Journal of Computational and Applied Mathematics 235 (2011) 1956–1975

Contents lists available at ScienceDirect

Journal of Computational and Applied Mathematics

journal homepage: www.elsevier.com/locate/cam

Evaluation of the performance of inexact GMRES

Roger B. Sidje *, Nathan Winkles

The University of Alabama, Department of Mathematics, P.O. Box 870350, Tuscaloosa, AL 35487, USA

ARTICLE INFO

Article history:
Received 23 April 2010

Keywords:
Nonsymmetric linear system
Iterative solver
Inexact GMRES
Sparse matrix
Relaxed matrix vector-product

ABSTRACT

The inexact GMRES algorithm is a variant of the GMRES algorithm where matrix–vector products are performed inexactly, either out of necessity or deliberately, as part of a trading of accuracy for speed. Recent studies have shown that relaxing matrix–vector products in this way can be justified theoretically and experimentally. Research, so far, has focused on decreasing the workload per iteration without significantly affecting the accuracy. But relaxing the accuracy per iteration is liable to increase the number of iterations, thereby increasing the overall runtime, which could potentially end up being greater than that of the exact GMRES if there were not enough savings in the matrix–vector products. In this paper, we assess the benefit of the inexact approach in terms of actual CPU time derived from realistic problems, and we provide cases that provide instructive insights into results affected by the build-up of the inexactness. Such information is of vital importance to practitioners who need to decide whether switching their workflow to the inexact approach is worth the effort and the risk that might come with it. Our assessment is drawn from extensive numerical experiments that gauge the effectiveness of the inexact scheme and its suitability for use in addressing certain problems, depending on how much inexactness is allowed in the matrix–vector products.

1. Introduction

The generalized minimum residual (GMRES) method of Saad and Schultz [1] is a popular technique for solving nonsymmetric linear systems of the form

$$Ax = b \tag{1}$$

where A is a nonsingular, large sparse matrix of dimension n, often with irregular structure. Given an initial estimate x_0 with corresponding residual vector $r_0 = b - Ax_0$, GMRES uses the Arnoldi process to build an orthogonal basis of the Krylov subspace

$$\mathcal{K}_m = \text{Span}\{r_0, Ar_0, A^2r_0, \ldots, A^{m-1}r_0\}.$$

The Arnoldi process leads to a relationship of the form

$$AV_m = V_{m+1}\overline{H}_m = V_mH_m + h_{m+1,m}v_{m+1}e_m^T, \tag{2}$$

where $V_m = [v_1, v_2, \ldots, v_m]$ is a basis of $\mathcal{K}_m$ and $H_m = V_m^TAV_m$ is upper Hessenberg, while $\overline{H}_m$ is H_m augmented with $h_{m+1,m}e_m^T$ in its last row. Here e_m denotes the mth column of the identity matrix I of appropriate size depending on its context. In exact arithmetic, GMRES converges in at most n steps, i.e., if $m = n$. However, because n is large the amount of

* Corresponding author.
E-mail addresses: roger.b.sidje@ua.edu (R.B. Sidje), newinkles@crimson.ua.edu (N. Winkles).

0377-0427/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.cam.2010.09.022

storage and the cost of orthogonalization in the Arnoldi process make it impractical to set $m = n$ and so restarting is used and $m \ll n$ in practice, which is why we will understand GMRES as meaning restarted GMRES unless the distinction is made explicit.

Variants of GMRES have been proposed, including most recently the inexact approach considered in this study. This approach uses approximate matrix–vector products [2,3], and it has been shown to work well even in some regularization applications where the coefficient matrix is singular [4]. To make the difference clear, we will call exact GMRES the classical method and this is not meant to imply exact arithmetic. The general principle of the inexact variant is to perform the matrix–vector product approximately as

$$Av \approx (A + E)v,$$

where E models some error matrix that varies with each product. Such inexactness (or relaxation) has been motivated by applications where the matrix is not known exactly or it is too expensive to apply. Research, so far, has focused on the modification that will decrease the workload per iteration without significantly affecting the accuracy. But relaxing the accuracy per iteration is liable to increase the number of iterations, thereby increasing the overall execution time, which could potentially end up greater than that of the exact GMRES should there be not enough savings in the matrix–vector products to compensate for the overhead of the extra iterations, if any. A related concern is that implementing a relaxation strategy may come at extra cost too. As we deal with large matrices over possibly thousands of steps, the inherent overhead of a given relaxation strategy may itself build up to a sizable proportion that has to be offset as well.

Our study will investigate the actual CPU time, which is an aspect that has received less attention so far. Such information is of vital importance to practitioners who need to decide whether switching their workflow to the inexact approach is worth the effort and the risk that might come with it. Also, regardless of the savings in computational time, practitioners are concerned about the robustness and reliability of newer solution techniques aimed at replacing what they have been accustomed to. Hence should a method fail to produce accurate enough answers, a trustworthy feedback (e.g., a residual) should be provided. But the difficulty is compounded in the inexact scheme by the so-called *residual gap* that we will discuss later. To assess all the aspects above with respect to inexact GMRES, we have performed extensive numerical experiments to gauge the effectiveness of the inexact scheme and its suitability for use in addressing certain problems, depending on how much inexactness is allowed in the matrix–vector products.

We developed three implementations to obtain our results:

- An optimized FORTRAN90 code based on SLAP (Sparse Linear Algebra Package [5]), where we focused on adapting the matrix–vector product to be inexact, based on a user-specified tolerance. Details on this will be given later. This represents the typical way that practitioners might proceed in changing their existing workflow. The timings reported throughout this study come from this optimized version.
- A separate instrumentation FORTRAN90 code based on the same SLAP code as above. In this code, we computed and captured all the additional debugging and instrumentation information aimed at allowing us to monitor the inexact solver without skewing the execution times to be reported. It is with this version that we computed the residual gap and the data used in the various tables and plots and reported in this study.
- A MATLAB implementation that we used to check the correctness of our changes to the SLAP code. While not as efficient as the compiled FOTRAN90 codes, it provided a convenient interface for fast prototyping and confirmation of the results reported in this study.

The organization of the paper is as follows. Section 2 summarizes the inexact method including some theoretical results pertaining to the impact of inexact matrix–vector products on the true residual. In Section 3 we describe our strategies for allowing the inexactness to percolate through the computations on the basis of a user-specified tolerance. In Section 4, we conduct a detailed series of experiments on both a desktop PC and a supercomputer as part of our performance analysis, leading in particular to the observation that for small problems ($n \leq 10{,}000$), the time that it takes to run either variant is very short, making the inexact variant not worth the porting effort in such cases. In Section 5 we provide concluding remarks.

2. The inexact GMRES algorithm

It is assumed that the reader is familiar with Krylov subspace techniques, or can consult for example [6,1] for more detailed information and background on the classical GMRES. In recent years there has been an interest in transitioning from exact matrix–vector products to inexact matrix–vector products in GMRES, either out of necessity or deliberately, as part of a trading of accuracy for speed. Results have been quite promising thus far [2,3,7]. Algorithm 1 shows a pseudo-code for the inexact variant in which as we noted earlier the inexact matrix–vector operation is modeled as

$$w_k := (A + E_k)v_k.$$

If we set $E_k = 0$ in the algorithm, we recover the classical GMRES. The foremost implication of such a relaxation of the matrix–vector product is that the classical Arnoldi relationship (2) no longer holds. Rather, we now have (in the notation of [7])

$$(A + \mathcal{E}_m)V_m = V_{m+1}\overline{H}_m, \qquad \mathcal{E}_m = \sum_{k=1}^{m} E_k v_k v_k^T, \tag{3}$$

or alternatively (in the notation of [3])

$$AV_m + F_m = V_{m+1}\overline{H}_m, \qquad F_m = \mathcal{E}_m V_m = [E_1 v_1, \ldots, E_m v_m], \tag{4}$$

where these equalities exploit the noteworthy fact that the basis V_m remains orthonormal. As noted in [7], the relation (3) makes it clear that V_m is a basis for a subspace spanned by the elements of a Krylov sequence obtained by some perturbation of the matrix A, where the perturbation is updated at each step. From theoretical and experimental evidence, the method can withstand cases where the norm of the perturbation $\mathcal{E}_m$ grows very large, and in effect the greater the iteration number k, the larger the error E_k allowable in the matrix–vector product.

ALGORITHM 1: Inexact GMRES

Choose an initial estimate x_0 and compute $r_0 := b - Ax_0$;
repeat
  $\beta := \|r_0\|_2$; $v_1 := r_0/\beta$;
  for $k = 1, 2, \ldots, m$, **do**
    $w_k := (A + E_k)v_k$;
    for $i = 1, 2, \ldots, k$ **do**
      $h_{ik} := v_i^T w_k$;
      $w_k := w_k - h_{ik} v_i$;
    end
    $h_{k+1,k} := \|w_k\|_2$;
    if $h_{k+1,k} \approx 0$ **then**
      $m := k$;
      break ;
    end
    $v_{k+1} := w_k / h_{k+1,k}$;
  end
  Define the $(m+1) \times m$ Hessenberg matrix $\overline{H}_m = (h_{ik})_{1 \le i \le m+1, 1 \le k \le m}$;
  Solve the least squares problem $\min \|\beta e_1 - \overline{H}_m y_m\|_2$ for y_m ;
  Set $x_0 \leftarrow x_m := x_0 + V_m y_m$ and $r_0 \leftarrow r_m := b - Ax_m$;
until convergence ;

Ideally, we look to reduce the number of operations, in order to make the CPU time smaller while keeping the same accuracy. Our numerical results in Section 4 show that provided that the distribution of non-zero elements per column is suitable, our times for the inexact matrix–vector products are less than that for the exact matrix–vector products. However, we observed that a reduced time does not come with the guarantee that the solution is as accurate as claimed by the inexact solver. This is because there is a discrepancy between the true residual that exact GMRES would provide and the estimated residual computed by inexact GMRES. This is the so-called *residual gap* issue and we shall start by discussing this issue in greater detail, as it is important in practice. Our discussion gives some additional insights into the issue and our experiments include a few instructive cases where the estimated residual of inexact GMRES does not have the very valuable monotonic property that the true residual of exact GMRES has. We shall even see a notorious case later where the computed residual increases dramatically.

2.1. The residual gap

Both the exact and inexact GMRES methods (cf. Algorithm 1) use the update formula

$$x_m = x_0 + V_m y_m,$$

where x_0 is the initial guess and y_m is obtained by solving the least squares problem

$$y_m = \arg\min_y \|\beta e_1 - \overline{H}_m y\|_2, \qquad r_0 = b - Ax_0, \quad \beta = \|r_0\|_2.$$

Associated with the new iterate x_m is the *true* residual

$$r_m = b - Ax_m,$$

but it is well known that it is not computed in this way in highly optimized production codes—neither by the exact GMRES nor by the inexact variant, where in this latter case, it is better not to assume that the matrix–vector product Ax_m can be performed exactly at will (although we shall see later that a *completely* inexact method is hard to achieve). Obtaining meaningful bounds on the true residual under such restrictions in the inexact setting has been analyzed in [7] as we summarize here. Recall (3) and (4); the inexact GMRES method ends up with (the norm of) the *computed* residual

$$\tilde{r}_m = r_0 - V_{m+1}\overline{H}_m y_m,$$

while the true residual satisfies

$$\begin{aligned} r_m &= b - Ax_m = b - A(x_0 + V_m y_m) \\ &= \tilde{r}_m + [E_1 v_1, \ldots, E_m v_m] y_m. \end{aligned} \tag{5}$$

In the exact case where all the $E_k = 0$, the two residuals coincide, $\tilde{r}_m = r_m$, and it is well known that the norm can be retrieved economically from the Givens rotations that are created to reduce the Hessenberg matrix $\overline{H}_m$ to triangular form. In the inexact case, however, there is an unknown *residual gap* between the true residual and the computed residual, and from (5) this gap satisfies

$$\delta_m = \|r_m - \tilde{r}_m\|_2 = \|[E_1 v_1, \ldots, E_m v_m] y_m\|_2. \tag{6}$$

The inexact solver still reports $\|\tilde{r}_m\|_2$ as the estimate of the residual, but the gap between this estimate and the true residual is not obvious, prompting a reservation about the reliability of the final solution. Clearly, monitoring δ_m, or its relative counterpart, $\delta_m / \|A\|_2 \|y_m\|_2$, which is more meaningful as a backward error criterion (see [7]), becomes necessary. This is discussed next.

2.2. Monitoring the inexact matrix–vector product

Letting $\eta_k^{(m)}$ be the kth component of y_m, i.e.,

$$y_m = (\eta_1^{(m)}, \eta_2^{(m)}, \ldots, \eta_m^{(m)})^T,$$

if follows from (5) that $\|r_m\|_2 \le \|\tilde{r}_m\|_2 + \delta_m$ with

$$\delta_m \le \sum_{k=1}^{m} |\eta_k^{(m)}| \cdot \|E_k v_k\|_2 \le \sum_{k=1}^{m} |\eta_k^{(m)}| \cdot \|E_k\|_2, \tag{7}$$

and we note in passing that this bound is observed to be often sharper than using

$$\delta_m = \|[E_1 v_1, \ldots, E_m v_m] y_m\|_2 \le \|[E_1 v_1, \ldots, E_m v_m]\|_2 \cdot \|y_m\|_2.$$

It also explains why the error in the matrix–vector product can be large provided that either of the products $|\eta_k^{(m)}| \cdot \|E_k v_k\|_2$ or $|\eta_k^{(m)}| \cdot \|E_k\|_2$ is small. But a major difficulty here is that y_m is only available at the mth iteration and so, as the iterations unfold, it is not clear how to determine the maximum allowable error at the kth iteration that can ultimately guarantee that the upcoming gap δ_m at the mth iteration stays within a prescribed tolerance. To address this problem, Simoncini and Szyld [7] attempted to bound $|\eta_k^{(m)}|$ in terms of quantities that are available at the kth iteration. They showed that the following inequality holds:

$$|\eta_k^{(m)}| \le \frac{1}{\sigma_{\min}(\overline{H}_m)} \|\tilde{r}_{k-1}\|_2, \tag{8}$$

where $\sigma_{\min}(\overline{H}_m) = \|(\overline{H}_m^T \overline{H}_m)^{-1}\|_2^{-1/2}$ is the smallest singular value of $\overline{H}_m$. Therefore if we can somehow enforce the extent of the inexactness in the matrix–vector products to satisfy

$$\|E_k\|_2 \le \frac{\sigma_{\min}(\overline{H}_m)}{m} \frac{\varepsilon}{\|\tilde{r}_{k-1}\|_2}, \quad k = 1, \ldots, m, \tag{9}$$

for a prescribed ε, we will automatically get from (7)

$$\delta_m = \|r_m - \tilde{r}_m\|_2 \le \varepsilon.$$

The bound (9) explains why the greater the iteration number k is, the smaller $\|\tilde{r}_{k-1}\|_2$ gets (in those cases where there is no stagnation), and so the larger the allowable error E_k in the matrix–vector product could be. However, while all the intermediate computed residuals $\|\tilde{r}_{k-1}\|_2$ are readily available via Givens rotations as we hinted earlier, the difficulty now has shifted to anticipating what $\sigma_{\min}(\overline{H}_m)$ might end up being, prior to reaching the mth iteration. This remains an open question and it is suggested in [7] that one use an estimate, e.g., $\sigma_{\min}(\overline{H}_m) \approx \sigma_{\min}(A)$, the smallest singular value of A. This issue is discussed further in [8], but their bounds are still not ideal in our case where $n \gg 1$ since we deal with very large problems. Another practical difficulty is that it may not be easy to measure and fine-tune $\|E_k\|_2$ (or more likely an approximation thereof). Thus the cost of monitoring a relaxation strategy may build up to a non-negligible amount when dealing with very large matrices over thousands of steps.

Our study is motivated by the aspects above. We will measure the actual computational time and illustrate how the gap evolves in representative examples. We will describe a simple but effective strategy for performing the relaxation based on controlling $\|E_k v_k\|_2$ in (7) using a parameter called *droptol*. Conceptually, our strategy amounts to dropping a number of elements in the matrix, therefore making the matrix more sparse. This causes E_k to be the elements that were dropped and we can pick a *droptol* that restricts $\|E_k v_k\|_2$ in a number of ways. This is discussed in Section 3 but we would like to first raise other aspects that are worth keeping in mind in an inexact setting, especially for those wishing to use inexact GMRES in their applications.

2.3. Restart and loss of optimality

Since inexact versions of GMRES are likely to come from modifying existing exact codes, there need to be a few extra adjustments beside the inexact matrix–vector product in order for the data to be accurately reported using the theory in [7] that was summarized above. The derivation of the gap assumes that the initial residual r_0 is exact (or at the very least within the tolerance desired on the solution; see the discussion at the end of [7, Section 3]). The theory does not hold if this assumption is overlooked. Yet, this is precisely what happens at restart if an exact code is converted to an inexact code by merely relaxing the matrix–vector product routine. A simple way to understand what happens is to recall that the exact GMRES method seeks to solve

$$\min \|b - Ax_m\|_2 = \min \|b - A(x_0 + z_m)\|_2 = \min \|r_0 - Az_m\|_2,$$

which yields $z_m = V_m y_m$ with $y_m = \overline{H}_m^{\dagger}(\beta e_1)$, $\beta = \|r_0\|_2$, and $\overline{H}_m^{\dagger}$ being the pseudo-inverse of $\overline{H}_m$. Thus if we substitute r_0 with a very distant $\tilde{r}_0 = b - (A + E_0)x_0$ where the perturbation E_0 is large, we end up solving

$$\min \|\tilde{r}_0 - A\tilde{z}_m\|_2$$

for some $\tilde{z}_m$ that ends up distant from the intended z_m. This is the effect that could happen in the inexact setting, and understandably, analyzing the residual gap serves little purpose in such a situation. It is easy to overlook this when converting an exact code to be inexact because, as we mentioned before, the exact GMRES commonly implements the shortcut method $r_0 \leftarrow r_m := r_0 - V_m\overline{H}_m y_m$ at restart, and we explained earlier that this does not match the true residual in the inexact case. Thus a completely inexact code is hard to achieve because the need for the true residual at restart is inevitable. The importance of an accurate residual at restart has also been discussed in the context of the Newton–GMRES algorithm for solving nonlinear problems [9]. At the very least, we recommend computing the true residual upon completion to double-check the accuracy of the solution when the inexact solver claims convergence. In our experiments, we changed the restart of the SLAP code to compute the residual explicitly using a full blown matrix–vector product

$$r_0 \leftarrow r_m := b - Ax_m.$$

Even when using the exact computation of the residual at restart there are still some instances where the behavior of inexact GMRES can markedly differ from what practitioners have come to expect with exact GMRES, notably the core tenet that the norm of the true residual never increases in exact GMRES. The opposite can happen in inexact GMRES, manifested by an erratic sequence of the norms of its corresponding *true* residual. This means that inexact GMRES loses the optimality of exact GMRES. To understand why, assume that y_m^{opt} is the vector that would have achieved the exact minimization of the true residual as

$$\|r_m^{\text{opt}}\|_2 = \|r_0 - AV_m y_m^{\text{opt}}\|_2 = \min_y \|r_0 - AV_m y\|_2$$

upon constructing the basis V_m through the inexact process (4). We have actually replaced the exact minimization of the residual with an approximation that can be described as

$$\begin{aligned} y_m^{\text{opt}} &= \arg\min_y \|r_0 - AV_m y\|_2 \\ &= \arg\min_y \|(r_0 - V_{m+1}\overline{H}_m y) + F_m y\|_2 \\ &\approx \arg\min_y \|r_0 - V_{m+1}\overline{H}_m y\|_2 \\ &= y_m. \end{aligned}$$

Thus even if r_0, y_m and the corresponding true residual r_m are computed accurately, there is no guarantee that y_m (resp. r_m) will be close to y_m^{opt} (resp. r_m^{opt}). A small gap $\|F_m y_m\|_2$ does not rule out this issue either because it could just be that y_m is almost in the null space of F_m with r_m already very different from r_m^{opt}. This latter scenario is expected to be rare however.

Again the point is not that inexact GMRES is not minimizing the residual due to the way its basis was constructed, but rather that it is not even getting the minimizer of the residual from within its own basis, and this means that inexact GMRES should be viewed much like a quasi-minimum residual method. Our experiments include illustrative plots of $\|y_m^{\text{opt}} - y_m\|_2$ and $\|r_m^{\text{opt}} - r_m\|_2$. See Figs. 12–14.

Finally, we should clarify that the *computed* residual can also have an erratic behavior but in a more subtle way. Indeed the computed residual always remains monotonic until the maximum size of the basis is reached because

$$\|\tilde{r}_0\|_2 \geq \cdots \geq \|\tilde{r}_k\|_2 = \min_y \|r_0 - V_{k+1}\overline{H}_k y\|_2 \geq \cdots \geq \|\tilde{r}_m\|_2 = \min_y \|r_0 - V_{m+1}\overline{H}_m y\|_2.$$

But since the computed residual is reset to the true residual at restart, this has the effect of creating an impulse if the gap is perceptible, giving the appearance of an erratic behavior. If an implementation chooses not to reset the computed residual to the true residual at restart or if the gap remains negligible throughout the restarts, then the computed residual will remain monotonic. In the case where the computed residual is not reset, the danger is that it may end up small by monotonicity when in reality the true residual is large, misleading the user into accepting an inaccurate solution. This is why we recommend at least computing a sufficiently accurate residual at completion to confirm the accuracy of the final solution.

3. Practical implementation

We saw that the effectiveness of the inexact scheme is tied to the residual gap

$$F_m y_m = [E_1 v_1, \ldots, E_m v_m] y_m.$$

We implemented two strategies for the inexact matrix–vector product by dropping operations that would otherwise involve elements of very small magnitude. In this section we discuss how the errors introduced by our schemes relate to the theoretical bounds seen earlier.

3.1. Inexact matrix–vector products

In the first strategy, writing

$$A = [a_1|\ldots|a_n], \quad v = (\mu_1, \ldots, \mu_n)^T,$$

where the $a_j \in \mathbb{R}^n$ and $\mu_j \in \mathbb{R}$ for $j = 1, \ldots, n$, we perform our matrix–vector product in the following column-oriented way:

$$Av = [a_1|\ldots|a_n] \begin{bmatrix} \mu_1 \\ \vdots \\ \mu_n \end{bmatrix} = \sum_{j \in J} \mu_j a_j + \sum_{j \in J^c} \mu_j a_j$$

where J is the set of indices such that $|\mu_j| > droptol$ and J^c is its complement, with *droptol* being a user-specified tolerance. We then use the approximation

$$Av \approx \sum_{j \in J} \mu_j a_j, \tag{10}$$

which conceptually amounts to dropping selected columns of the matrix A and entails the error

$$Ev = \sum_{j \in J^c} \mu_j a_j,$$

and so for any norm $\|\cdot\|$, since J^c is the set of indices such that $|\mu_j| \leq droptol$, we have

$$\|Ev\| \leq \sum_{j \in J^c} |\mu_j| \cdot \|a_j\| \leq \sum_{j \in J^c} droptol \cdot \|a_j\| \leq droptol \cdot |J^c| \cdot \|A\|, \tag{11}$$

where $|J^c|$ denotes the cardinality of J^c. A large $|J^c|$ would imply the presence of many small μ_j (in magnitude), meaning that several columns have been dropped in the matrix–vector product, thereby leading to substantial savings provided that these columns had many non-zero elements.

In the context of inexact GMRES for a large sparse problem, the Arnoldi process computes

$$w_k = (A + E_k) v_k$$

where $v_k = (\mu_1^{(k)}, \ldots, \mu_n^{(k)})^T$ is a normalized basis vector with $\|v_k\|_2 = 1$. It is very likely that some components are very small since $\sum_{j=1}^{n} (\mu_j^{(k)})^2 = 1$ with $n \gg 1$. Hence, the larger the n, the smaller the components (in magnitude) and the more of them there will be, and in the light of the previous discussion, this will result in substantial savings in the cost of our inexact matrix–vector product provided that the corresponding columns of the matrix that we drop have many non-zero elements.

As $|J^c|$ grows large and brings more savings, the error bound (11) may become too loose to be useful. This is because it caters for any norm and so is too general. A tighter bound is summarized below.

Theorem 1. *The residual gap resulting from the inexact GMRES method based on the relaxed matrix–vector product (10) satisfies*

$$\frac{\|F_m y_m\|_\infty}{\|A\|_\infty \|y_m\|_1} \leq droptol.$$

Proof. First note that the error from (10) can be bounded as

$$\|Ev\|_\infty \leq droptol \cdot \|A\|_\infty, \tag{12}$$

which gives a clearer picture of the actual accuracy being traded. This bound comes from the fact that we can write $Ev = Av^c$ where v^c denotes the vector such that the jth component is the same as μ_j if $j \in J^c$; thus $|e_j^T v^c| = |e_j^T v| = |\mu_j| \leq droptol$, and is 0 otherwise. Recall that $y_m = (\eta_1^{(m)}, \eta_2^{(m)}, \ldots, \eta_m^{(m)})^T$; we can infer from (12) that

$$\|F_m y_m\|_\infty \leq \sum_{k=1}^{m} |\eta_k^{(m)}| \cdot \|E_k v_k\|_\infty \leq droptol \cdot \|A\|_\infty \sum_{k=1}^{m} |\eta_k^{(m)}| = droptol \cdot \|A\|_\infty \|y_m\|_1,$$

which produces the relative form stated in the theorem. □

Recall (8); we can also conclude that

$$\|F_m y_m\|_\infty \le \frac{droptol \cdot \|A\|_\infty}{\sigma_{\min}(\overline{H}_m)} \sum_{k=1}^{m} \|\tilde{r}_{k-1}\|_2$$

and this shows that

$$\|F_m y_m\|_\infty \le \epsilon \|A\|_\infty$$

where

$$\epsilon \le droptol \cdot \|y_m\|_1 \le \frac{droptol}{\sigma_{\min}(\overline{H}_m)} \sum_{k=1}^{m} \|\tilde{r}_{k-1}\|_2,$$

where we can make $\epsilon \in (0, 1)$ by choosing a sufficiently small *droptol*. Therefore we have now provided an error bound for our inexact scheme using the parameter ϵ and $\|A\|_\infty$ where ϵ depends on values from our iterations and so cannot be pre-determined for all problems exactly. A similar technique can be seen in [7].

Our first strategy applied a *droptol* to the components of a basis vector v in the inexact product $w \approx Av$. However this strategy did not weight the components of v, where $v = (\mu_1, \ldots, \mu_n)^T$, with the matrix $A = [a_1 | \ldots | a_n]$. When we drop $\mu_j a_j$ in our matrix–vector product with $|\mu_j| \le droptol$, it is possible, on one hand, for $\|\mu_j a_j\|_\infty$ to be large enough to contribute significantly to the error; on the other hand, if the column a_j only has small elements, since $\mu_j < 1$, it is possible for $\mu_j a_j$ to shrink enough to become negligible even if $|\mu_j| > droptol$. Therefore a second strategy is to use a weighted threshold where the combination of $\mu_j a_j$ is omitted in the matrix–vector product if

$$|\mu_j| \cdot \|a_j\|_\infty \le droptol.$$

In this case, the matrix–vector product becomes

$$Av = \sum_{j \in J_w} \mu_j a_j + \sum_{j \in J_w^c} \mu_j a_j$$

where J_w is the set of indices such that $|\mu_j| \cdot \|a_j\|_\infty < droptol$. As before we use the approximation

$$Av \approx \sum_{j \in J_w} \mu_j a_j, \tag{13}$$

which has the error

$$Ev = \sum_{j \in J_w^c} \mu_j a_j,$$

but this time

$$\|Ev\|_\infty \le \sum_{j \in J_w^c} |\mu_j| \cdot \|a_j\|_\infty \le \sum_{j \in J_w^c} droptol \le |J_w^c| \cdot droptol.$$

Thus this scheme uses *droptol* in a way that takes into consideration the entries of our matrix whereas the previous scheme did not. We can express the bound without $|J_w^c|$ in the following way.

Theorem 2. *Let $A_w(v)$ be the matrix derived from $A = [a_{ij}]$ and $v = (\mu_1, \ldots, \mu_n)^T$ such that its ijth entry is $\mu_j a_{ij}$ if it is less than droptol in magnitude and* 0 *otherwise. Then the residual gap resulting from the inexact GMRES method based on the relaxed matrix–vector product (13) satisfies*

$$\frac{\|F_m y_m\|_\infty}{\|y_m\|_1} \le \max_{1 \le k \le m} \|A_w(v_k)\|_\infty \le droptol \cdot \|\mathbb{1}_A\|_\infty,$$

where $\mathbb{1}_A$ is the sparsity pattern of A, with 1 *at the non-zero positions and* 0 *elsewhere.*

Proof. It follows from the definition of $A_w(v)$ that the error due to (13) satisfies

$$Ev = \sum_{j \in J_w^c} \mu_j a_j = A_w(v)e, \quad e = (1, \ldots, 1)^T,$$

with $|e_i^T A_w(v) e_j| = |\mu_j a_{ij}| \le droptol$ if $j \in J_w^c$ and 0 otherwise. And so

$$\|Ev\|_\infty \le \|A_w(v)\|_\infty.$$

Like with the previous theorem, we get

$$\begin{aligned} \|F_m y_m\|_\infty &\le \sum_{k=1}^{m} |\eta_k^{(m)}| \cdot \|E_k v_k\|_\infty \le \sum_{k=1}^{m} |\eta_k^{(m)}| \cdot \|A_w(v_k)\|_\infty \\ &\le \|y_m\|_1 \cdot \max_{1 \le k \le m} \|A_w(v_k)\|_\infty. \end{aligned}$$

Since entries of $A_w(v)$ are less than *droptol* in magnitude, the stated inequality follows. □

Table 1
Characteristics of the problems (*symmetric problem, but symmetry is not exploited here). $\|A\|_2^{\text{est}}$ was estimated by the function call *normest(A)* in MATLAB. These matrices come from The University of Florida Sparse Matrix Collection [11].

Matrix	n	$nz(A)$	Cond. number	$\|A\|_2^{\text{est}}$	$\|A\|_\infty$
cage13	445,315	7,479,343		6.6732e+02	1.4051
cage14	1,505,785	27,130,349		1.2271e+03	1.3333
cfd1*	70,656	1,828,364	1.3351e+006	1.2702e+03	8.9202
cfd2*	123,440	3,087,898		1.8770e+03	8.8104
circuit_4	80,209	307,625	5.7143e+011	1.4741e+05	5.3429e+03
epb3	84,617	463,625	7.4546e+004	9.5144e+01	6.7590e−01
finan512*	74,512	596,992	9.8391e+001	2.6496e+03	6.3233e+01
hcircuit	105,676	513,120	6.6598e+005	2.0228e+03	8.8179e+01
pwtk*	217,918	11,634,424		3.6177e+09	1.8231e+08
stomach	213,360	3,021,648		5.1782e+02	4.0785
torso3	259,156	4,429,042		6.7086e+02	1.0936e+01

The matrix $A_w(v)$ symbolizes all the elements that were not included in the weighted matrix–vector product (13), with these elements further scaled down by the corresponding μ_j. If the matrix A already had small elements, it is possible for $A_w(v)$ to capture many of them (which means many savings in the matrix–vector product). But if A had large elements, there may not be that much saving, unless $\mu_j \ll 1$ and *droptol* is not stringent. This is different from the first scheme where a large n assured many small μ_j, which then virtually assured savings. The sparsity pattern of $A_w(v)$ is a subset of that of A and we have $\|A_w(v_k)\|_\infty \leq \|A\|_\infty$, but *not* necessarily $\|A_w(v_k)\|_\infty \leq droptol \cdot \|A\|_\infty$.

Our dropping strategies are columnwise and so we converted the test matrices into the Compressed Column Storage (CCS) format, which is the most suitable compact format in our context. We implemented the weighted scheme by computing all the $\|a_j\|_\infty$ at the very beginning and re-using them throughout the computations. Trying to implement a finer strategy would mean scanning the entire matrix at each matrix–vector product, which will be very much prohibitive.

3.2. Savings

Let us denote the number of non-zero elements of the jth column of our matrix A by $nz(a_j)$. Therefore when we skip the product of $\mu_j a_j$ in either of our schemes, we are saving $nz(a_j)$ floating point operations. Now, considering the entire matrix, we are saving $\sum_{j\in J^c} nz(a_j)$ operations (substitute J^c with J_w^c as appropriate for the weighted scheme). Recall that this means that we skip $|J^c|$ columns. Consequently we can expect meaningful savings only if the values of $nz(a_j)$ are not too small on average and if $|J^c| \gg 1$. This explains why when we considered cases where $n \leq 10{,}000$, the savings were not that significant because $|J^c|$ was not that large when one takes account of today's very fast CPUs. Moreover, even for larger problems, if the $nz(a_j)$ are small on average, we do not see any savings worth using inexact matrix–vector products for, because the cost of deciding whether or not to use a column (using a conditional if-check) must be compensated for by the savings generated by that column, and this can only happen if $nz(a_j) \gg 1$. Therefore we will focus our experiments on large problems in which we additionally have $nz(a_j) \gg 1$. We provide histograms of $nz(a_j)$ in our results to give the reader a sense of the distribution of the non-zeros. The importance of this consideration was also observed in [10].

4. Numerical results

All calculations were done on a supercomputer hosted at the Alabama Supercomputer Center (ASC) in Huntsville, Alabama. The ASC hosts units of two types: the Dense Memory Cluster (DMC) and the SGI Altix. At the time of our experiments the DMC has 1256 CPU cores and 6176 GB of distributed memory. The machine is physically configured as a set of eight CPU core SMP boards. Sixty of the nodes have 64 GB of memory. Twenty nodes have 3.0 GHz dual-core AMD Opteron processors. Forty nodes have 2.3 GHz quad-core AMD Opterons. Ninety-six nodes have 2.26 GHz Intel quad-core Nehalem processors. Our executions were done in batch mode on the DMC and the batch system automatically assigns the job to a CPU core depending on the queue. Table 1 lists the characteristics of the problems used in our experiments.

Our results are detailed in the Appendix. We set the true solution $x = (1, 0, \ldots, 0, 1)^T$ and generate the right side $b = Ax$ for each problem. Hence we can test the true error $\|x - x_m\|_2$ on completion. For ease of presentation of the results, we use the index m (as in x_m) to denote the final iterate, even though there could have been intermediate restarts and/or the convergence could have occurred midway during the iterations, as is often the case with GMRES. In the SLAP solver, we set the restart parameter $m = 50$ and request an accuracy tolerance $tol = 10^{-6}$ everywhere. Convergence means that $\|\tilde{r}_m\|_2/\|b\|_2 \leq tol$. We set 2500 as the maximum overall number of iterations (the current number of iterations is not reset at restart and so is not bounded by m).

For each set of results, we give:

- The histogram of the distribution of non-zero entries per column of the matrix under consideration. Thus with $nz(a_j)$ indicating the number of non-zeros in the column, #cols represents the number of columns of the matrix that have this indicated number of non-zeros. This preparatory information is very useful because it allows one to predict the potential

savings in the inexact matrix–vector products as we explained earlier. If the histogram shows that most columns have few non-zero entries then we can expect no significant difference, as far as time is concerned, to motivate using the inexact scheme. This is why we focused on large problems where the $nz(a_j)$ are not too small. We observed in general that if a high number of columns have more than 10 non-zero entries, the matrix will be more likely to make the inexact scheme worthwhile on our testing platform. On the basis of such preparatory information and depending on their environment, practitioners may decide whether our inexact matrix–vector strategies are suitable for their applications in terms of savings in computational time.

- The data collected from the execution, with the CPU time coming from the optimized code and the debugging data from the separate instrumentation code as we described at the beginning.
  - *droptol*: relaxation parameter in the inexact matrix–vector as detailed in Section 3.1.
  - $\|x - x_m\|_2/\|x\|_2$: true error in the final computed solution.
  - $\|r_m\|_2/\|b\|_2$: true residual on completion.
  - $\|\tilde{r}_m\|_2/\|b\|_2$: computed residual on completion (note that our implementation reset $\tilde{r}_0$ to the true residual r_0 at restart positions).
  - δ_m: residual gap on completion, $\delta_m = \|r_m - \tilde{r}_m\|_2$.
  - Runtime (s): CPU time in seconds.
  - #Iter: number of iterations (not to be confused with the number of restarts). It is incremented every time a new approximated solution is possible (even if it is not computed explicitly—as is the case with GMRES) and is not reset at restart. Convergence failure is declared if #Iter reaches 2500 without the computed residual meeting the convergence criteria.
  - Savings: number of operations saved. This is a counter that is incremented with $nz(a_j)$ if the jth column is skipped in our inexact matrix–vector product.
- Graphical plots of the history of the residual norm (scaled by $\|b\|_2$) when using either the unweighted matrix–vector product and the weighted alternative.

For a few problems (Figs. 12–14) we plot the history of some of their characteristics throughout the entire run with $droptol = 10^{-3}$, including the residual gap δ_m, as well as $\|y_m^{\text{opt}} - y_m\|_2$ and $\|r_m^{\text{opt}} - r_m\|_2$ that give an idea of the quasi-minimization that takes place within inexact GMRES. Fig. 13 is an example where inexact GMRES (with $droptol = 10^{-3}$) claimed early convergence (after only six iterations), whereas it appears clear from the history of its attributes in Fig. 5 that the residual gap was not within the requested tolerance.

We run the experiments with *droptol* ranging from 10^{-3} to 10^{-10}, and the results show that only when $droptol = 10^{-3}$ does the inexact scheme show most of its perceptible accuracy difficulties. This shows that the inexact scheme is fairly robust as far as accuracy is concerned. The smaller *droptol* is, the closer inexact GMRES gets to the exact GMRES. It is remarkable how in many problems the plots of the residual norm became almost indistinguishable at $droptol = 10^{-5}$ and lower.

We did not vary the accuracy per iteration, but the fact that we use a wide range of *droptol* provides similar evidence that starting with $droptol = 10^{-10}$ and relaxing gradually within the iterations would have worked equally well, as predicted by the theory and experiments by previous authors. We did not also consider preconditioners as they would have introduced other parameters that would have increased the number of experiments beyond our already large set.

Problems **cage13** and **cage14** show identical behaviors although the latter is twice as big as the former. The same can be said about **cfd1** and **cfd2**. Problem **ebp3** is peculiar in that it makes the inexact scheme diverge when the weighted matrix–vector product is used. This is due to the fact that its matrix is of small norm, $\|A\|_\infty = 0.68$, and as we discussed in Section 3.1, the bigger $droptol = 10^{-3}$ causes many elements to fall into the error term of the matrix–vector, making it too inexact. Contrast this with **pwtk** for which the matrix is of large norm, $\|A\|_\infty = 1.82 \cdot 10^8$, indicating the presence of large elements in the matrix. In this case, it is the unweighted matrix–vector that suffers because it only checks the operand vector and skips columns of the matrix without regard to the magnitude of its elements. The large perturbation that results from this matrix leads to a larger residual gap, which explains the spikes at restart positions that we see in the plot. Spikes appear in general when the computed residual is reset to the true residual at restart positions, but may be imperceptible if the residual gap is too small. They can be seen on some of the other plots as well. Problem **pwtk** is also a case where none of the schemes converge within the allotted maximum number of iterations, but their residual decreases quite well.

Another example that stands out is **circuit_4**, where the inexact scheme returns a highly optimistic residual when $droptol = 10^{-3}$ under either of the matrix–vector strategies, suggesting that it is a matrix where the perturbations should not be too big at the beginning of the iterations. There are other instances where the computed residual is too optimistic; for example in Table 5 for **cage14**, the gap is large when $droptol = 10^{-3}$. Protection against such optimistic computed residuals requires at least computing an accurate residual on completion so as not to mislead the user into thinking that the returned solution is as accurate as claimed by the inexact solver.

We noted earlier that the computed residual need not be monotonic. Problem **hcircuit** is a clear illustration of this under the weighted matrix–vector product.

In results not reported here, we experimented with not computing the restart residual accurately. We saw that it is very risky to adopt this approach because the inexact scheme can subsequently claim convergence when actually this has not been achieved. In such cases the final residual gap remains wide (but it should be recalled that the residual gap is not computed in an optimized setting and therefore users cannot expect to base their decision on its availability). Hence we recommend computing the restart residual at least as accurately as with the tolerance requested for the solution.

Table 2
Histogram of the distribution of non-zeros per column in **cage13**.

$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols
3	64	12	17,862	19	31,056	26	7444	33	882
5	2,312	13	28,844	20	25,348	27	5588	34	604
6	368	14	37,962	21	19,914	28	3070	35	604
7	5,604	15	30,248	22	19,032	29	2576	36	212
8	12,212	16	36,326	23	13,523	30	1510	37	196
9	4,652	17	35,556	24	13,136	31	845	38	32
10	16,548	18	30,752	25	8,507	32	914	39	24
11	30,988								

Table 3
Execution of **cage13** (first block: unweighted matrix–vector, second: weighted).

droptol	$\|x - x_m\|_2/\|x\|_2$	$\|r_m\|_2/\|b\|_2$	$\|\tilde{r}_m\|_2/\|b\|_2$	δ_m	Runtime (s)	#Iter	Savings
Exact	1.61e−06	4.77e−07	4.77e−07	0	2.32	18	0
10^{-3}	1.94e−02	1.71e−02	4.87e−07	1.66e−02	1.00	18	7742,498
10^{-5}	2.52e−05	2.28e−05	4.77e−07	2.21e−05	1.35	18	5978,936
10^{-6}	1.98e−06	1.25e−06	4.77e−07	1.11e−06	1.50	18	5072,962
10^{-8}	1.61e−06	4.77e−07	4.77e−07	2.40e−09	1.60	18	4273,275
10^{-10}	1.61e−06	4.77e−07	4.77e−07	3.53e−12	1.62	18	4145,407
Exact	1.61e−06	4.77e−07	4.77e−07	0	2.34	18	0
10^{-3}	3.70e−02	3.00e−02	5.26e−07	2.90e−02	1.00	18	7862,514
10^{-5}	7.18e−05	5.62e−05	4.74e−07	5.44e−05	1.34	18	6152,908
10^{-6}	2.58e−06	1.83e−06	4.77e−07	1.71e−06	1.49	18	5219,790
10^{-8}	1.61e−06	4.77e−07	4.77e−07	3.30e−09	1.62	18	4296,666
10^{-10}	1.61e−06	4.77e−07	4.77e−07	5.14e−12	1.64	18	4148,461

Overall, we can see that both of the inexact schemes are quite robust, leaving the savings only really dependent on the sparsity pattern of the problem and the CPU speed. The savings in floating point operations in many of the problems appear to be quite significant (ranging from several hundreds of thousands to several millions of operations), but with today's very fast CPUs, it takes a fair chunk of work for them to start affecting the CPU time. This is why even though the inexact GMRES with varying *droptol* is often as accurate as the exact GMRES while having substantial savings, their execution times remain close. It is also why for smaller matrices the time for the exact method and that for the inexact method were so close that there was no need to use an inexact method to help improve efficiency.

5. Conclusion

We discussed the inexact GMRES method and presented two strategies for relaxing the matrix–vector product based on a *droptol* parameter. Conceptually, our inexact strategies amount to dropping a number of elements in the matrix, therefore making it more sparse. We compared them with the exact scheme for a wide range of problems. Our extensive experiments showed that the inexact schemes are competitive with the exact scheme for very large problems, provided that their savings grow enough to become perceptible with today's very fast CPUs. We saw that accuracy was not the main concern with the inexact schemes, if implemented with care. In particular, the uncertainty with the residual gap can be virtually eliminated by computing the true residual at restart positions. Indeed, we used an instrumentation code and plotted the history of the residual norms and they became almost indistinguishable at reasonable *droptol*. Our analysis provided additional insight into the inexact GMRES method and our experiments provided further data that should assist practitioners in deciding whether switching to inexact GMRES is worthwhile for their applications.

Appendix A. Results for cage13

$n = 445{,}315 \quad nz = 7{,}479{,}343 \quad \|A\|_\infty = 1.41 \quad \|b\|_2 = 9.67 \cdot 10^1$ (see Tables 2, 3 and Fig. 1).

Appendix B. Results for cage14

$n = 1{,}505{,}785 \quad nz = 27{,}130{,}349 \quad \|A\|_\infty = 1.33 \quad \|b\|_2 = 9.58 \cdot 10^1$ (see Tables 4, 5 and Fig. 2).

Appendix C. Results for cfd1

$n = 70{,}656 \quad nz = 1{,}828{,}364 \quad \|A\|_\infty = 8.92 \quad \|b\|_2 = 2.15$ (see Tables 6, 7 and Fig. 3).

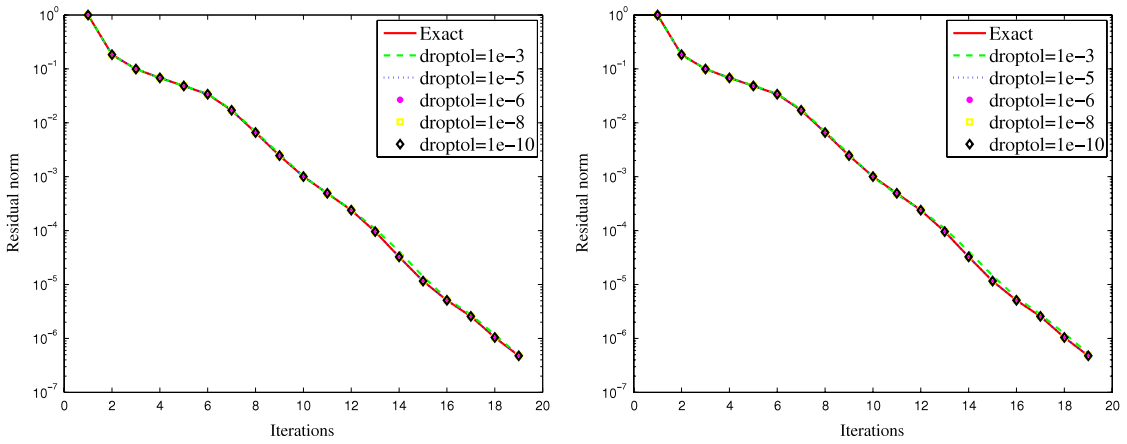

Fig. 1. Residual history of **cage13** (left: unweighted matrix–vector, right: weighted).

Table 4
Histogram of the distribution of non-zeros per column in **cage14**.

$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols
5	4,096	14	113,480	21	79,914	28	20,146	35	2869
7	12,800	15	96,400	22	77,754	29	15,354	36	1478
8	26,240	16	112,232	23	69,761	30	11,074	37	1358
9	10,688	17	117,689	24	48,500	31	7,627	38	502
10	42,368	18	105,818	25	42,577	32	6,110	39	396
11	79,888	19	105,709	26	37,646	33	4,257	40	72
12	45,088	20	100,980	27	23,578	34	2,648	41	48
13	78,640								

Table 5
Execution of **cage14** (first block: unweighted matrix–vector, second: weighted).

droptol	$\|x - x_m\|_2/\|x\|_2$	$\|r_m\|_2/\|b\|_2$	$\|\tilde{r}_m\|_2/\|b\|_2$	δ_m	Runtime (s)	#Iter	Savings
Exact	2.41e−06	8.78e−07	8.78e−07	0	7.90	17	0
10^{-3}	2.41e−02	2.09e−02	9.14e−07	2.00e−02	3.23	17	25,179,457
10^{-5}	5.27e−05	4.50e−05	8.78e−07	4.31e−05	4.32	17	20,321,953
10^{-6}	2.86e−06	1.67e−06	8.78e−07	1.35e−06	4.72	17	18,026,832
10^{-8}	2.41e−06	8.78e−07	8.78e−07	2.06e−09	5.17	17	15,177,001
10^{-10}	2.41e−06	8.78e−07	8.78e−07	3.18e−12	5.24	17	14,634,152
Exact	2.41e−06	8.78e−07	8.78e−07	0	7.72	17	0
10^{-3}	4.76e−02	3.70e−02	9.55e−07	3.54e−02	3.05	17	25,426,046
10^{-5}	1.42e−04	1.15e−04	8.80e−07	1.10e−04	4.14	17	20,739,576
10^{-6}	4.87e−06	3.50e−06	8.78e−07	3.24e−06	4.56	17	18,409,265
10^{-8}	2.41e−06	8.78e−07	8.78e−07	3.83e−09	5.04	17	15,283,500
10^{-10}	2.41e−06	8.78e−07	8.78e−07	4.99e−12	5.14	17	14,648,294

Table 6
Histogram of the distribution of non-zeros per column in **cfd1**.

$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols
12	216	21	192	24	3,580	28	32	31	48
16	104	22	32	26	16	29	16	32	16
18	7352	23	16	27	58,372	30	328	33	336

Appendix D. Results for cfd2

$n = 123{,}440$ $nz = 3{,}087{,}898$ $\|A\|_\infty = 8.81$ $\|b\|_2 = 2.26$ (see Tables 8, 9 and Fig. 4).

Appendix E. Results for circuit_4

$n = 80{,}209$ $nz = 307{,}625$ $\|A\|_\infty = 5.34 \cdot 10^3$ $\|b\|_2 = 1.41$ (see Tables 10, 11 and Fig. 5).

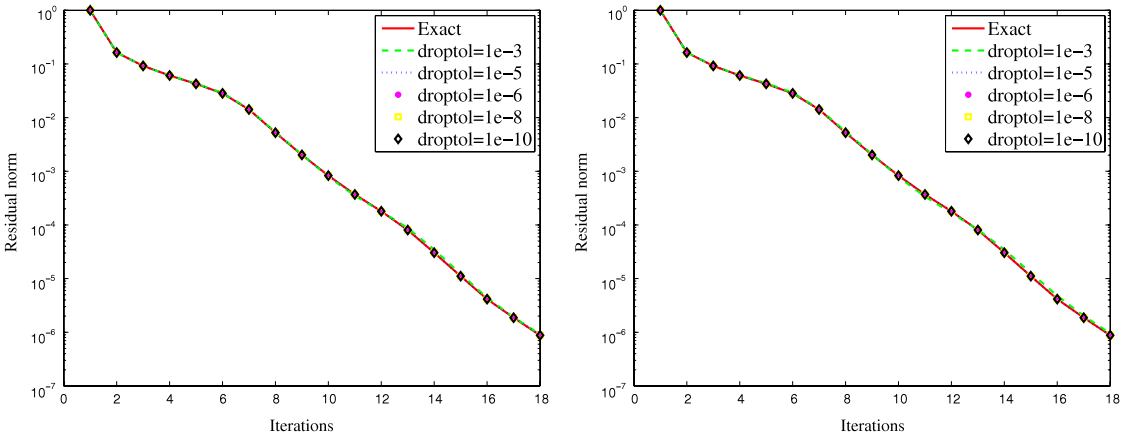

Fig. 2. Residual history of **cage14** (left: unweighted matrix–vector, right: weighted).

Table 7
Execution of **cfd1** (first block: unweighted matrix–vector, second: weighted).

droptol	$\|x - x_m\|_2/\|x\|_2$	$\|r_m\|_2/\|b\|_2$	$\|\tilde{r}_m\|_2/\|b\|_2$	δ_m	Runtime (s)	#Iter	Savings
Exact	2.26e−03	9.91e−07	9.91e−07	0	9.52	282	0
10^{-3}	2.21e−03	4.03e−05	9.97e−07	8.68e−05	10.88	398	12,513,028
10^{-5}	2.24e−03	9.99e−07	9.95e−07	1.79e−07	8.85	286	3,727,981
10^{-6}	2.24e−03	9.88e−07	9.88e−07	7.46e−09	8.88	284	3,275,940
10^{-8}	2.25e−03	9.89e−07	9.89e−07	6.55e−12	8.91	282	2,842,602
10^{-10}	2.25e−03	9.91e−07	9.91e−07	0	8.99	283	2,629,494
Exact	2.26e−03	9.91e−07	9.91e−07	0	9.51	282	0
10^{-3}	2.21e−03	4.03e−05	9.97e−07	8.68e−05	10.91	398	12,513,028
10^{-5}	2.24e−03	9.99e−07	9.95e−07	1.79e−07	8.87	286	3,727,981
10^{-6}	2.24e−03	9.88e−07	9.88e−07	7.46e−09	8.89	284	3,275,940
10^{-8}	2.25e−03	9.89e−07	9.89e−07	6.55e−12	8.92	282	2,842,602
10^{-10}	2.25e−03	9.91e−07	9.91e−07	0	9.19	283	2,629,494

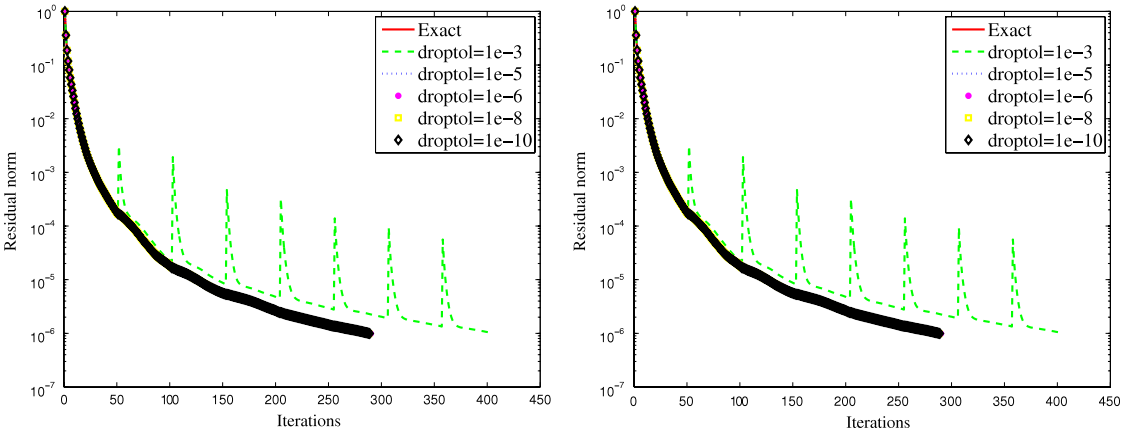

Fig. 3. Residual history of **cfd1** (left: unweighted matrix–vector, right: weighted).

Table 8
Histogram of the distribution of non-zeros per column in **cfd2**.

$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols
8	14	16	179	21	668	24	1858	27	93,198
12	1152	18	24,334	22	86	26	56	30	1,680
14	16	20	143	23	56				

Table 9
Execution of **cfd2** (first block: unweighted matrix–vector, second: weighted).

droptol	$\|x - x_m\|_2/\|x\|_2$	$\|r_m\|_2/\|b\|_2$	$\|\tilde{r}_m\|_2/\|b\|_2$	δ_m	Runtime (s)	#Iter	Savings
Exact	8.58e−03	1.00e−06	1.00e−06	0	43.96	745	0
10^{-3}	8.61e−03	3.21e−05	9.99e−07	7.26e−05	64.59	1300	61,257,142
10^{-5}	8.67e−03	1.00e−06	1.00e−06	4.09e−08	44.40	765	8,229,280
10^{-6}	8.51e−03	9.98e−07	9.98e−07	1.89e−08	43.06	749	7,106,423
10^{-8}	8.53e−03	9.99e−07	9.99e−07	1.46e−11	43.26	748	6,156,389
10^{-10}	8.54e−03	9.97e−07	9.97e−07	1.59e−15	43.13	747	5,660,679
Exact	8.58e−03	1.00e−06	1.00e−06	0	44.62	745	0
10^{-3}	8.61e−03	3.21e−05	9.99e−07	7.26e−05	65.82	1300	61,257,142
10^{-5}	8.67e−03	1.00e−06	1.00e−06	4.09e−08	44.13	765	8,229,280
10^{-6}	8.51e−03	9.98e−07	9.98e−07	1.89e−08	43.41	749	7,106,423
10^{-8}	8.53e−03	9.99e−07	9.99e−07	1.46e−11	43.32	748	6,156,389
10^{-10}	8.54e−03	9.97e−07	9.97e−07	1.59e−15	43.40	747	5,660,679

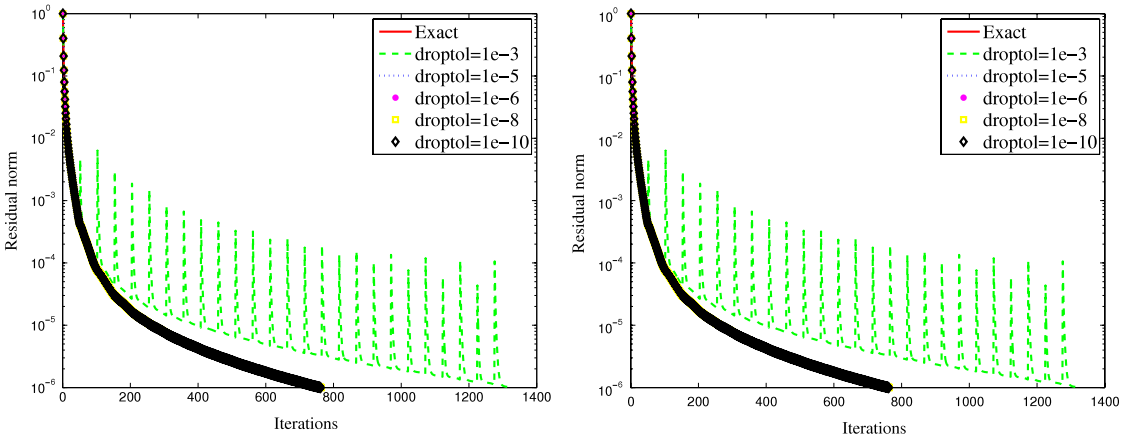

Fig. 4. Residual history of **cfd2** (left: unweighted matrix–vector, right: weighted).

Table 10
Histogram of the distribution of non-zeros per column in **circuit_4**.

$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols
1	27,922	16	33	35	5	62	10	159	44
2	111	17	31	36	1	69	1	165	1
3	23,587	18	58	40	3	70	11	169	1
4	1,381	19	13	42	3	72	1	170	1
5	8,299	20	12	43	3	84	2	286	1
6	16,510	21	20	44	2	85	1	335	1
7	661	22	2	45	1	86	2	367	1
8	238	23	165	47	1	87	2	443	1
9	247	24	21	49	2	88	1	460	1
10	259	25	1	50	1	89	2	500	1
11	234	26	11	51	2	101	1	532	1
12	45	27	11	56	1	104	1	850	1
13	2	28	1	60	129	133	1	6852	1
14	52	29	1	61	2	149	1	8900	1
15	25	33	6						

Appendix F. Results for epb3

$n = 84{,}617 \quad nz = 463{,}625 \quad \|A\|_\infty = 0.68 \quad \|b\|_2 = 6.24 \cdot 10^{-2}$ (see Tables 12, 13 and Fig. 6).

Appendix G. Results for finan512

$n = 74{,}512 \quad nz = 596{,}992 \quad \|A\|_\infty = 63.2 \quad \|b\|_2 = 9.84$ (see Tables 14, 15 and Fig. 7).

Table 11
Execution of **circuit_4** (first block: unweighted matrix–vector, second: weighted).

droptol	$\|x - x_m\|_2/\|x\|_2$	$\|r_m\|_2/\|b\|_2$	$\|\tilde{r}_m\|_2/\|b\|_2$	δ_m	Runtime (s)	#Iter	Savings
Exact	5.80e−04	9.59e−07	9.59e−07	0	1.27	60	0
10^{-3}	5.00e−04	5.16e−04	3.25e−18	7.30e−04	4.00e−02	6	481,224
10^{-5}	5.75e−04	1.28e−04	2.55e−07	1.81e−04	4.40e−02	7	561,400
10^{-6}	5.75e−04	1.68e−04	2.81e−07	2.37e−04	1.68e−01	18	1351,621
10^{-8}	5.79e−04	1.12e−06	9.92e−07	7.26e−07	6.52e−01	39	2609,778
10^{-10}	5.80e−04	9.79e−07	9.78e−07	2.07e−08	7.48e−01	41	1943,079
Exact	5.80e−04	9.59e−07	9.59e−07	0	1.26	60	0
10^{-3}	5.00e−04	5.16e−04	3.25e−18	7.30e−04	3.60e−02	6	481,224
10^{-5}	5.91e−04	1.03e−04	7.25e−07	1.46e−04	2.72e−01	24	1865,115
10^{-6}	5.79e−04	6.50e−06	9.55e−07	9.05e−06	2.52e−01	23	1726,534
10^{-8}	5.79e−04	1.03e−06	9.99e−07	3.77e−07	6.56e−01	39	2640,018
10^{-10}	5.79e−04	9.65e−07	9.65e−07	3.82e−08	7.64e−01	41	1803,025

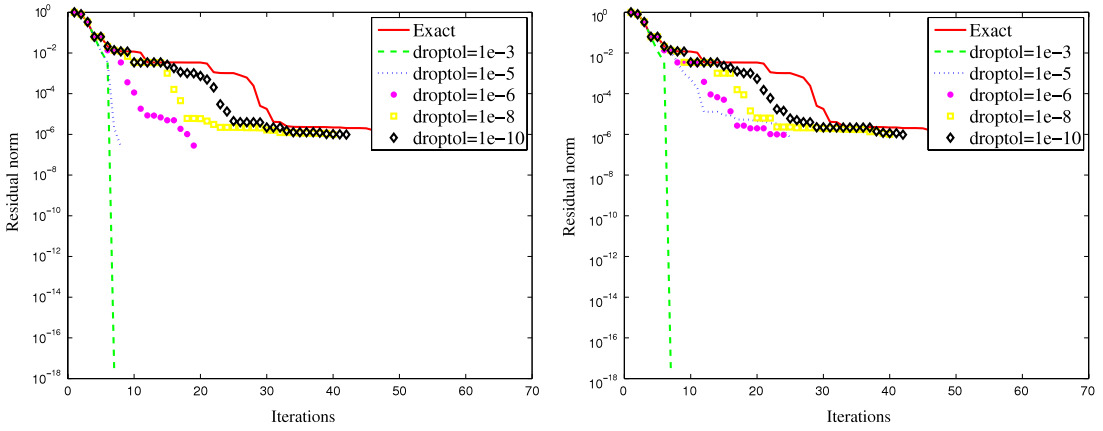

Fig. 5. Residual history of **circuit_4** (left: unweighted matrix–vector, right: weighted).

Table 12
Histogram of the distribution of non-zeros per column in **epb3**.

$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols
3	143	4	42,614	5	20	6	240	7	41,600

Table 13
Execution of **epb3** (first block: unweighted matrix–vector, second: weighted).

droptol	$\|x - x_m\|_2/\|x\|_2$	$\|r_m\|_2/\|b\|_2$	$\|\tilde{r}_m\|_2/\|b\|_2$	δ_m	Runtime (s)	#Iter	Savings
Exact	1.17e−03	1.00e−06	1.00e−06	0	62.21	2410	0
10^{-3}	2.17e−03	1.47e−05	4.07e−06	9.11e−07	56.02	2500	176,562,644
10^{-5}	1.19e−03	1.04e−06	1.04e−06	3.36e−09	60.74	2500	78,465,851
10^{-6}	1.17e−03	1.01e−06	1.01e−06	1.14e−10	61.21	2500	64,695,666
10^{-8}	1.16e−03	9.99e−07	9.99e−07	1.04e−15	60.10	2406	52,366,981
10^{-10}	1.17e−03	1.00e−06	1.00e−06	7.98e−18	58.82	2378	45,727,316
Exact	1.17e−03	1.00e−06	1.00e−06	0	61.40	2410	0
10^{-3}	3.16e+08	1.08e+09	3.19e+08	8.39e+07	53.84	2500	208,367,539
10^{-5}	1.33e−03	1.28e−06	1.27e−06	1.43e−08	59.72	2500	114,392,292
10^{-6}	1.21e−03	1.06e−06	1.06e−06	6.32e−10	61.09	2500	81,485,347
10^{-8}	1.17e−03	9.99e−07	9.99e−07	2.50e−12	54.25	2196	57,600,405
10^{-10}	1.17e−03	9.99e−07	9.99e−07	4.28e−18	61.62	2453	49,479,260

Table 14
Histogram of the distribution of non-zeros per column in **finan512**.

$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols
3	3,072	6	1,536	9	9216	13	4608	23	512
4	7,680	7	24,064	11	1536	14	512	52	512
5	13,824	8	4,608	12	1536	21	1024	55	512

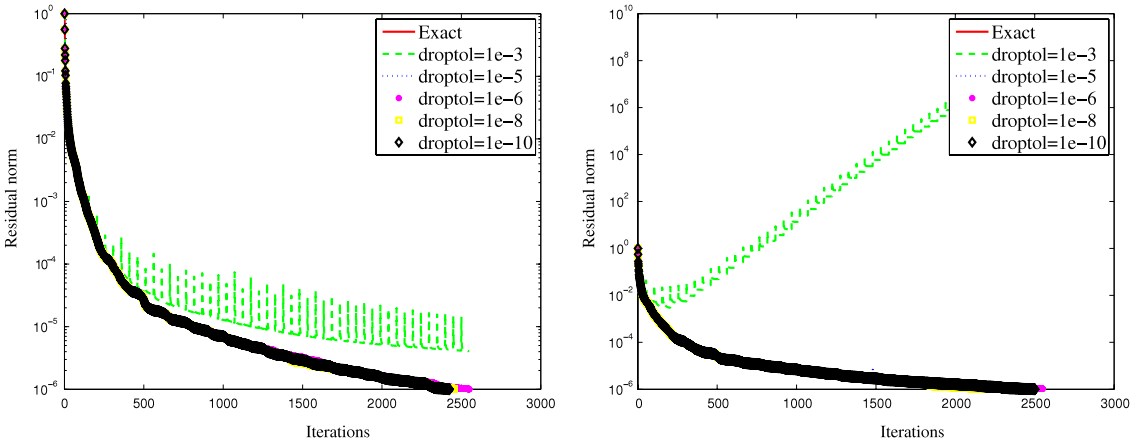

Fig. 6. Residual history of **epb3** (left: unweighted matrix–vector, right: weighted).

Table 15
Execution of **finan512** (first block: unweighted matrix–vector, second: weighted).

droptol	$\|x - x_m\|_2/\|x\|_2$	$\|r_m\|_2/\|b\|_2$	$\|\tilde{r}_m\|_2/\|b\|_2$	δ_m	Runtime (s)	#Iter	Savings
Exact	3.06e−06	8.61e−07	8.61e−07	0	3.52e−01	22	0
10^{-3}	1.10e−03	1.03e−03	8.11e−07	1.02e−02	2.12e−01	22	1626,620
10^{-5}	6.61e−06	4.97e−06	8.61e−07	4.82e−05	2.16e−01	22	1587,385
10^{-6}	3.08e−06	9.24e−07	8.61e−07	3.27e−06	2.16e−01	22	1566,795
10^{-8}	3.06e−06	8.61e−07	8.61e−07	9.44e−09	2.24e−01	22	1530,750
10^{-10}	3.06e−06	8.61e−07	8.61e−07	3.53e−11	2.24e−01	22	1500,905
Exact	3.06e−06	8.61e−07	8.61e−07	0	3.52e−01	22	0
10^{-3}	2.28e−04	1.38e−04	8.34e−07	1.36e−03	2.20e−01	22	1616,209
10^{-5}	3.25e−06	1.06e−06	8.61e−07	6.21e−06	2.20e−01	22	1575,073
10^{-6}	3.06e−06	8.62e−07	8.61e−07	3.43e−07	2.24e−01	22	1555,181
10^{-8}	3.06e−06	8.61e−07	8.61e−07	1.30e−09	2.28e−01	22	1521,005
10^{-10}	3.06e−06	8.61e−07	8.61e−07	4.73e−12	2.28e−01	22	1493,429

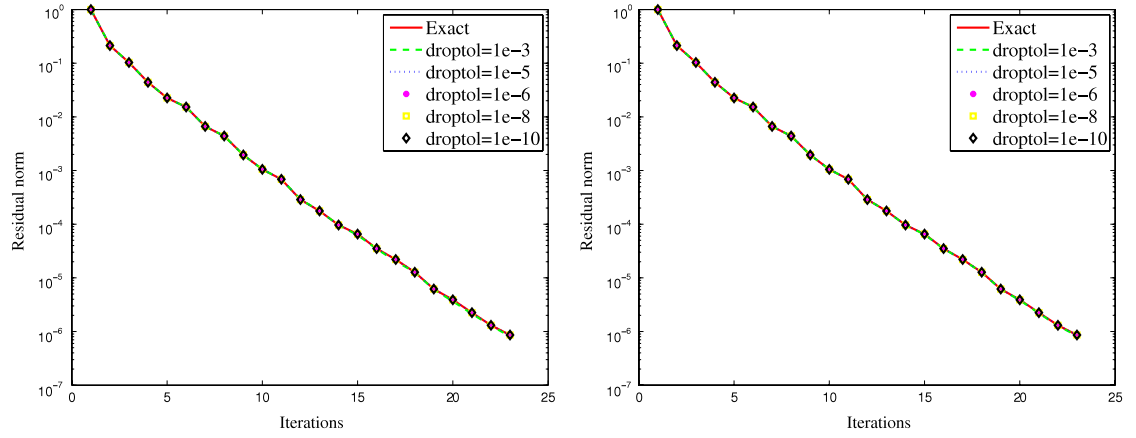

Fig. 7. Residual history of **finan512** (left: unweighted matrix–vector, right: weighted).

Appendix H. Results for hcircuit

$n = 105{,}676 \quad nz = 513{,}120 \quad \|A\|_\infty = 88.2 \quad \|b\|_2 = 1.42$ (see Tables 16, 17 and Fig. 8).

Appendix I. Results for pwtk

$n = 217{,}918 \quad nz = 11{,}634{,}424 \quad \|A\|_\infty = 1.82 \cdot 10^8 \quad \|b\|_2 = 2.27 \cdot 10^7$ (see Tables 18, 19 and Fig. 9).

Table 16
Histogram of the distribution of non-zeros per column in **hcircuit**.

$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols
2	983	14	25	32	31	62	11	146	7
3	14,612	15	533	33	1	69	1	147	5
4	52,067	16	22	34	24	74	6	148	1
5	30,275	17	402	35	2	75	6	160	1
6	425	18	12	38	4	76	6	171	2
7	199	19	170	39	5	82	6	176	6
8	530	20	42	41	3	88	12	177	6
9	2,751	21	658	44	3	92	13	182	7
10	734	23	3	50	17	93	1	266	4
11	301	26	13	51	14	122	1	723	1
12	434	27	15	56	1	134	4	1297	1
13	237	29	9	57	4	141	6	1399	1

Table 17
Execution of **hcircuit** (first block: unweighted matrix–vector, second: weighted).

droptol	$\|x - x_m\|_2/\|x\|_2$	$\|r_m\|_2/\|b\|_2$	$\|\tilde{r}_m\|_2/\|b\|_2$	δ_m	Runtime (s)	#Iter	Savings
Exact	2.64e−04	9.96e−07	9.96e−07	0	3.54	120	0
10^{-3}	7.26e−04	1.63e−04	9.86e−07	2.30e−04	9.72e−01	43	4,540,666
10^{-5}	2.68e−04	9.90e−07	9.90e−07	5.96e−09	2.26	92	9,617,470
10^{-6}	2.57e−04	9.93e−07	9.93e−07	8.59e−10	2.32	93	9,581,063
10^{-8}	2.57e−04	9.93e−07	9.93e−07	1.29e−11	2.22	90	8,936,471
10^{-10}	2.57e−04	9.89e−07	9.89e−07	1.01e−13	2.33	92	8,628,089
Exact	2.64e−04	9.96e−07	9.96e−07	0	3.52	120	0
10^{-3}	3.14e−03	9.69e−04	6.81e−07	1.37e−03	3.76e−01	25	2,641,128
10^{-5}	2.97e−04	9.90e−07	9.90e−07	1.05e−08	3.11	128	13,479,729
10^{-6}	2.61e−04	9.98e−07	9.98e−07	4.20e−09	2.35	94	9,854,798
10^{-8}	2.52e−04	9.71e−07	9.71e−07	6.88e−11	2.28	92	9,412,928
10^{-10}	2.56e−04	9.90e−07	9.90e−07	1.26e−12	2.31	92	9,031,938

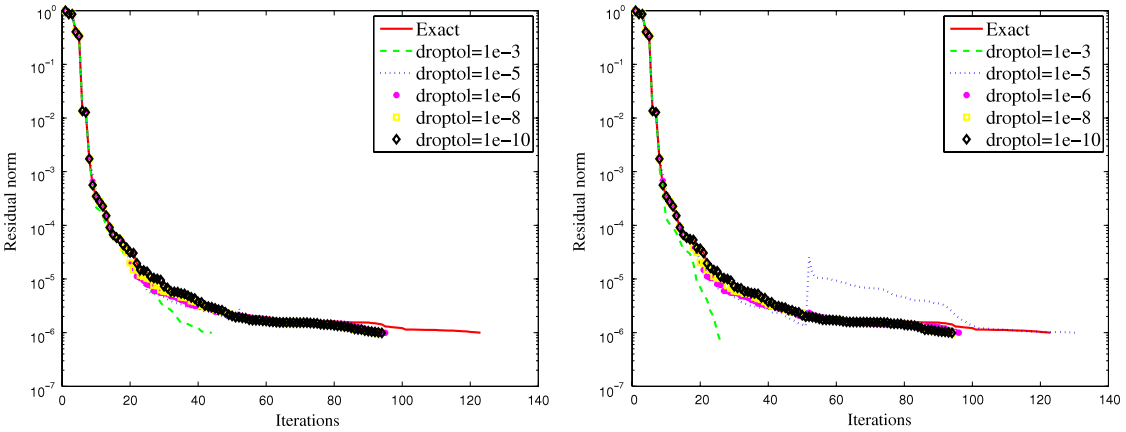

Fig. 8. Residual history of **hcircuit** (left: unweighted matrix–vector, right: weighted).

Table 18
Histogram of the distribution of non-zeros per column in **pwtk**.

$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols
3	64	12	17,862	19	31,056	26	7444	33	882
5	2,312	13	28,844	20	25,348	27	5588	34	604
6	368	14	37,962	21	19,914	28	3070	35	604
7	5,604	15	30,248	22	19,032	29	2576	36	212
8	12,212	16	36,326	23	13,523	30	1510	37	196
9	4,652	17	35,556	24	13,136	31	845	38	32
10	16,548	18	30,752	25	8,507	32	914	39	24
11	30,988								

Table 19
Execution of **pwtk** (first block: unweighted matrix–vector, second: weighted).

droptol	$\|x - x_m\|_2/\|x\|_2$	$\|r_m\|_2/\|b\|_2$	$\|\tilde{r}_m\|_2/\|b\|_2$	δ_m	Runtime (s)	#Iter	Savings
Exact	4.51e−02	1.42e−06	1.42e−06	0	4.04e+02	2500	0
10^{-3}	9.44e−02	3.38e−05	4.65e−06	7.92e+02	1.84e+02	2500	463,992,469
10^{-5}	4.21e−02	1.29e−06	1.29e−06	1.39	3.07e+02	2500	205,041,066
10^{-6}	4.23e−02	1.30e−06	1.30e−06	4.02e−02	3.16e+02	2500	183,990,104
10^{-8}	4.28e−02	1.32e−06	1.32e−06	9.31e−05	3.48e+02	2500	112,674,877
10^{-10}	4.42e−02	1.38e−06	1.38e−06	6.16e−08	3.68e+02	2500	67,381,045
Exact	4.51e−02	1.42e−06	1.42e−06	0	4.04e+02	2500	0
10^{-3}	4.42e−02	1.38e−06	1.38e−06	1.60e−04	3.28e+02	2500	166,083,405
10^{-5}	4.41e−02	1.37e−06	1.37e−06	1.40e−06	3.57e+02	2500	100,875,373
10^{-6}	4.42e−02	1.38e−06	1.38e−06	9.73e−08	3.71e+02	2500	76,454,387
10^{-8}	4.45e−02	1.39e−06	1.39e−06	3.11e−10	3.80e+02	2500	54,245,835
10^{-10}	4.41e−02	1.38e−06	1.38e−06	1.88e−13	3.81e+02	2500	46,676,933

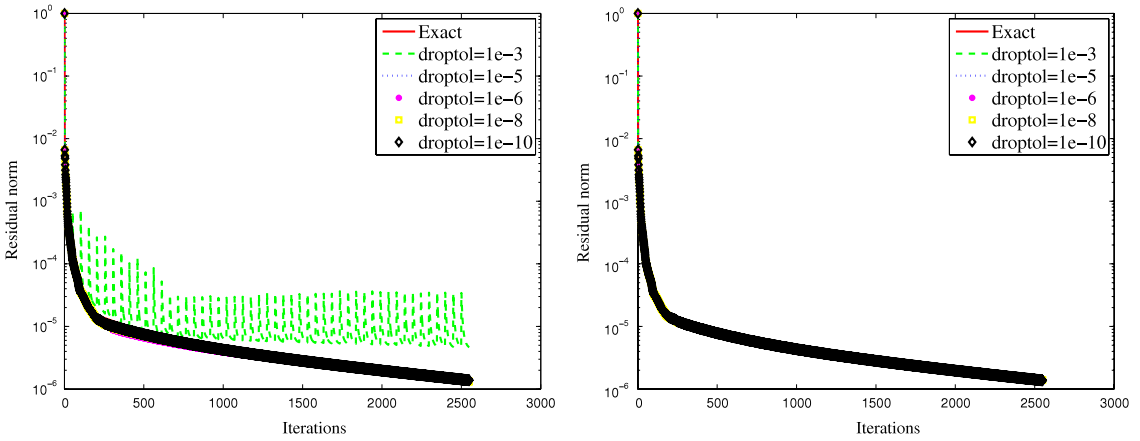

Fig. 9. Residual history of **pwtk** (left: unweighted matrix–vector, right: weighted).

Table 20
Histogram of the distribution of non-zeros per column in **stomach**.

$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols
6	448	10	84,000	12	672	16	448	21	42,000
9	896	11	448	15	84,000	17	224	22	224

Table 21
Execution of **stomach** (first block: unweighted matrix–vector, second: weighted).

droptol	$\|x - x_m\|_2/\|x\|_2$	$\|r_m\|_2/\|b\|_2$	$\|\tilde{r}_m\|_2/\|b\|_2$	δ_m	Runtime (s)	#Iter	Savings
Exact	1.17e−06	7.85e−07	7.85e−07	0	9.20e−01	17	0
10^{-3}	2.26e−04	6.01e−04	5.69e−07	3.91e−04	4.04e−01	17	3625,372
10^{-5}	1.30e−06	1.66e−06	7.85e−07	9.37e−07	4.08e−01	17	3622,495
10^{-6}	1.17e−06	8.21e−07	7.85e−07	1.32e−07	4.08e−01	17	3621,217
10^{-8}	1.17e−06	7.85e−07	7.85e−07	6.16e−10	4.04e−01	17	3618,647
10^{-10}	1.17e−06	7.85e−07	7.85e−07	2.07e−12	4.04e−01	17	3616,129
Exact	1.17e−06	7.85e−07	7.85e−07	0	9.00e−01	17	0
10^{-3}	2.49e−04	6.06e−04	5.23e−07	3.94e−04	4.04e−01	17	3625,699
10^{-5}	1.34e−06	1.69e−06	7.85e−07	9.53e−07	4.08e−01	17	3622,758
10^{-6}	1.17e−06	8.22e−07	7.85e−07	1.34e−07	4.04e−01	17	3621,476
10^{-8}	1.17e−06	7.85e−07	7.85e−07	6.21e−10	4.04e−01	17	3618,921
10^{-10}	1.17e−06	7.85e−07	7.85e−07	2.25e−12	4.04e−01	17	3616,387

Table 22
Histogram of the distribution of non-zeros per column in **torso3**.

$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols	$nz(a_j)$	#cols
6	1028	10	35,880	13	16	16	2300	19	141,166
7	4	11	2,056	14	8	17	8	20	36,104
9	2056	12	1,268	15	35,864	18	366	21	1,032

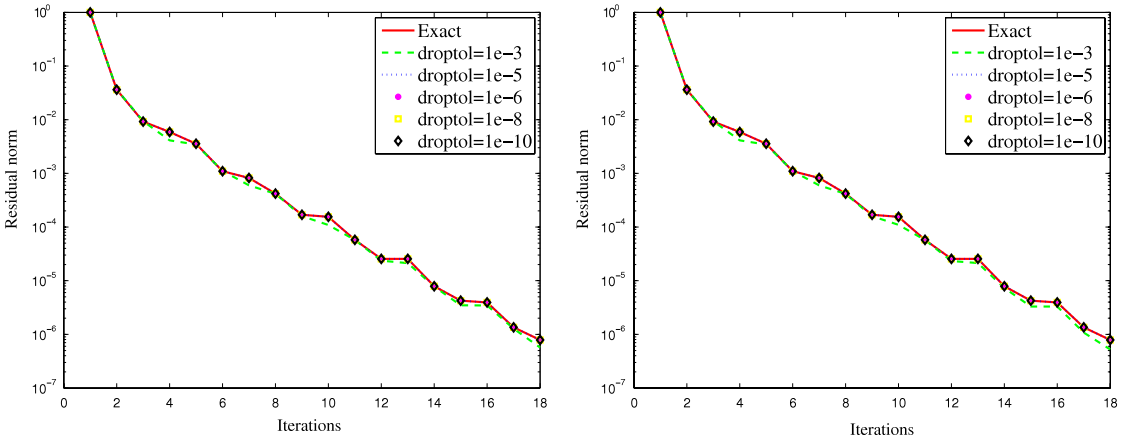

Fig. 10. Residual history of **stomach** (left: unweighted matrix–vector, right: weighted).

Table 23
Execution of **torso3** (first block: unweighted matrix–vector, second: weighted).

droptol	$\|x - x_m\|_2/\|x\|_2$	$\|r_m\|_2/\|b\|_2$	$\|\tilde{r}_m\|_2/\|b\|_2$	δ_m	Runtime (s)	#Iter	Savings
Exact	1.55e−06	9.54e−07	9.54e−07	0	2.69	31	0
10^{-3}	1.51e−03	6.61e−03	9.46e−07	2.20e−03	1.35	31	8020,103
10^{-5}	8.79e−06	3.79e−05	9.54e−07	1.26e−05	1.36	31	7988,988
10^{-6}	1.71e−06	3.35e−06	9.54e−07	1.07e−06	1.36	31	7971,895
10^{-8}	1.55e−06	9.55e−07	9.54e−07	1.13e−08	1.37	31	7933,562
10^{-10}	1.55e−06	9.54e−07	9.54e−07	8.10e−11	1.38	31	7885,684
Exact	1.55e−06	9.54e−07	9.54e−07	0	2.64	31	0
10^{-3}	3.69e−03	5.60e−03	9.50e−07	1.87e−03	1.35	31	8021,900
10^{-5}	1.85e−05	3.88e−05	9.55e−07	1.29e−05	1.36	31	7990,262
10^{-6}	2.27e−06	3.58e−06	9.54e−07	1.14e−06	1.36	31	7973,179
10^{-8}	1.55e−06	9.55e−07	9.54e−07	1.13e−08	1.36	31	7934,593
10^{-10}	1.55e−06	9.54e−07	9.54e−07	7.92e−11	1.37	31	7887,047

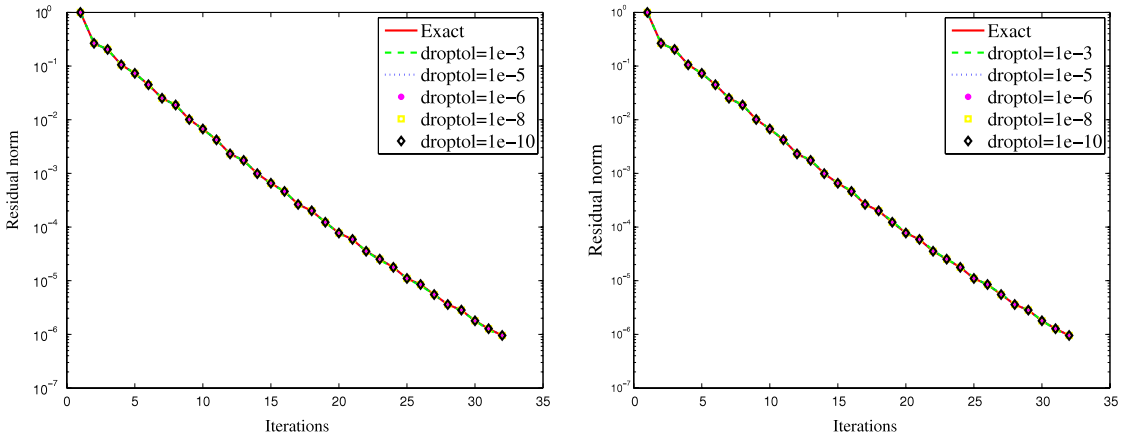

Fig. 11. Residual history of **torso3** (left: unweighted matrix–vector, right: weighted).

Appendix J. Results for stomach

$n = 213{,}360 \quad nz = 3{,}021{,}648 \quad \|A\|_\infty = 4.09 \quad \|b\|_2 = 6.51 \cdot 10^1$ (see Tables 20, 21 and Fig. 10).

Appendix K. Results for torso3

$n = 259{,}156 \quad nz = 4{,}429{,}042 \quad \|A\|_\infty = 10.9 \quad \|b\|_2 = 3.33 \cdot 10^1$ (see Tables 22, 23 and Fig. 11).

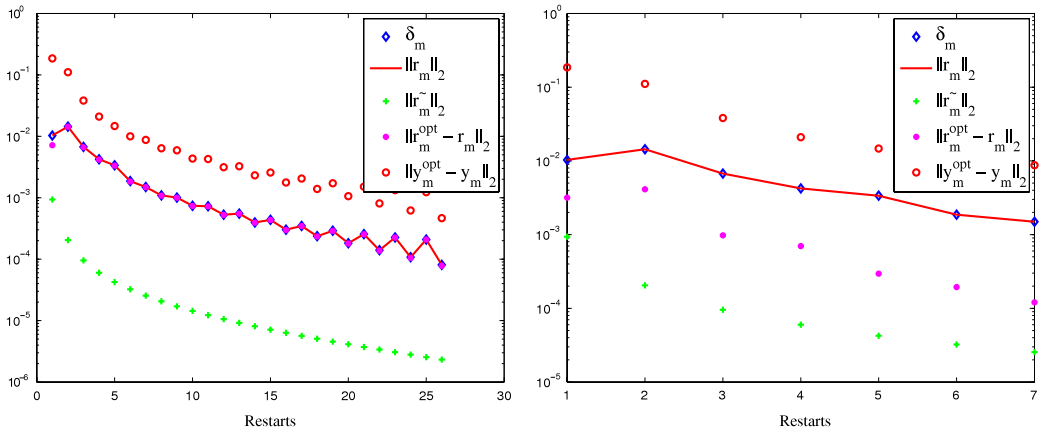

Fig. 12. History of attributes of **cfd2** (left: unweighted matrix–vector, right: weighted).

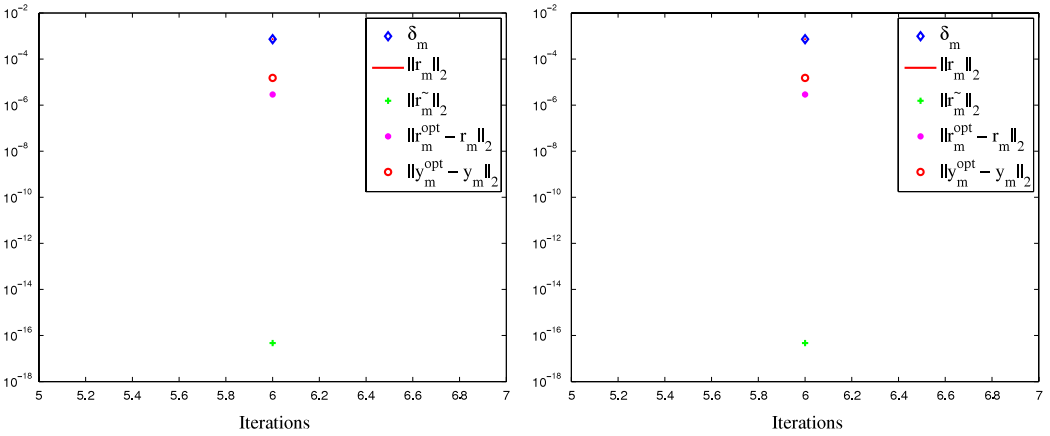

Fig. 13. History of attributes of **circuit_4** (left: unweighted matrix–vector, right: weighted).

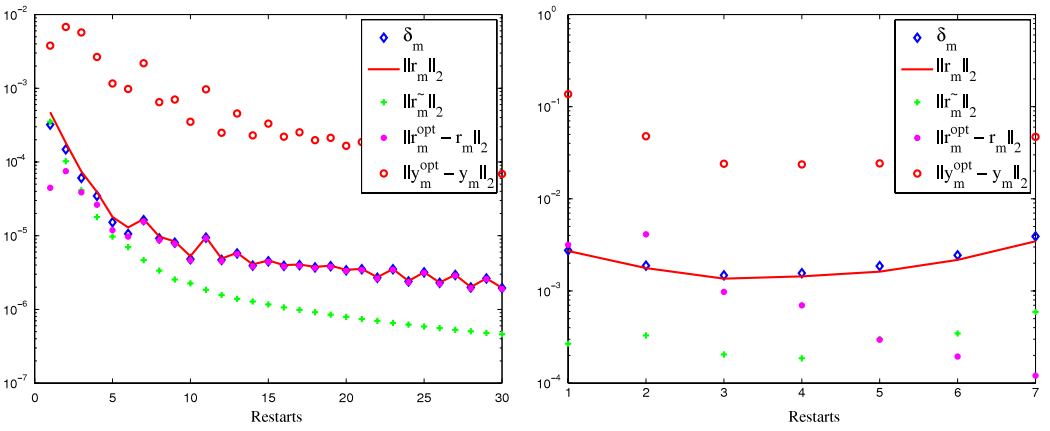

Fig. 14. History of attributes of **epb3** (left: unweighted matrix–vector, right: weighted).

References

[1] Y. Saad, M.H. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 3 (7) (1986) 856–869.
[2] A. Bouras, V. Fraysse, Inexact matrix–vector products in Krylov methods for solving linear systems: a relaxation strategy, SIAM J. Matrix Anal. Appl. 26 (3) (2005) 660–678.
[3] J.V.D. Eshof, G.L.G. Sleijpen, Inexact Krylov subspace methods for linear systems, SIAM J. Matrix Anal. Appl. 26 (1) (2004) 125–153.
[4] X. Du, D.B. Szyld, Inexact GMRES for singular linear systems, BIT 48 (3) (2008) 511–531.

[5] M.K. Seager, A. Greenbaum, SLAP: Sparse Linear Algebra Package, version 2, 1989, FOTRAN90 compatibility due to John Burkardt: http://people.sc.fsu.edu/~burkardt/f_src/dlap/dlap.html.
[6] Y. Saad, Iterative Methods for Sparse Linear Systems, PWS Publishing Company, 1996.
[7] V. Simoncini, D.B. Szyld, Theory of inexact Krylov subspace methods and applications to scientific computing, SIAM J. Sci. Comput. 25 (2) (2003) 454–477.
[8] L. Giraud, S. Gratton, J. Langou, Convergence in backward error of relaxed GMRES, SIAM J. Sci. Comput. 29 (2007) 710–728.
[9] C. Rémi, E. Jocelyne, Newton–GMRES algorithm applied to compressible flows, Internat. J. Numer. Methods Fluids 23 (2) (1996) 177–190.
[10] S. MacNamara, K. Burrage, R.B. Sidje, Inexact uniformization method for computing transient distributions of Markov chains, SIAM J. Sci. Comput. 29 (2007) 2562–2580.
[11] T.A. Davis, Y.F. Hu, The University of Florida sparse matrix collection, ACM TOMS, 2010 (submitted for publication). http://www.cise.ufl.edu/research/sparse/matrices.