓ-step GMRES	$(\ell+1)$(MV)$+2\ell^2 n + 4\ell^2$
ℓ-cycle REGMRES(k)	$\ell((k+1))$(MV)$+\ell(2k^2 n + 4k^2)$
ℓ-cycle HBGMRES(k)	$\ell(k+2)$(MV)$+\ell(2(k+2)^2 n + 4(k+1)^2)$
ℓ-step GMRESR(m)	$\ell(m+2)$(MV)$+2(\ell^2 + \ell m^2)n + 4(\ell n + \ell m^2)$
ℓ-step FGMRES(m)	$\ell(m+2)$(MV)$+2(\ell^2 + \ell m^2)n + 4(\ell^2 + \ell m^2)$

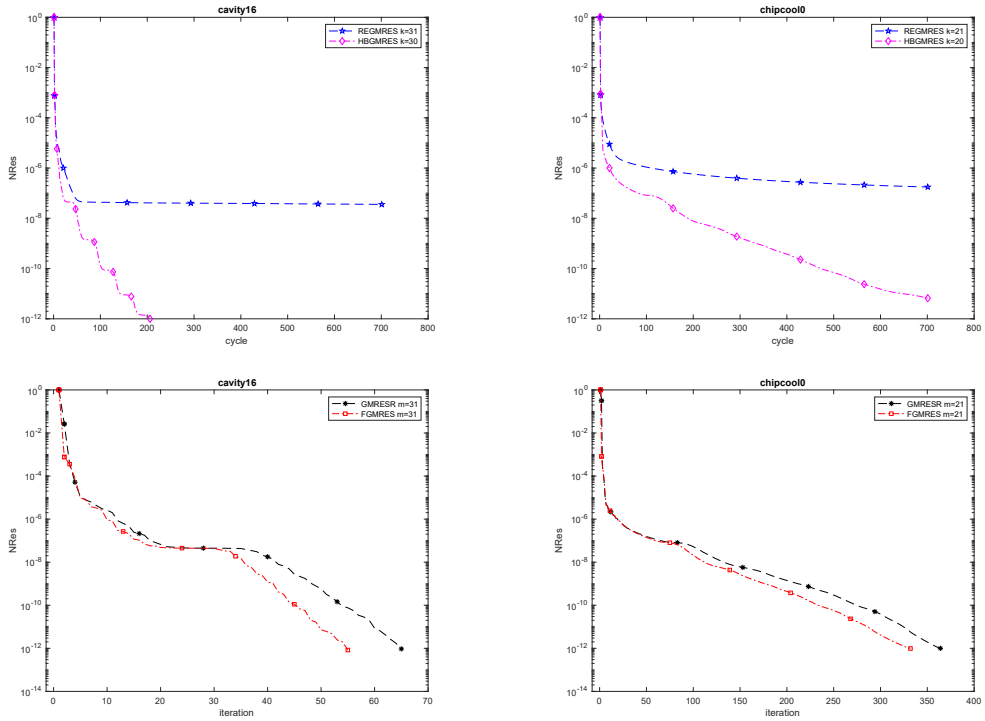

Fig. 2.1. Numerical Tests on `cavity16` and `chipcool0`. Top-row: REGMRES and HBGMRES; Bottom-row: GMRESR and FGMRES.

Next we present the numerical results by the last four methods in Table 2.1 on two notorious difficult testing examples `cavity16` and `chipcool0` from SuiteSparse Matrix Collection[1)] . Both of them have bad condition number over 10^7. More detailed information on the examples can be found in Section 4. In choosing the number m of GMRES steps for each inner iteration of GMRESR and FGMRES, we decide to make the cost be about the same as one cycle in REGMRES and HBGMRES. We point out that one outer iteration in GMRESR and FGMRES is like one cycle in REGMRES and HBGMRES. Therefore, we choose the parameter m in the inner iteration of FGMRES and GMRESR is to be $m = k + 1$. As a result, each cycle of REGMRES$(k + 1)$ and HBGMRES(k) and each outer iterative step of GMRESR(m) and FGMRES(m) have about the same cost. We terminate each method by either the maximal number of cycles or outer iterative steps exceeds 700, or the normalized residual moves below 10^{-12}. Table 2.2 shows the numbers of cycles/iterations and CPU times needed by the four methods and Fig. 2.1 displays the convergence histories in terms of the normalized residual.

Table 2.2: Cycles and CPU Time (s).

matrix	k	m	REGMRES$(k+1)$		HBGMRES(k)		GMRESR(m)		FGMRES(m)	
			cycle	CPU	cycle	CPU	iteration	CPU	iteration	CPU
`cavity16`	30	31	–	–	206	2.25	65	0.78	55	0.57
`chipcool0`	20	21	–	–	–	–	364	8.21	332	7.54

[1)] Available at https://sparse.tamu.edu/

We observed the following:

- REGMRES didn't take the normalized residual down to 10^{-12} in 700 cycles for both examples. In fact, for `cavity16` with parameter $k = 31$, REGMRES takes 31193 iterations and 343.01 seconds, which are way too large, to make the normalized residual down to the tolerance. For `chipcool0` with parameter $k = 21$, the normalized residual is as much as 10^{-9} after 40000 iterations.

- HBGMRES didn't take the normalized residual down to 10^{-12} in 700 cycles for `chipcool0`, although it did manage to take it to about 10^{-11} in 700 cycles.

- While both FGMRES and GMRESR perform much better than HBGMRES, FGMRES comes out as the best among all.

Basing on these preliminary results, in what follows we will focus on improving the restarted FGMRES in the rest of this paper.

3. Heavy Ball Flexible GMRES Method

3.1. HBFGMRES

As mentioned before, it is possible for FGMRES to require a large number of outer iterative steps to converge, although less frequently than GMRES. Hence sometimes FGMRES has to be restarted in order to control the memory usage and reorthogonalization cost of V_k (see Algorithms 2.1 and 2.2). Meanwhile, in order to make up the possibly slow convergence due to restart, in this section we borrow the heavy-ball idea to devise the heavy ball FGMRES (HBFGMRES). The purpose is to include some Krylov subspace information of previous cycles for the benefit of faster convergence. The framework of HBFGMRES is presented in Algorithm 3.1.

The main difference between HBFGMRES and REFGMRES lies in its outer iterative cycles in that after building the Krylov subspace, HBFGMRES further expands the subspace by adding a new direction, the difference between two approximations of previous cycles. According to the analysis in [1], since FGMRES may be viewed as an accelerated GMRES via variable preconditioning technique, HBGMRES is also an accelerated GMRES, that improves the approximation space besides the preconditioning technique.

We remark that similarly to LGMRES [1], we include more than one previous difference vectors to the search space, i.e., modify the update at Line 7 in Algorithm 3.1 to

$$x_k^{(\ell)} = x_0^{(\ell)} + \underset{z \in \mathrm{span}\{Z_k\} + \sum_{q=p}^{q=\ell-p} \mathrm{span}\{x_{\mathrm{d}}^{(q)}\}}{\operatorname{argmin}} \left\| r_0^{(\ell)} - Az \right\|_2, \tag{3.1}$$

where $x_{\mathrm{d}}^{(q)} = x_k^{(\ell)} - x_k^{(\ell-1)}$. Here, instead of one previous difference vectors, $\ell - p$ previous difference vectors are used to expand the search space.

In the next subsection, we will explain how HBFGMRES(m, k) is implemented. With modifications, updating scheme (3.1) can be implemented in a similar way. Later, we will provide some numerical evidence to demonstrate that including more than one previous difference vectors does not yield much reward, if any.

Algorithm 3.1. HBFGMRES(m, k)

Input: $A \in \mathbb{C}^{n\times n}$, $x_0^{(1)} \in \mathbb{C}^n$, integers k and m;
Output: An approximate solution $x_k^{(\ell)}$ to $Ax = b$.
1: $r_0^{(1)} = b - Ax_0^{(1)}$ (and $x_0^{(0)} = 0$ for convenience);
2: **for** $\ell = 1, 2, \ldots,$
3: **if** $\|r_0^{(\ell)}\|_2 \le \texttt{tol} \times (\|A\|_1\|x_0^{(\ell)}\| + \|b\|_2)$ **then**
4: **break**;
5: **else**
6: build $Z_k = [z_1, z_2, \ldots, z_k]$ with $r_0^{(\ell)}$, k and m as in Algorithm 2.2;
7: compute

$$x_k^{(\ell)} = x_0^{(\ell)} + \operatorname*{argmin}_{z\in \mathrm{span}\{Z_k\}+\mathrm{span}\{x_0^{(\ell)}-x_0^{(\ell-1)}\}} \left\|r_0^{(\ell)} - Az\right\|_2;$$

8: $x_0^{(\ell+1)} = x_k^{(\ell)}$;
9: $r_0^{(\ell+1)} = b - Ax_0^{(\ell+1)}$;
10: **end if**
11: **end for**
12: **return** $x_k^{(\ell)}$ as the computed solution to $Ax = b$.

3.2. Implementation of HBFGMRES

The approximate solution of the ℓ-th cycle of REFGMRES(m, k) can be expressed as

$$x_k^{(\ell)} = x_0^{(\ell)} + Z_k y_k \in x_0^{(\ell)} + \mathrm{span}\{Z_k\}.$$

Our proposed HBFGMRES improves REFGMRES by expanding the search space $\mathrm{span}\{Z_k\}$ to include a new direction $x_{\mathrm{d}} = x_0^{(\ell)} - x_0^{(\ell-1)}$, and then a new approximate solution, denoted by the same notation $x_k^{(\ell)}$ without confusion, can be written as

$$x_k^{(\ell)} = x_0^{(\ell)} + Z_k y_k + \alpha x_{\mathrm{d}}. \tag{3.2}$$

Correspondingly, the ℓ-th residual becomes

$$r_k^{(\ell)} = b - Ax_k^{(\ell)} = r_0^{(\ell)} - V_{k+1}\check{H}y_k - \alpha A x_{\mathrm{d}}, \tag{3.3}$$

which has the smallest norm $\|b - Ax\|_2$ among all $x \in x_0 + \mathrm{span}\{Z_k, x_{\mathrm{d}}\}$. In what follows, we explain how to numerically compute $x_k^{(\ell)}$ efficiently. Recall that we have $AZ_k = V_{k+1}\check{H}$ by the Arnoldi process before the new direction x_{d} is added, and that is why we have (3.3).

Denote by $p = Ax_{\mathrm{d}}$. If $p = 0$, this case is exactly the restarted FGMRES. We assume $p \neq 0$. Now we orthogonalize p against V_{k+1} , and define

$$d = V_{k+1}^{\mathrm{H}} p, \quad h = p - V_{k+1}d.$$

Case 1. $h \neq 0$. This is the most generic and common case. Define

$$\check{v}_{k+2} = h/\|h\|_2, \quad \check{V}_{k+2} = [V_{k+1} \quad \check{v}_{k+2}].$$

We have

$$\begin{aligned} r_k^{(\ell)} &= r_0^{(\ell)} - V_{k+1}\check{H}y_k - \alpha A x_{\rm d} \\ &= r_0^{(\ell)} - V_{k+1}\check{H}y_k - \alpha(\check{v}_{k+2}\|h\|_2 + V_{k+1}d) \\ &= r_0^{(\ell)} - \check{V}_{k+2}\begin{bmatrix}\check{H} & d \\ 0 & \|h\|_2\end{bmatrix}\begin{bmatrix}y_k \\ \alpha\end{bmatrix}. \end{aligned}$$

Therefore in order for $\|r_k^{(\ell)}\| = \min \|b - Ax\|_2$ among all $x \in x_0 + \text{span}\{Z_k, x_{\rm d}\}$, y_k and α must be given by

$$\begin{bmatrix}y_k \\ \alpha\end{bmatrix} = \underset{y,\alpha}{\text{argmin}} \left\| \beta e_1 - \begin{bmatrix}\check{H} & d \\ 0 & \|h\|_2\end{bmatrix}\begin{bmatrix}y \\ \alpha\end{bmatrix}\right\|_2, \tag{3.4}$$

where $\beta = \|r_0^{(\ell)}\|_2$. Finally, the approximate solution of the ℓ-th cycle of HBFGMRES(k) is given by (3.2), satisfying

$$x_k^{(\ell)} - x_0^{(\ell)} \in \text{span}\{Z_k\} + \text{span}\{x_{\rm d}\}.$$

Case 2. $h = 0$. In this case, we have $\|h\|_2 = 0$ and $Ax_{\rm d} = V_{k+1}d$. Then

$$\begin{aligned} r_k^{(\ell)} &= r_0^{(\ell)} - V_{k+1}\check{H}y_k - \alpha A x_{\rm d} \\ &= r_0^{(\ell)} - V_{k+1}\check{H}y_k - \alpha(V_{k+1}d) \\ &= r_0^{(\ell)} - V_{k+1}\begin{bmatrix}\check{H} & d\end{bmatrix}\begin{bmatrix}y_k \\ \alpha\end{bmatrix}. \end{aligned}$$

Therefore

$$\begin{bmatrix}y_k \\ \alpha\end{bmatrix} = \underset{y,\alpha}{\text{argmin}} \left\| \beta e_1 - \begin{bmatrix}\check{H} & d\end{bmatrix}\begin{bmatrix}y \\ \alpha\end{bmatrix}\right\|_2. \tag{3.5}$$

The details of our HBFGMRES(m, k) is summarized in Algorithm 3.2.

Algorithm 3.2. HBFGMRES(m, k) with implementation details

Input: $A \in \mathbb{C}^{n\times n}$, $x_0^{(1)} \in \mathbb{C}^n$, integers k and m;
Output: An approximate solution $x_k^{(\ell)}$ to $Ax = b$.
1: $r_0^{(1)} = b - Ax_0^{(1)}$ (and $x_0^{(0)} = 0$ for convenience);
2: **for** $\ell = 1, 2, \ldots,$ **do**
3: **if** $\|r_0^{(\ell)}\|_2 \le \texttt{tol} \times (\|A\|_1\|x_0^{(\ell)}\|_2 + \|b\|_2)$ **then**
4: **break**;
5: **else**
6: build $Z_k = [z_1, z_2, \ldots, z_k]$ with A and $r_0^{(\ell)}$ as in Algorithm 2.2;
7: set $x_{\rm d} = x_0^{(\ell)} - x_0^{(\ell-1)}$, and orthogonalize $Ax_{\rm d}$ against V_{k+1};
8: solve (3.4) or (3.5) for (y_k, α);
9: $x_k^{(\ell)} = x_0^{(\ell)} + Z_k y_k + \alpha x_{\rm d}$;
10: $x_0^{(\ell+1)} = x_k^{(\ell)}$;
11: $r_0^{(\ell+1)} = b - Ax_0^{(\ell+1)}$;
12: **end if**
13: **end for**
14: **return** $x_k^{(\ell)}$ as a computed solution to $Ax = b$.

4. Numerical Experiments

In general, in terms of the number of matrix-vector multiplications, GMRES uses fewer than REGMRES for achieving the same solution accuracy. The latter gets used because when GMRES needs a large number of Arnoldi steps to converge, the memory requirement to store the orthonormal basis vectors and the orthogonalization cost are too great for a large scale problem, not to mention the tendency in orthogonality loss among the basis vectors. For the same reason, REFGMRES may have to be used in place of FGMRES, even though FGMRES in general uses fewer matrix-vector multiplications.

In [7], it was reported that the heavy-ball idea in HBGMRES usually improves the plain REGMRES and often significantly. We have showed in Section 2 that FGMRES is faster than GMRESR. In what follows, we will numerically demonstrate how much the heavy ball idea in HBFGMRES improves the plain REFGMRES. In order to fairly compare the two algorithms and make it easy to understand, we use the following testing scenarios:

- In order to equalize the cost per cycle, we run REFGMRES$(m, k+1)$ and HBFGMRES(m, k). There are two reasons for this choice. Firstly, all approximate solutions at a cycle are computed from a subspace of dimension $k+1$: $\text{span}\{x_0\}+\text{span}\{Z_{k+1}\}$ for REFGMRES$(m, k+1)$, and $\text{span}\{x_0\}+\text{span}\{Z_k\}+\text{span}\{x_\text{d}\}$ for HBFGMRES(m, k); secondly, this will make the computational costs per cycle about the same for both methods. Note that for the first two cycles, there is no difference happened, both REFGMRES and HBFGMRES have the same restart number k, which indicates the exactly same relative residual obtained for both methods.

- Each $Ax = b$ is tested on selective reorthogonalization and always reorthogonalization, which takes place in the Arnoldi process and in the additional orthogonalizations in HBFGMRES. REFGMRES and HBFGMRES involve two Arnoldi process. One occurs in the inner iteration and the other one is the outer iteration of each cycle. Here, the choice of using selective or always reorthogonalization is consistently applied regardless inner or outer iterations. Selective reorthogonalization goes as follows. At Line 6-8 of Algorithm 2.1, it applies the modified Gram-Schmidt (MGS) orthogonalization to orthogonalize Az_j against already computed columns of V. Denote $\alpha_0 = \|f\|_2$ at Line 5 and $\alpha_1 = \|f\|_2$ obtained after Line 8. If $\alpha_1 \leq \texttt{tol\_orth} \times \alpha_0$ is true, where `tol_orth` is a preset tolerance,

Table 4.1: Testing Matrices.

matrix	n	nnz	sparsity	application
cavity10	2597	76367	1.13×10^{-2}	computational fluid dynamics
garon2	13535	373235	2.00×10^{-3}	computational fluid dynamics
comsol	1500	97645	4.34×10^{-2}	structural problem
chipcool0	20082	281150	6.97×10^{-4}	model reduction problem
flowmeter5	9669	67391	7.21×10^{-4}	computational fluid dynamics
e20r0000	4241	131413	7.30×10^{-3}	computational fluid dynamics
e20r0100	4241	131413	7.30×10^{-3}	computational fluid dynamics
sherman3	5005	20033	7.99×10^{-4}	computational fluid dynamics
hcircuit	105676	513072	4.59×10^{-4}	circuit simulation problem
xenon2	157464	3866688	1.56×10^{-4}	materials problem

Table 4.2: Number of Cycles.

matrix	k	m	always reorth		selective reorth	
			REF	HBF	REF	HBF
cavity10	20	10	95	**38**	106	**38**
garon2	50	10	725	107	593	**102**
comsol	10	20	53	16	60	**14**
chipcool0	30	20	55	**21**	66	**21**
flowmeter5	10	10	496	**333**	492	**333**
e20r0000	10	10	816	**14**	732	**14**
e20r0100	30	20	45	**14**	50	**14**
sherman3	30	10	211	**41**	232	**41**
hcircuit	10	10	–	**735**	–	849
xenon2	10	10	32	**22**	33	**22**

Table 4.3: CPU Time (s).

matrix	k	m	always reorth		selective reorth	
			REF	HBF	REF	HBF
cavity10	20	10	4.35	1.68	4.44	**1.48**
garon2	50	10	502.93	74.35	354.68	**61.20**
comsol	10	20	2.91	0.69	2.89	**0.58**
chipcool0	30	20	66.63	24.26	68.02	**21.91**
flowmeter5	10	10	29.32	18.45	22.65	**14.45**
e20r0000	10	10	40.36	0.64	32.32	**0.51**
e20r0100	30	20	15.49	4.48	13.71	**3.58**
sherman3	30	10	20.35	3.91	15.38	**2.73**
hcircuit	10	10	–	340.84	–	**318.77**
xenon2	10	10	41.08	24.59	37.08	**24.41**

we repeat MGS one more time, i.e., at the end of the for-loop at Line 6-8, we add

$$\begin{aligned} &\textbf{for} \quad i = 1, 2, \ldots, j \quad \textbf{do} \\ &\quad t = V_{(:,i)}^{\mathrm{H}} f, \check{H}_{(i,j)} = \check{H}_{(i,j)} + t, f = f - V_{(:,i)}\, t. \\ &\textbf{end for} \end{aligned} \tag{4.1}$$

For the always reorthogonalization, we repeat (4.1) immediately after Line 8 of Algorithm 2.1. In our numerical tests, we take $\texttt{tol\_orth} = 10^{-2}$.

- The stopping criterion is set as 10^{-12} on NRes

$$\mathrm{NRes} = \frac{\|r\|_2}{\|A\|_1 \times \|x\|_2 + \|b\|_2},$$

where using $\|A\|_1$ is for its easiness in computation. We also set a maximum number of cycles to 1000.

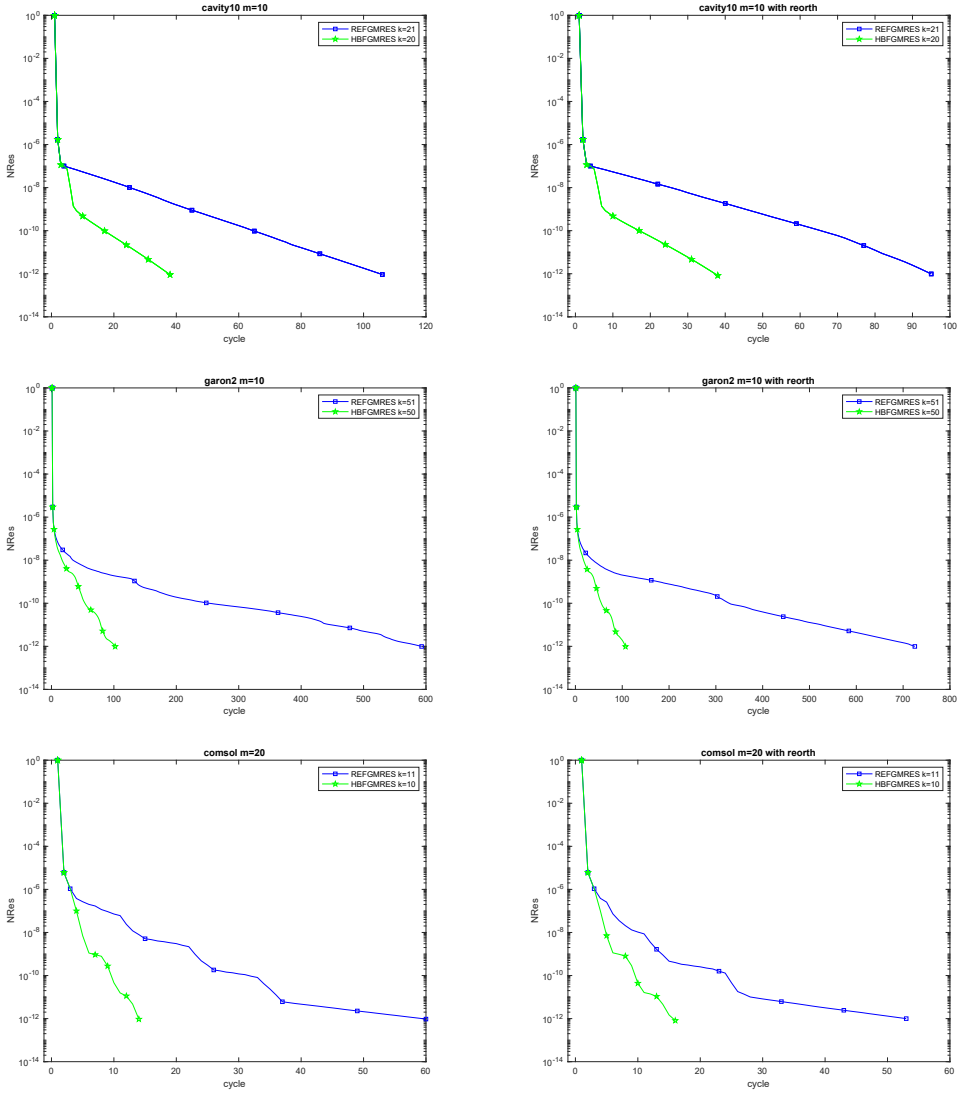

Fig. 4.1. NRes *vs*. cycle for `cavity10`, `garon2` and `comsol`. Left: selective reorthogonalization; Right: always reorthogonalization.

In what follows, we will report our numerical results on 10 testing problems which are selected from the SuiteSparse Matrix Collection[1)] and Matrix Market[2)] .

Each comes with a right-hand side b except for `garon2` and `xenon2` whose right-hand sides are randomly generated with `rand` in MATLAB. Table 4.1 lists some of their important characteristics, including the size of matrix n, the number `nnz` of nonzero entries in A, and the sparsity $\texttt{nnz}/n^2$. These are representatives among many other problems from the collections we have tested. The Matlab is of version 2018b and all the tests are done on a Mac PC with 2.7 GHz Intel Core i7 and 16GB memory.

Numerical results of testing examples are displayed in Tables 4.2 and 4.3, where the best results appear in boldface. For each matrix, we ran different k and m. Table 4.2 lists the

[1)] Available at https://sparse.tamu.edu/
[2)] Available at https://math.nist.gov/MatrixMarket/

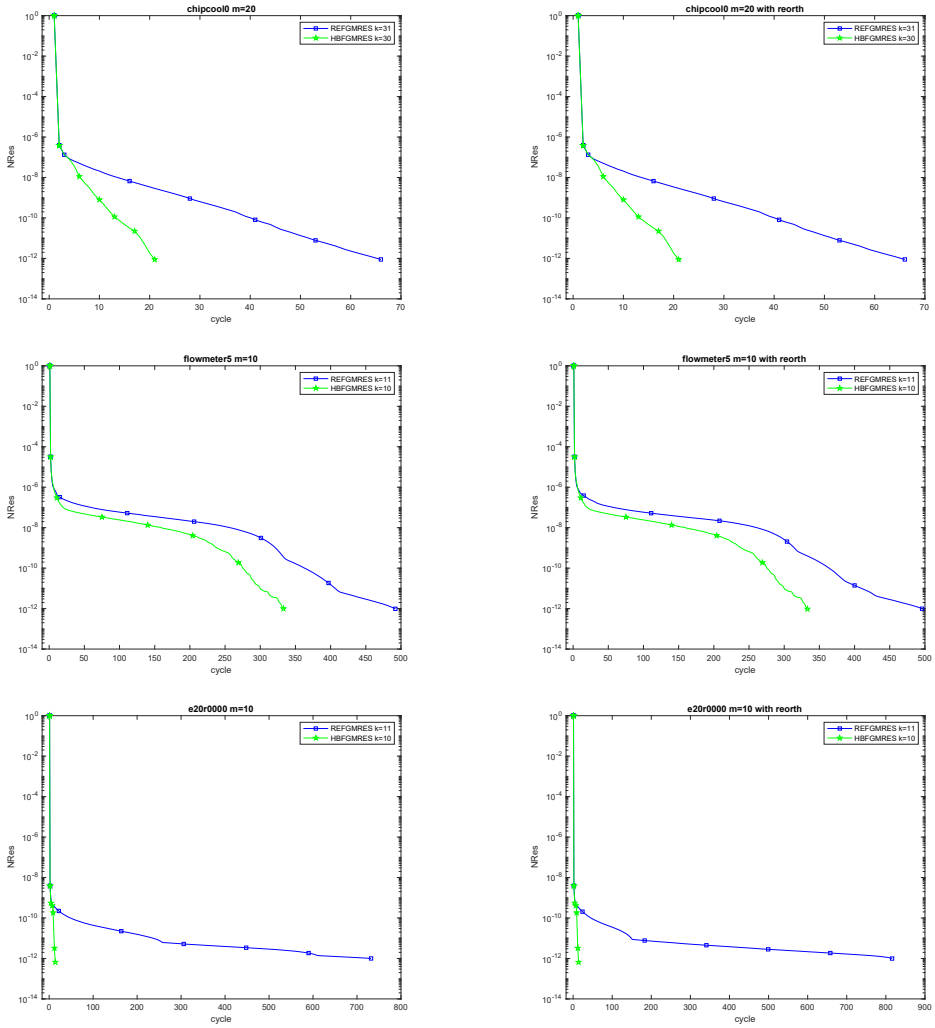

Fig. 4.2. NRes *vs.* cycle for `chipcool0`, `flowmeter5` and `e20r0000`. Left: selective reorthogonalization; Right: always reorthogonalization.

number of cycles needed by the algorithms to achieve NRes less than or equal to 10^{-12}. In the table, REF and HBF stand for REFGMRES and HBFGMRES, respectively. The table clearly demonstrates huge savings achieved by HBFGMRES over REFGMRES.

Figs. 4.1-4.4 show how NRes moves against the cycle index. Together with Tables 4.2 and 4.3, we make the following observations.

- HBFGMRES converges faster than REFGMRES, both in terms of the number of cycles and CPU times.
- Except for `hcircuit`, there is little difference in the number of cycles with always reorthogonalization or selective reorthogonalization. But selective reorthogonalization cost less in CPU time for all examples. Hence, selective reorthogonalziation seems to be a better choice for cost consideration.

We point out in passing that `flowmeter5` and `sherman3` are two difficult problems for FGM-

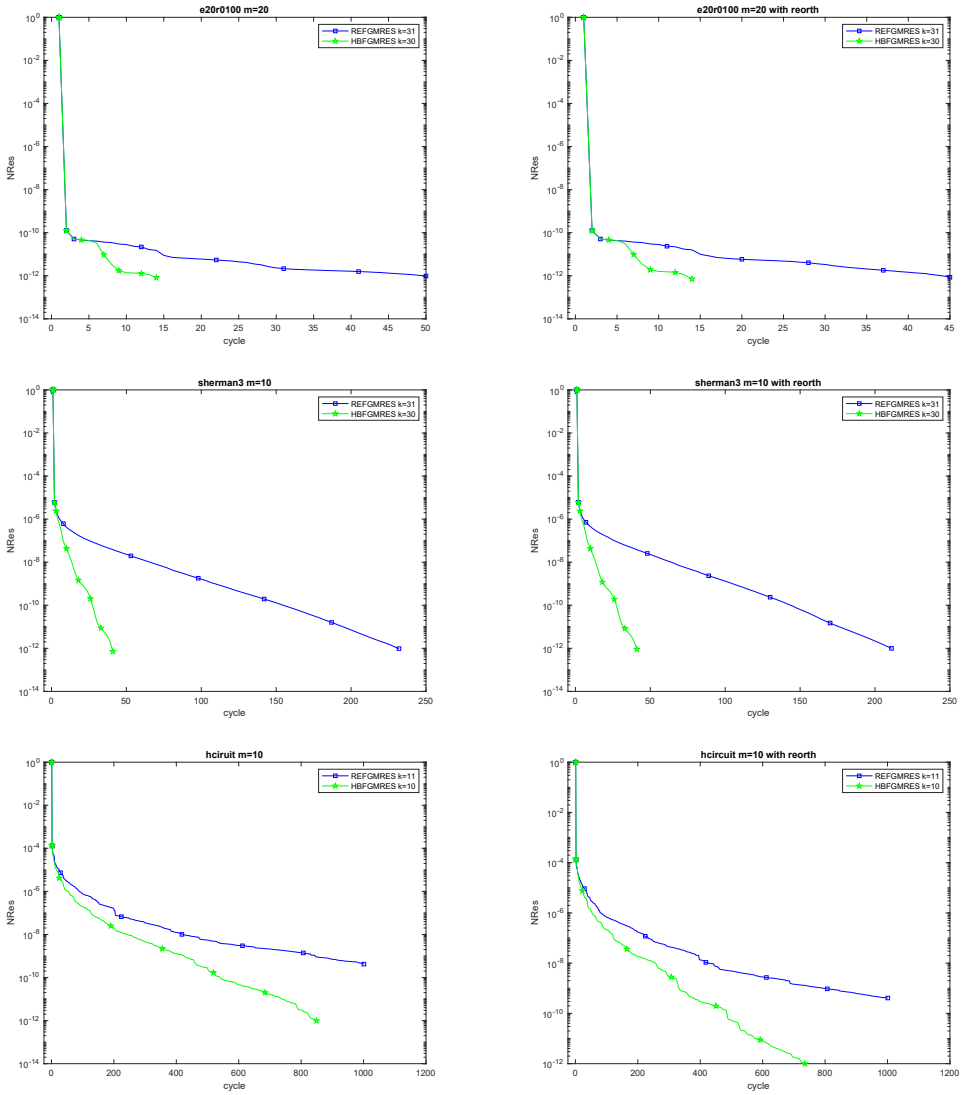

Fig. 4.3. NRes *vs*. cycle for `e20r0100`, `sherman3` and `hcircuit`. Left: selective reorthogonalization; Right: always reorthogonalization.

RES, needing a large number of outer iterative steps to converge. For example, with selective reorthogonalziation, $m = 10$, FGMRES took 708 outer steps and used 24.19 seconds on `flowmeter5`, which is faster than REFGMRES but slower than HBFGMRES (see Table 4.3). It also means that FGMRES will have to store 707 basis vectors at the end, whereas at any moment HBFGMRES stores no more than $k + m + 1 = 21$ basis vectors. On `sherman3`, FGMRES took 561 outer steps and used 8.17 seconds, also faster than REFGMRES but slower than HBFGMRES.

Notice that for examples `garon2` and `e20r0000`, HBFGMRES converges overwhelmingly faster. According to the theoretical and numerical analysis in [2] and [1], consecutive angles, i.e., angles between the residuals $r_0^{(\ell)}$ and $r_0^{(\ell+1)}$, determine the convergence rate for Krylov subspace methods. The larger the angles are, the faster the convergence will be. Thus, for `garon2`

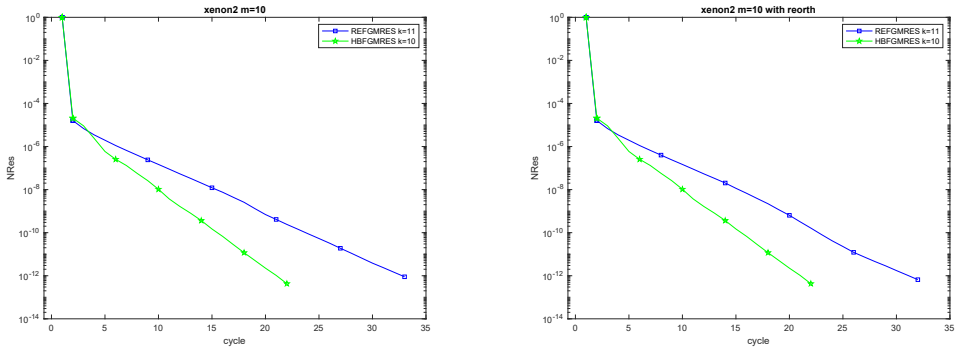

Fig. 4.4. NRes *vs*. cycle for `xenon2`. Left: selective reorthogonalization; Right: always reorthogonalization.

and `e20r0000`, we numerically test consecutive angles for REFGMRES and HBFGMRES, respectively. We looked into the consecutive angles and found that the angles for HBFGMRES are much larger than these for REFGMRES, which explains why HBFGMRES performs so much superior to REFGMRES.

Lastly, we will present some numerical results to show that little will be gained when HBFGMRES is extended to include more than one previous difference vectors. Here, we compare HBFGMRES(m, k) with the next variant that includes two most recent difference vectors in each cycle. For ease of reference, they are denoted by HBF1 and HBF2, respectively. For a fair comparison, HBF2 will use $k-1$ z-vectors in the outer iteration while HBF1 uses k z-vectors (cf. Algorithm 3.1) so that the dimension of the search spaces for both HBF1 and HBF2 are the same in their outer iterations. Table 4.4 displays the numbers of cycles needed by HBF1 and HBF2 on the 10 testing problems when selective reorthogonalization is used. What we can observe from the table is that HBF1 and HBF2 have comparable performance for most of examples, except for `hcircuit`. Most importantly, the improvements, when HBF2 indeed costs less, aren't that significant over HBF1, comparing to HBFGMRES over REFGMRES, not to mention there are cases HBF2 costs more. In view of this, we would recommend the use of HBFGMRES.

Table 4.4: Number of Cycles.

matrix	k	m	REF	HBF1	HBF2
`cavity10`	20	10	106	38	**31**
`garon2`	50	10	593	102	**89**
`comsol`	10	20	60	14	**10**
`chipcool0`	30	20	66	**21**	22
`flowmeter5`	10	10	492	**333**	346
`e20r0000`	10	10	732	**14**	**14**
`e20r0100`	30	20	50	14	**9**
`sherman3`	30	10	232	41	**37**
`hcircuit`	10	10	–	849	**433**
`xenon2`	10	10	33	**22**	**22**

5. Conclusion

In this paper, we proposed the heavy-ball flexible GMRES (HBFGMRES) method for nonsymmetric linear systems. This method accelerates the convergence of restarted FGMRES (REFGMRES) by adding a new direction to recoup some of the Krylov subspace information from previous search spaces. Numerical examples show that HBFGMRES converges much faster than REFGMRES. Together in consideration of earlier work in [7], we may conclude that the heavy-ball idea is a good way to accelerate convergence of restarted GMRES-related methods.

Acknowledgments. Ren-Cang Li was supported in part by NSF grants DMS-1719620 and DMS-2009689. The authors wish to thank the two anonymous referees for their careful readings, useful comments and suggestions that improve the presentation.

References

[1] A.H. Baker, E.R. Jessup and T. Manteuffel, A technique for accelerating the convergence of restarted GMRES, *SIAM J. Matrix Anal. Appl.*, **26**:4 (2005), 962–984.

[2] M. Eiermann and O. Ernst, Geometri aspects of the theory of Krylov subspace methods, *Acta Numerica*, **10** (2001), 251–312.

[3] S.C. Eisenstat and H.C. Elman and M. Schultz, Variational iterative methods for nonsymmetric systems of linear equations, *SIAM J. Numer. Anal.*, **20**:2 (1983), 345–357.

[4] D.R. Fokkema, G.L.G. Sleijpen and H.A. Van der Vorst, Accelerated inexact Newton schemes for large systems of nonlinear equations, *SIAM J. Sci. Comput.*, **19**:2 (1998), 657–674.

[5] I. Gohberg, P. Lancaster and L. Rodman, *Matrix Polynomials*, SIAM, 2009.

[6] A. Greenbaum, *Iterative Methods for Solving Linear Systems*, SIAM, 1997.

[7] A. Imakura and R.C. Li and S.L. Zhang, Locally optimal and heavy ball GMRES methods, *Japan J. Indust. Appl. Math.*, **33** (2016), 471–499.

[8] R.B. Morgan, A restarted GMRES method augmented with eigenvectors, *SIAM J. Matrix Anal. Appl.*, **16**:4 (1995), 1154–1171.

[9] R.B. Morgan, Implicitly restarted GMRES and Arnoldi methods for nonsymmetric systems of equations, *SIAM J. Matrix Anal. Appl.*, **21**:4 (2000), 1112–1135.

[10] R.B. Morgan, GMRES with deflated restarting, *SIAM J. Matrix Anal. Appl.*, **24**:1 (2002), 20–37.

[11] B.T. Polyak, *Introduction to Optimization*, Optimization Software, SIAM, 1987.

[12] Y. Saad, Krylov subspace methods for solving large unsymmetric linear systems, *Math. Comp.*, **37**:155 (1981), 105–126.

[13] Y. Saad, A flexible inner-outer preconditioned GMRES algorithm, *SIAM J. Sci. Comput.*, **14**:2 (1993), 461–469.

[14] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd edition, SIAM, 2003

[15] H.A. Van der Vorst and C. Vuik, GMRESR: a family of nested GMRES methods, *Numer. Linear Algebra Appl.*, **1**:4 (1994), 369–386.

[16] C. Vuik, New insights in GMRES-like methods with variable preconditioners, *J. Comput. Appl. Math.*, **61**:2 (1995), 189–204.