DETECTION AND CORRECTION OF SILENT ERRORS IN THE CONJUGATE GRADIENT ALGORITHM

GÉRARD MEURANT*

This article is dedicated to Claude Brezinski on the occasion of his 80th birthday

Abstract. We propose a new way to detect and correct silent errors in the conjugate gradient algorithm. The detection criterion is simple, cheap to implement and can be used at each iteration. This simplifies the correction process. Numerical experiments show that the new criterion is robust and reliable.

Key words. Conjugate gradient algorithm, silent error, detection and correction.

AMS subject classifications. 65F10,65F30,65F50

1. Introduction. Today's supercomputers have more and more processing elements. Therefore, it is more likely than before that there can be some hardware failures during the execution of a user's parallel code. Some failures may crash the job but there are also what are called *soft faults* or *silent errors*.

A silent error may cause an application floating point value to be changed to a different floating point value that is valid, but incorrect. This may have a strong and negative effect on the output of the code. The main problem is that these errors may emit no indication that something has gone wrong. The faults can be persistent or transient. Major challenges with silent errors are their detection and correction. These problems were considered and discussed in [14, 3, 2, 24, 5, 11, 22, 20, 6, 7, 8, 9, 10, 12, 15, 19, 1] (ordered by date).

The most basic technique for fault detection is to do redundant computations. These very general techniques are called *double modular redundancy* (DMR) and *triple modular redundancy* (TMR). However, doing redundant computations is very costly and may be impracticable.

A more interesting class of methods is named algorithm-based fault tolerance (ABFT). It exploits the knowledge we may have about the numerical method in use. Early in the 1980s the paper [14] described an algorithm capable of detecting and correcting a single silent error in a matrix-matrix multiplication using row and column check-sums. The authors considered dense matrices. Check-sum techniques for sparse matrix-vector product and for sparse triangular solves were considered in [24] and applied to the conjugate gradient (CG) method for solving symmetric positive definite linear systems $Ax = b$.

Other techniques for detecting silent errors in CG (see Algorithm 1) were

- monitoring the residual norms, from one iteration to the next or in the average, checking if the current residual norm is larger than the mean of the last m residual norms,
- check of local orthogonality. For instance, check of $(p_{k-1}, Ap_k)/[\|p_{k-1}\| \, \|Ap_k\|]$ which must be zero in exact arithmetic; see [23],
- comparison of the true and computed residual vectors.

Monitoring the residual norms may not be easy when there are large oscillations of those norms and checking local orthogonality properties is not that obvious either.

*(gerard.meurant@gmail.com) started in Saint Aubin sur mer, September 2021, version July 27, 2022

Moreover, checking $(p_{k-1}, Ap_k)/[\|p_{k-1}\| \, \|Ap_k\|]$ may work but it needs the computation of three additional dot products (because the normalization is necessary) which are not useful for the algorithm independently of the error detection.

A criterion based on the gap between the computed and true residual norms was proposed in [1] but the norm of the matrix A is needed. It was complemented by another criterion using an approximation of the largest eigenvalue of A.

Many papers proposed to use checkpointing to be able to correct the detected errors. For instance, verification can be done every d iterations and checkpointing every $c \times d$ iterations.

Self-stabilizing corrections after error detection in the CG method were investigated in [22]. But the correction amounts to modifying the algorithm and this may lead to a delayed convergence.

In this paper we are interested in the detection and correction of silent errors in CG for solving linear systems $Ax = b$. In Section 2 we examine the effect of silent errors on CG convergence. We introduce a new way of detecting silent errors in CG in Section 3. We use a relation that must be satisfied by the CG coefficients and we discuss its validity in finite precision arithmetic. In Section 4 we show how to correct the errors and resume the algorithm. Checking if an error has occurred need some additional computations. We show in Section 5 how these extra operations can be used to introduce some parallelism in the algorithm.

2. The effect of silent errors on CG convergence. The well-known CG algorithm is described as Algorithm 1. In this work we simulate silent errors by bit flips in the output of some CG steps. Since we use IEEE double precision floating point arithmetic, the mantissa (or significand) has 52 bits that we number from 1 to 52, 1 being the rightmost bit, the exponent has 11 bits that is, bits 53 to 63 and the sign bit is bit 64.

A bit flip is changing the value of one of the bit b_i, $i = 1, \ldots, 64$. If $b_i = 0$ the new value is 1 and if $b_i = 1$, the new value is 0. When we modify the output of a scalar like, for instance, $r_k^T r_k$ we flip just one given bit of the output. When we modify the output of a vector operation, we flip one bit in only one component of the vector. This component is chosen randomly.

Algorithm 1 Conjugate Gradient

input A, b, x_0
$r_0 = b - Ax_0$
$p_0 = r_0$
for $k = 1, \ldots$ until convergence **do**
  $\alpha_{k-1} = \frac{r_{k-1}^T r_{k-1}}{p_{k-1}^T A p_{k-1}}$
  $x_k = x_{k-1} + \alpha_{k-1} p_{k-1}$
  $r_k = r_{k-1} - \alpha_{k-1} A p_{k-1}$
  $\beta_k = \frac{r_k^T r_k}{r_{k-1}^T r_{k-1}}$
  $p_k = r_k + \beta_k p_{k-1}$
end for

Intuitively we may think that flipping low order bits has less importance than, say, flipping a bit in the exponent field. It is also likely that flipping a bit when CG has almost converged is less dramatic than an early bit flip. Actually, these were the

conclusions of the statistical study done in [1]. Let us check this when solving a linear system with the matrix *bcsstk01*. This matrix is of order 48 with a condition number of 8.8234 10^5. However, almost 220 CG iterations are needed to reach the maximum attainable accuracy for the true residual norm $\|b - Ax_k\|$. We use a zero initial vector an a random right-hand side with components in $[0, 1)$.

Figure 2.1 shows the relative true residual norms when flipping bits 7, 35 or 61 in the output of $(r_k, r_k) = r_k^T r_k$ at iteration 50. We can see that flipping the low order bit 7 does not do too much harm to CG convergence. Flipping bit 35 delays convergence. It takes almost 500 iterations to reach the maximum attainable accuracy. CG does not converge any longer when flipping bit 61 which is in the exponent field.

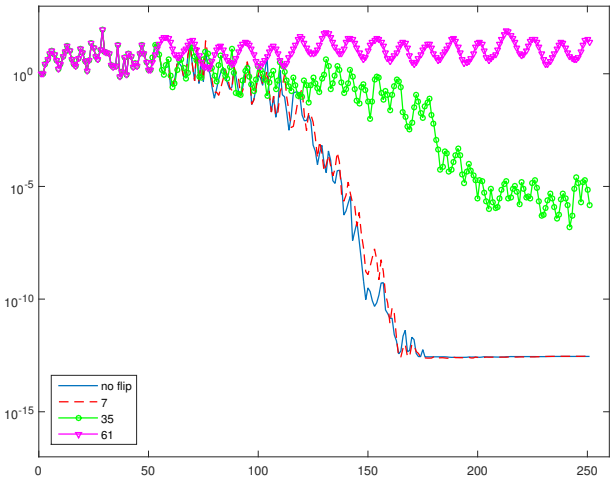

FIG. 2.1. *bcsstk01, relative true residual norms when flipping bits 7, 35 or 61 in* (r_k, r_k) *at iteration 50*

Figure 2.2 shows the residual norms when flipping the same bits later at iteration 110. Flipping bit 35 delays convergence and CG residual norms stall when flipping bit 61.

Flipping bit 35 later at iteration 140 does not prevent CG convergence but there is still no convergence when flipping bit 61 as one can see in Figure 2.3.

For Figure 2.4 we successively flip bits 10, 20, 30, 40, 50 and 60 at iteration 110. Flipping bits 10 and 20 once does not change CG convergence. Perturbation of bits larger than or equal to 30 delays or completely spoils convergence. For bits 30, 40 and 50 there is finally convergence but with a larger and larger delay. With the flip of bit 60 there is no convergence in 1000 iterations.

Figure 2.5 shows the residual norms when we flip bit 60 in different CG steps at iteration 110. For vectors we flip the bit in only one component chosen randomly. In this computation without preconditioning the vector labelled z is equal to the vector r, but it was not the same component that was modified and it explains why the results are not the same. We observe that in most cases there is no convergence. In fact, when flipping bit 60 in the output of the matrix-vector product, there is not even convergence in 1000 iterations. When flipping bits which are not in the exponent field, the algorithm may converge, but sometimes with a large number of iterations.

The results above confirm what was already observed in [1] where an interesting statistical study of bit flips in the multiplication Ap_{k-1} was done. Since silent errors

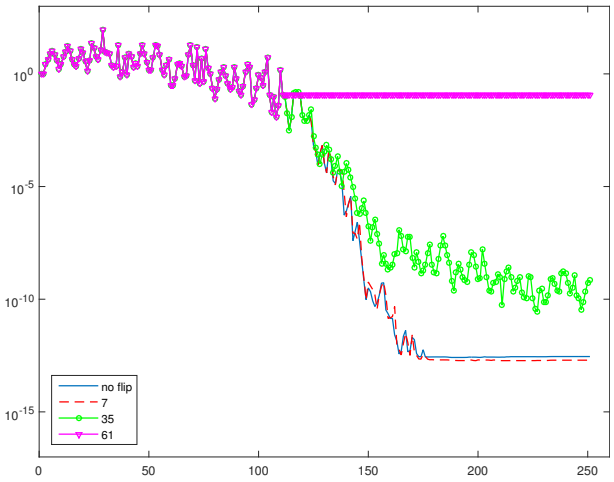

FIG. 2.2. *bcsstk01, relative true residual norms when flipping bits 7, 35 or 61 in* (r_k, r_k) *at iteration 110*

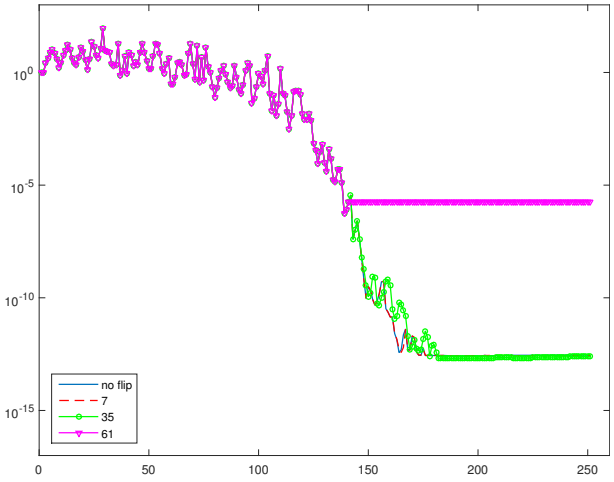

FIG. 2.3. *bcsstk01, relative true residual norms when flipping bits 7, 35 or 61 in* (r_k, r_k) *at iteration 140*

can have very bad consequences for CG convergence, particularly when the leftmost bits are concerned, it is important to be able to detect these errors. We explain how this can be done in the next section.

3. Detection of silent errors in CG. Let us consider a formula that was derived in [21] and used in [16] in 1987 for the sake of introducing more parallelism in CG with a predictor-corrector technique. This formula was reconsidered and its usefulness discussed more recently in [4]. In CG the residual vectors satisfy the relation

$$r_k - r_{k-1} = -\alpha_{k-1} A p_{k-1}.$$

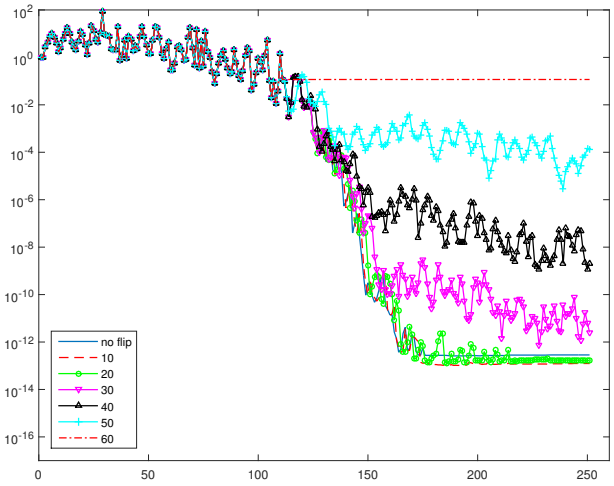

FIG. 2.4. *bcsstk01, relative true residual norms when flipping bits 10 to 60 in* (r_k, r_k) *at iteration 110*

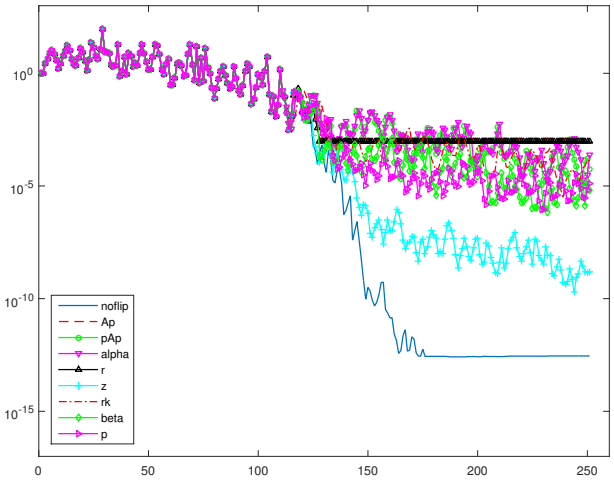

FIG. 2.5. *bcsstk01, relative true residual norms when flipping bit 60 in different CG steps at iteration 110*

Computing the norm squared of the left-hand side and using the local orthogonality of the residual vectors $(r_k, r_{k-1}) = 0$, we obtain

$$\alpha_{k-1}^2 (Ap_{k-1}, Ap_{k-1}) = (r_{k-1}, r_{k-1}) + (r_k, r_k) = \|r_{k-1}\|^2 + \|r_k\|^2, \tag{3.1}$$

which leads to the following relation between the CG coefficients α_{k-1} and β_k,

$$\alpha_{k-1}^2 \frac{(Ap_{k-1}, Ap_{k-1})}{(r_{k-1}, r_{k-1})} = 1 + \beta_k.$$

We claim that if there are one or several silent errors in iteration k, relation (3.1) won't be satisfied. The only CG step that cannot be checked by this relation is the computation of x_k because a fault in this step does not have any influence on the

next steps of the algorithm. Faults in x_k can be tested by computing the norm of the true residual $b - Ax_k$ at the end of the computation. However, in that case, the only choice we have is to run the whole algorithm again.

Let us first see how relation (3.1) is satisfied when there is no fault. We define

$$d_k = \frac{|\alpha_{k-1}\|Ap_{k-1}\| - (\|r_{k-1}\|^2 + \|r_k\|^2)^{\frac{1}{2}}|}{(\|r_{k-1}\|^2 + \|r_k\|^2)^{\frac{1}{2}}},$$

that is, the absolute value of the difference of the square roots of the left-hand side and the right-hand side of (3.1) divided by the square root of the right-hand side. We use this to avoid too small values in the denominator. In Figure 3.1 we plot d_k for the matrix *bcsstk01* when there are no silent errors. It is increasing with the iteration number but stabilized at most at the level of 10^{-13}.

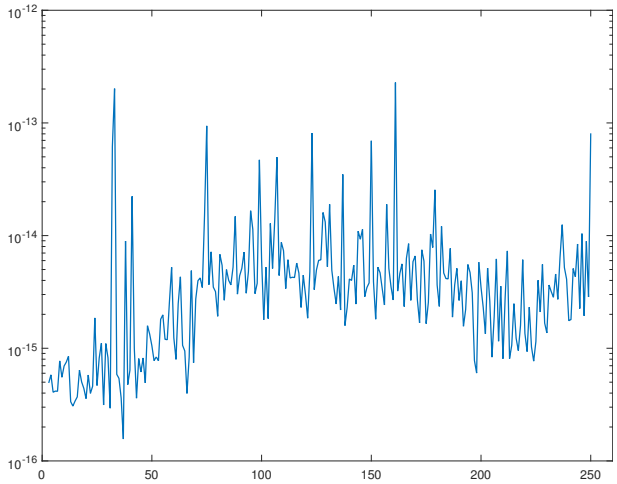

FIG. 3.1. *bcsstk01, relative difference d_k when there is no error*

This is not the machine precision level. Hence, we may ask: How is relation (3.1) satisfied in finite precision arithmetic and what happens to d_k in that case?

The purpose of the following rounding error analysis is to understand why this relation is not satisfied up to machine precision. For the analysis we use the standard model for the IEEE floating point arithmetic; see, for instance, [13] and [17]. We note that a rounding error analysis of a formula mathematically equivalent to (3.1) was done in [4]. From [17], the recurrence relations for the residual and the descent direction become

$$\begin{aligned} r_k &= r_{k-1} - \alpha_{k-1}Ap_{k-1} + \delta^r_{k-1}, \\ p_k &= r_k + \beta_k p_{k-1} + \delta^p_{k-1}, \end{aligned} \tag{3.2}$$

where now r_k and p_k are the computed vectors. Of course, in finite precision arithmetic, the coefficients α_{k-1} and β_k are not computed exactly, but the corresponding rounding errors can be incorporated in the perturbation terms δ^r_{k-1} and δ^p_{k-1}. Then, we can assume that the coefficients are computed exactly (see [17]) and the error terms can be bounded as

$$\|\delta^r_{k-1}\| \leq u\|r_{k-1}\| \left[1 + \bar{\kappa}(A)\frac{\|r_{k-1}\|}{\|p_{k-1}\|}\{m + 2 + (m+n)\bar{\kappa}(A) + n + 1\}\right] + O(u^2),$$

$$\|\delta^p_{k-1}\| \leq u\|r_k\| \left[1 + (3+2n)\frac{\|p_{k-1}\|}{\|r_{k-1}\|^2}\|r_k\|\right] + O(u^2),$$

with $\bar{\kappa}(A) = \| \, |A| \, \|/\lambda_1$, λ_1 being the smallest eigenvalue of A, n is the order of A, m is the maximum number of nonzero entries in rows of A and u is the roundoff unit. These inequalities can be written as

$$\begin{aligned}\|\delta^r_{k-1}\| &\leq u\|r_{k-1}\| \, (1 + C^r_{k-1}) + O(u^2),\\ \|\delta^p_{k-1}\| &\leq u\|r_k\| \, (1 + C^p_{k-1}) + O(u^2).\end{aligned}$$

The positive coefficients C^r_{k-1} and C^p_{k-1} are bounded because the ratios $\|r_{k-1}\|/\|p_{k-1}\|$, $\|p_{k-1}\|/\|r_{k-1}\|$ and $\|r_k\|/\|r_{k-1}\|$ are bounded.

For relation (3.1), without assuming exact local orthogonality and taking into account the rounding errors, we have the following result.

THEOREM 3.1.
In finite precision arithmetic it holds

$$\begin{aligned}\alpha^2_{k-1}\|Ap_{k-1}\|^2 = \|r_{k-1}\|^2 + \|r_k\|^2 + \|\delta^r_{k-1}\|^2\\ + 2\,[(r_{k-1} - r_k, \delta^r_{k-1}) - (r_{k-1}, r_k)],\end{aligned}$$

Proof. This relation is obtained by using (3.2) □

Then, we have

$$\alpha_{k-1}\|Ap_{k-1}\| \leq (\|r_{k-1}\|^2 + \|r_k\|^2)^{\frac{1}{2}} + (\big| \, \|\delta^r_{k-1}\|^2 + 2[(r_{k-1} - r_k, \delta^r_{k-1}) - (r_{k-1}, r_k)] \, \big|)^{\frac{1}{2}}.$$

We have to consider the terms in the second part of the right-hand side divided by $(\|r_{k-1}\|^2 + \|r_k\|^2)^{\frac{1}{2}}$. Let us see what can be the largest term in the right-hand side. The first term within the parentheses is positive and can be bounded as

$$\begin{aligned}\frac{\|\delta^r_{k-1}\|^2}{\|r_{k-1}\|^2 + \|r_k\|^2} &\leq u^2 \frac{\|r_k\|^2}{\|r_{k-1}\|^2 + \|r_k\|^2}(1 + C^p_{k-1})^2 + O(u^4),\\ &\leq u^2 \frac{1}{\frac{\|r_{k-1}\|^2}{\|r_k\|^2} + 1}(1 + C^p_{k-1})^2 + O(u^4).\end{aligned}$$

When CG converges fast the denominator in the second line is large and, in any case, the right-hand side is small. For the next term, we have

$$\begin{aligned}\frac{2\,|(r_{k-1} - r_k, \delta^r_{k-1})\,|}{\|r_{k-1}\|^2 + \|r_k\|^2} &\leq \frac{2\,(\|r_{k-1}\| + \|r_k\|\,)\|\delta^r_{k-1}\|}{\|r_{k-1}\|^2 + \|r_k\|^2},\\ &= \frac{2\,\|\delta^r_{k-1}\|}{\|r_{k-1}\| + \frac{\|r_k\|^2}{\|r_{k-1}\|}} + \frac{2\,\|\delta^r_{k-1}\|}{\frac{\|r_{k-1}\|^2}{\|r_k\|} + \|r_k\|}.\end{aligned} \tag{3.3}$$

To simplify the notation, we can write $2\,\|\delta^r_{k-1}\| \leq uC\|r_{k-1}\|$ where C is bounded. The first term of the right-hand side of (3.3) is bounded by

$$\frac{u\,C\|r_{k-1}\|}{\|r_{k-1}\| + \frac{\|r_k\|^2}{\|r_{k-1}\|}} = \frac{u\,C}{1 + \frac{\|r_k\|^2}{\|r_{k-1}\|^2}}.$$

When CG converges fast, this term is close to $u\,C$. The second term is bounded by

$$\frac{u\,C\|r_{k-1}\|}{\frac{\|r_{k-1}\|^2}{\|r_k\|}+\|r_k\|} = \frac{u\,C}{\frac{\|r_{k-1}\|}{\|r_k\|}+\frac{\|r_k\|}{\|r_{k-1}\|}}.$$

If one term of the denominator is large, the other is small and vice-versa. Hence, this term of the left-hand side of (3.3) is proportional to u. Note that this is a very loose bound since, when CG converges, the vector $r_{k-1}-r_k$ becomes small, meaning that the term $(r_{k-1}-r_k;\delta^r_{k-1})$ is also small.

The term that can be a problem for us is

$$\frac{2|(r_{k-1},r_k)|}{\|r_{k-1}\|^2+\|r_k\|^2},$$

because, even though $|(r_{k-1},r_k)|$ is small (as it can be seen in the Appendix), the denominator $\|r_{k-1}\|^2+\|r_k\|^2$ can be small too. However, if we want to use (r_{k-1},r_k) in our detection criterion, an additional dot product has to be computed. This is too expensive since we already have to compute (Ap_{k-1},Ap_{k-1}). Therefore, we will use the relative difference without the term proportional to (r_{k-1},r_k) and we will have to deal with the fact that relation (3.1) is not satisfied up to machine precision.

The reader may wonder why we use the relative difference and not the absolute one. The main reason is that the absolute difference can sometimes be large and more problem dependent than the relative difference.

Let us now consider what happens to the relative difference d_k when there are bit flips during the computation. We first flip bits 7, 35 or 61 of the result of the computation of (r_k,r_k) at iteration 110 for *bcsstk01* as we did above. From Figure 3.2 we clearly see that when flipping bits 35 or 61, something bad happened at iteration 110. Flipping bit 7 is almost innocuous since the relative difference is almost the same as what we had in Figure 3.1. The largest value of d_k is obtained when we flip bit 61 which is in the exponent field.

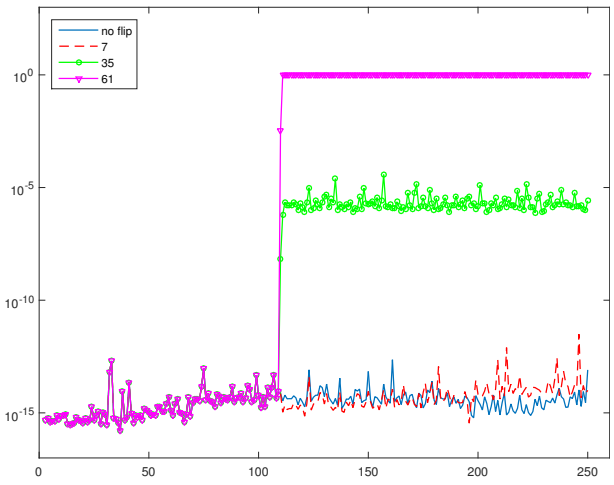

FIG. 3.2. *bcsstk01, relative difference d_k when flipping bits 7, 35 or 61 in (r_k,r_k) at iteration 110*

Figure 3.3 shows the result of flipping bits in a random component of Ap_{k-1} at iteration 110. The effects of flipping bits are more or less the same as for (r_k,r_k) even though the level of d_k when flipping bit 35 is smaller.

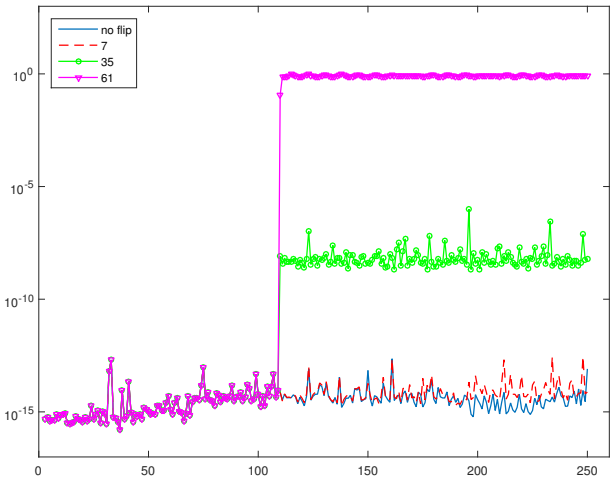

FIG. 3.3. *bcsstk01, relative difference d_k when flipping bits 7, 35 or 61 in one component of Ap_{k-1} at iteration 110*

Figure 3.4 displays d_k when flipping different bits in the output of (r_k, r_k) at iteration 110. The larger is the bit number, the larger is the value of d_k. Remember that 1 is the rightmost bit. Flipping a bit in the exponent field gives the largest value of d_k.

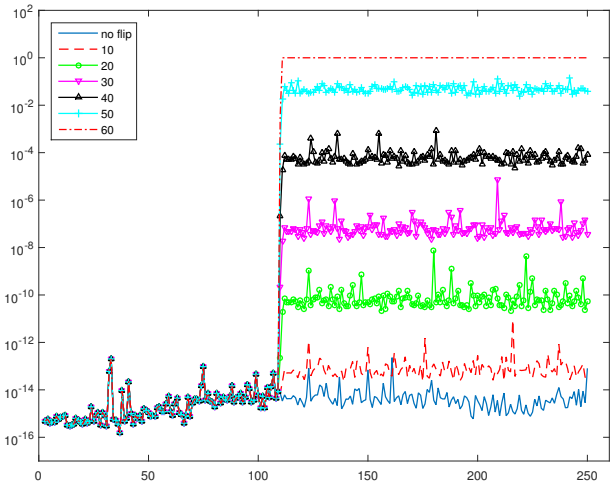

FIG. 3.4. *bcsstk01, relative difference d_k when flipping bits in (r_k, r_k) at iteration 110*

Let ε_d be a given threshold. We say that there was a fault in the computation if $d_k > \varepsilon_d$. Note that there can also be a fault when computing d_k. However, the computation of d_k, which only involves a few scalars, can be done redundantly to be sure that the result is correct.

Most of the time, CG is used with a preconditioner M which is a symmetric positive definite matrix. With preconditioning and defining z_k by $Mz_k = r_k$, the

relation (3.1) becomes

$$\alpha_{k-1}^2(Ap_{k-1}, M^{-1}Ap_{k-1}) = (r_k, z_k) + (r_{k-1}, z_{k-1}), \tag{3.4}$$

in exact arithmetic and we define d_k as

$$d_k = \frac{|\alpha_{k-1}(Ap_{k-1}, M^{-1}Ap_{k-1})^{\frac{1}{2}} - [(r_{k-1}, z_{k-1}) + (r_k, z_k)]^{\frac{1}{2}}|}{[(r_{k-1}, z_{k-1}) + (r_k, z_k)]^{\frac{1}{2}}},$$

It may seem that we have to do two linear solves with the matrix M at each iteration, but we will show how to avoid that in Section 5.

Of course, the main problem is to choose the threshold ε_d. The relative residual norms corresponding to the bit flips for Figure 3.4 are displayed in Figure 2.4. We can allow having the curves corresponding to bits 10 and 20, or even 30. In most of our numerical experiments we chose $\varepsilon_d = 10^{-12}$. However, this value of ε_d is a little too small for *bcsstk01*. It could give what are called *false positives* that is, errors which do not exist. Correcting this type of errors is useless and costly.

To check if our error detection works correctly we use a protocol similar to what was done in [1]. The stopping criterion for CG is $\|r_k\|/\|b\| \leq \varepsilon_c$ and the initial vector is the zero vector. We use *bcsstk01* and a subset of the matrices[1] used in [1]. The characteristics of the matrices are given in Table 3.1.

TABLE 3.1
Characteristics of the test matrices

Name	order	condition number
bcsstk01	48	8.8234 10^5
bodyy5	18,589	7.873336 10^3
bundle1	10,581	1.004238 10^3
crystm02	13,965	2.504738 10^2
crystm03	24,696	2.640325 10^2
dubcova1	16,129	9.971199 10^2
fv1	9,604	8.806042
fv3	9,801	2.026840 10^3
jnlbrng1	40,000	1.832266 10^2
kuu	7,102	1.575800 10^4
wathen100	30,401	5.816014 10^3

The exact solution x_{ex} for each run is a random vector with components in $[-1, 1]$ and the right-hand side is $b = Ax_{ex}$. We do 1000 runs for each matrix and preconditioner which are a diagonal preconditioner (diag) or an incomplete Cholesky factorization without fill-in (IC).

In case we do not flip a bit we check if an alarm was raised. Otherwise, we first do a run without flip to obtain the number m of iterations needed to satisfy the stopping criterion. Then, we do a run flipping a bit chosen randomly at iteration $\lfloor m/2 \rfloor$. If the bit flip is not detected but if the stopping criterion is nevertheless satisfied within $m + \lfloor m/2 \rfloor$ iterations, we consider that CG has converged.

The acronyms used in Tables 3.2 and 3.4 for the same categories as in [1] are defined below:

- nc: Number of the 1000 cases with a bitflip.

[1] They can be obtained at `https://sparse.tamu.edu`

- true positive (tp): A bitflip occurred, it prevented convergence, and an alert was raised.
- special positive (sp): A bitflip occurred, it did not prevent convergence, and an alert was raised.
- false positive (fp): No bitflip occurred and an alert was raised.
- true negative (tn): No bitflip occurred, and there was no alert.
- false negative (fn): A bitflip occurred, it prevented convergence, but there was no alert.
- special negative (sn): A bitflip occurred, it did not prevent convergence, and there was no alert.
- max it.: Maximum number of iterations to satisfy the convergence criterion for special negative cases.
- max bit: Maximum bit number perturbed for special negative cases.

What we would like to achieve is not to have false positives or false negatives.

For the results of Table 3.2, we do bit flips in the output of Ap_{k-1}. We see that for *bcsstk01* without preconditioning there are a few false positives because ε_d was a little too small. This does not happen for the other matrices. What is important is that there was no false negative. When a bit flip occurred and no alert was raised, there was convergence within the 50% additional number of iterations that were allowed. In these cases the last column shows what was the maximum number of the bits that were flipped. It is generally between 30 and 40.

The matrix *bcsstk01* has the largest condition number in the set of matrices in Table 3.1 and we may wonder if the value ε_d is well suited for matrices having a larger condition number, since the upper bound (6.1) in the Appendix is depending on $\kappa(A)$. To check this, let us consider the matrix *nos7* of order 729. Its condition number is $2.3745\ 10^9$. We do bit flips in the output of Ap_{k-1} using the same protocol as before, but with different values of ε_d. The results are shown in Table 3.3. We see that, with $\varepsilon_d = 10^{-12}$, we have many false positives, even with a diagonal or an incomplete Cholesky preconditioner. However, these preconditioners do not reduce too much the condition numbers which are respectively $1.2934\ 10^8$ and $9.0066\ 10^6$ for "diag" and "IC". We have to increase ε_d at least up to 10^{-8} to get rid of most of the false positives. Therefore, a "good" value of ε_d is somehow dependent on the condition number of the matrix, but so far, we do not have an efficient way to automatically choose ε_d, even though the condition number can be estimated during CG iterations, see [18]. Note also that, taking a too small value of ε_d and having false positives, is dangerous when using a recovery procedure because, if this is done without caution, it may lead to an infinite looping of the algorithm.

For Table 3.4, we do bit flips in the output of (r_k, z_k). The results are similar to those of Table 3.2 except for the number of true positives and for the maximum bit numbers which are generally around 15. This means that there were more cases without convergence. These results show that our detection method is working satisfactorily if a proper value of ε_d is chosen.

4. Correction of the silent errors. Having a way to test if there was a silent fault at iteration k, we can implement a recovery procedure.

Let us assume that we have at hand the values of the CG vectors and scalars at the beginning of iterations k and $k-1$. If we detect a fault, we go back to the beginning of iteration $k-1$. This is necessary because the fault could have happened in the computation of p_{k-1} at iteration $k-1$. We know that the data used for the recovery is correct because, otherwise an error would have been detected (and corrected) in

TABLE 3.2
Bit flip in Ap_{k-1}, $\varepsilon_d = 10^{-12}$, $\varepsilon_c = 10^{-10}$

matrix	prec.	nc	tp	sp	fp	tn	fn	sn	max it.	max bit
bcsstk01	no	909	0	680	10	81	0	229	150	39
	diag	898	382	276	0	102	0	240	49	24
	IC	886	0	593	0	114	0	293	18	24
bodyy5	no	895	9	437	0	105	0	449	724	40
	diag	909	29	470	0	91	0	410	444	36
	IC	875	32	385	0	125	0	458	54	43
bundle1	no	898	53	422	0	102	0	423	232	38
	diag	899	91	402	0	101	0	406	58	34
	IC	922	0	489	0	78	0	433	25	37
crystm02	no	915	42	478	0	85	0	395	138	32
	diag	906	49	417	0	94	0	440	56	33
	IC	898	0	314	0	102	0	584	3	43
crystm03	no	923	42	459	0	77	0	422	144	33
	diag	898	55	453	0	102	0	390	56	32
	IC	898	55	453	0	102	0	390	3	44
dubcova1	no	913	49	454	0	87	0	410	164	35
	diag	901	61	454	0	99	0	386	172	33
	IC	907	27	474	0	93	0	406	89	34
fv1	no	907	27	474	0	93	0	406	31	34
	diag	912	0	455	0	88	0	457	31	34
	IC	891	0	465	0	109	0	426	10	33
fv3	no	879	106	422	0	121	0	351	220	30
	diag	885	108	418	0	115	0	359	220	29
	IC	895	26	508	0	105	0	361	68	36
jnlbrng1	no	903	29	488	0	97	0	386	133	35
	diag	902	35	469	0	98	0	398	125	32
	IC	902	25	455	0	98	0	422	27	35
kuu	no	904	34	505	0	96	0	365	697	32
	diag	899	82	449	0	101	0	368	593	30
	IC	901	28	502	0	99	0	371	77	30
wathen100	no	881	27	454	0	119	0	400	319	34
	diag	901	26	466	0	99	0	409	48	36
	IC	921	0	456	0	79	0	465	13	38

iteration $k-1$. The recovery procedure is implemented in Algorithm 2 when using a preconditioner M. For simplicity we assume that there is no error in the initialization phase and that the data (A, b) was correct. In Algorithm 2, at each iteration we have to compute $M^{-1}r_k$ and $M^{-1}Ap_{k-1}$. These two solves are independent but if $M = LL^T$ and if we are able to work with the matrix $\tilde{A} = L^{-T}AL^{-1}$ we could avoid one of these two solves. The product $w = \tilde{A}p_{k-1}$ is then obtained as

$$Ly = p_{k-1}, \quad L^T w = Ay.$$

We will see another way to avoid one of the two solves with the matrix M in Section 5.

Figure 4.1 shows the norms of the true residuals when flipping bit 60 of the output of (r_k, z_k) at iteration 110 but with the recovery procedure for *bcsstk01*. There is a delay when using the recovery procedure because we did not update the iteration number after the recovery.

TABLE 3.3
Bit flip in Ap_{k-1}, nos7, $\varepsilon_c = 10^{-10}$

ε_d	prec.	nc	tp	sp	fp	tn	fn	sn	max it.	max bit
10^{-12}	no	897	70	827	103	0	0	0	-	-
	diag	901	40	844	97	2	0	17	50	33
	IC	889	40	843	110	1	0	6	17	24
10^{-10}	no	901	67	751	84	15	0	83	1524	41
	diag	895	35	766	78	27	0	94	90	37
	IC	907	30	806	84	9	0	71	28	41
10^{-8}	no	912	77	354	6	82	0	481	2086	62
	diag	913	29	296	0	87	0	588	91	61
	IC	901	38	251	0	99	0	612	28	62
10^{-6}	no	900	76	225	0	100	0	599	2139	63
	diag	897	34	202	0	103	0	661	93	64
	IC	886	25	138	0	114	0	723	29	64

Algorithm 2 Preconditioned Conjugate Gradient with silent fault check and recovery

input A, b, M x_0, ε_d
$r_0 = b - Ax_0$, $r^{oo} = r_0$
Solve $Mz_0 = r_0$
$p_0 = z_0$, $p^{oo} = p_0$, $z^{oo} = z_0$
$rtr = r_0^T z_0$, $rtr^{oo} = rtr$, $x^{oo} = x_0$
for $k = 1, \ldots$ until convergence **do**
  $r^o = r_{k-1}$, $z^o = z_{k-1}$, $p^o = p_{k-1}$, $rtr^o = rtr$, $x^o = x_{k-1}$
  $\alpha_{k-1} = \frac{rtr}{p_{k-1}^T Ap_{k-1}}$
  $x_k = x_{k-1} + \alpha_{k-1} p_{k-1}$
  $r_k = r_{k-1} - \alpha_{k-1} Ap_{k-1}$
  $d_1 = \alpha_{k-1} \left[Ap_{k-1}\right]^T M^{-1} Ap_{k-1}$
  Solve $Mz_k = r_k$
  $rtrk = r_k^T z_k$
  $\beta_k = \frac{rtrk}{rtr}$
  $d_2 = (rtr + rtrk)^{\frac{1}{2}}$
  $rtr = rtrk$
  $p_k = z_k + \beta_k p_{k-1}$
  $d_k = \frac{|d_1 - d_2|}{d_2}$
  if $k > 2$ and $d_k > \varepsilon_d$ **then**
    $k = k - 2$
    $p_k = p^{oo}$, $r_k = r^{oo}$, $z_k = z^{oo}$, $x_k = x^{oo}$, $rtr = rtr^{oo}$
  else
    $p^{oo} = p^o$, $r^{oo} = r^o$, $z^{oo} = z^o$, $x^{oo} = x^o$, $rtr^{oo} = rtr^o$
  end if
end for

TABLE 3.4
Bit flip in (r_k, z_k), $\varepsilon_d = 10^{-12}$, $\varepsilon_c = 10^{-10}$

matrix	prec.	nc	tp	sp	fp	tn	fn	sn	max it.	max bit
bcsstk01	no	885	175	610	16	99	0	100	150	20
	diag	895	384	353	0	105	0	158	49	15
	IC	897	0	706	0	103	0	191	18	16
bodyy5	no	891	153	516	0	109	0	222	724	15
	diag	905	172	532	0	95	0	201	445	16
	IC	907	171	506	0	93	0	230	54	16
bundle1	no	906	327	414	0	94	0	165	233	15
	diag	889	142	538	0	111	0	209	58	15
	IC	889	142	538	0	111	0	209	25	14
crystm02	no	903	176	505	0	97	0	222	138	16
	diag	907	178	534	0	93	0	195	56	14
	IC	907	0	727	0	93	0	180	3	13
crystm03	no	897	175	516	0	103	0	206	144	16
	diag	906	165	533	0	94	0	208	56	15
	IC	901	0	726	0	99	0	175	3	14
dubcova1	no	904	322	361	0	96	0	221	164	16
	diag	891	328	354	0	109	0	209	172	16
	IC	901	213	466	0	99	0	222	89	16
fv1	no	889	149	544	0	111	0	196	32	14
	diag	894	161	551	0	106	0	182	31	14
	IC	913	0	739	0	87	0	174	10	13
fv3	no	911	326	379	0	89	0	206	220	16
	diag	902	266	389	0	98	0	247	221	16
	IC	897	209	487	0	103	0	201	68	16
jnlbrng1	no	903	143	541	0	97	0	219	133	16
	diag	899	162	512	0	101	0	225	125	16
	IC	901	177	530	0	99	0	194	27	14
kuu	no	890	264	412	0	110	0	214	694	16
	diag	919	213	490	0	81	0	216	593	16
	IC	897	275	398	0	103	0	224	77	16
wathen100	no	898	298	388	0	102	0	212	319	16
	diag	896	166	526	0	104	0	204	48	15
	IC	916	0	732	0	84	0	184	13	14

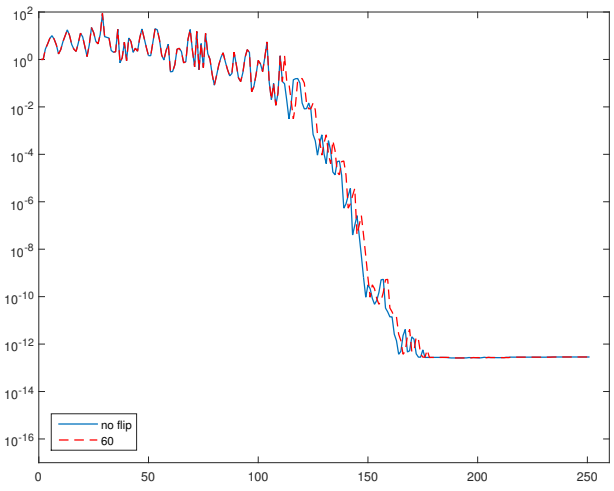

FIG. 4.1. *bcsstk01, relative true residual norms when flipping bit 60 in* (r_k, r_k) *at iteration 110 with recovery*

5. A parallel variant of CG with silent error detection and correction. When looking at Algorithm 2, it seems, as we said above, that we have to solve two linear systems with the matrix M at each iteration. Even though the two solves can be done in parallel, this can also be avoided by using the predictor-corrector CG variant introduced in [16]. Formula (3.4) is used to predict the value of $r_k^T z_k$ which is used to compute β_k and p_k. The value of $r_k^T z_k$ is recomputed at the end of the iteration to prevent instability. This method is described in Algorithm 3. It is mathematically equivalent to standard CG, but not in finite precision arithmetic, see [16, 4].

We check the equality (3.4) using the corrected value of $r_k^T z_k$. Since the stability of the method is a little different than for standard CG and because we are using (3.4) for two different purposes, we had to slightly modify the criterion for the detection of silent errors, adding a test on the numerator. We did so because the relative difference can sometimes become large without a bit flip.

Algorithm 3 Parallel preconditioned Conjugate Gradient with silent fault check and recovery

input A, b, x_0, ε_d
$r_0 = b - Ax_0$, $r^{oo} = r_0$
$p_0 = r_0$, $p^{oo} = p_0$
Solve $Mz_0 = r_0$
$rtr = r_0^T z_0$, $rtr^{oo} = rtr$, $x^{oo} = x_0$
for $k = 1, \ldots$ until convergence **do**
  $r^o = r_{k-1}$, $z^o = zk - 1$, $p^o = p_{k-1}$, $rtr^o = rtr$, $x^o = x_{k-1}$
  Solve $Mv = Ap_{k-1}$
  $vAp = v^T Ap_{k-1}$, $pAp = p_{k-1}^T Ap_{k-1}$
  $\alpha_{k-1} = \frac{rtr}{pAp}$
  $d_1 = \alpha_{k-1}\sqrt{vAp}$
  $s = d_1^2 - rtr$ % prediction
  $\beta_k = \frac{s}{rtr}$
  $x_k = x_{k-1} + \alpha_{k-1} p_{k-1}$
  $r_k = r_{k-1} - \alpha_{k-1} Ap_{k-1}$
  $z_k = z_{k-1} - \alpha_{k-1} v$
  $p_k = (z_{k-1} - \alpha_{k-1} v) + \beta_k p_{k-1}$
  $rtrk = r_k^T z_k$ % correction
  $d_2 = (rtr + rtrk)^{\frac{1}{2}}$
  $rtr = rtrk$
  $d_k = \frac{|d_1 - d_2|}{d_2}$
  if $k > 2$ and $d_k > \varepsilon_d$ and $|d_1 - d_2| > \varepsilon_d$ **then**
    $k = k - 2$
    $p_k = p^{oo}$, $r_k = r^{oo}$, $z_k = z^{oo}$, $x_k = x^{oo}$, $rtr = rtr^{oo}$
  else
    $p^{oo} = p^o$, $r^{oo} = r^o$, $z^{oo} = z^o$, $x^{oo} = x^o$, $rtr^{oo} = rtr^o$
  end if
end for

Table 5.1 shows the same quantities as in Table 3.2 when we flip a bit in the output of Ap_{k-1}. However, three of the test matrices are badly scaled and, with Algorithm 3, it was not possible to use the same value of ε_d as for the other matrices. This problem

was solved by scaling these three matrices by the maximum of the absolute values of their entries. In these computations we used $\varepsilon_d = 10^{-10}$ for all matrices.

What is different from the previous tables is the maximum bit number. There are cases where a bit flip in the exponent field does not prevent convergence. This is due to the slight change in the detection criterion. However, for most special negative cases the maximum bit number was smaller than 40. We observe that there were no false positives and false negatives.

TABLE 5.1
Bit flip in Ap_{k-1}, $\varepsilon_d = 10^{-10}$, $\varepsilon_c = 10^{-10}$, *Algorithm 3*

matrix	prec.	nc	tp	sp	fp	tn	fn	sn	max it.	max bit
bcsstk01	no	892	4	561	0	108	0	327	180	62
	diag	908	345	210	0	92	0	353	50	38
	IC	900	0	550	0	100	0	350	18	38
bodyy5	no	897	1	408	0	103	0	488	728	63
	diag	891	20	332	0	109	0	539	449	63
	IC	892	15	300	0	108	0	577	54	63
bundle1	no	897	20	124	0	103	0	753	255	64
scaled	diag	901	13	202	0	99	0	686	58	63
	IC	901	0	147	0	99	0	754	25	64
crystm02	no	906	15	208	0	94	0	683	139	63
scaled	diag	900	16	235	0	100	0	649	59	63
	IC	913	0	231	0	87	0	682	3	63
crystm03	no	906	14	205	0	94	0	687	144	63
scaled	diag	900	18	199	0	100	0	683	56	63
	IC	920	0	213	0	80	0	707	3	64
dubcova1	no	900	59	227	0	100	0	614	164	63
	diag	914	64	200	0	86	0	650	172	64
	IC	900	16	235	0	100	0	649	89	63
fv1	no	896	0	285	0	104	0	611	31	63
	diag	881	0	275	0	119	0	606	31	63
	IC	891	0	236	0	109	0	655	10	63
fv3	no	914	41	255	0	86	0	618	221	63
	diag	899	38	219	0	101	0	642	221	63
	IC	903	10	245	0	97	0	648	68	63
jnlbrng1	no	891	7	244	0	109	0	640	133	63
	diag	906	8	216	0	94	0	682	125	63
	IC	895	0	246	0	105	0	649	27	63
kuu	no	911	19	276	0	89	0	616	696	63
	diag	884	13	259	0	116	0	612	598	63
	IC	914	14	266	0	86	0	634	78	63
wathen100	no	900	21	274	0	100	0	605	319	63
	diag	900	3	272	0	100	0	625	48	63
	IC	910	0	300	0	90	0	610	13	63

Figure 5.1 shows the true residual norms when flipping bit 60 of the output of (r_k, z_k) at iteration 50 but with the recovery procedure for the matrix *kuu* with an incomplete Cholesky preconditioner. As above, there is a delay when using the recovery procedure because we did not update the iteration number after the recovery.

6. Conclusion. In this paper, we have proposed a new way to detect and correct silent errors in the conjugate gradient algorithm. The detection criterion is simple

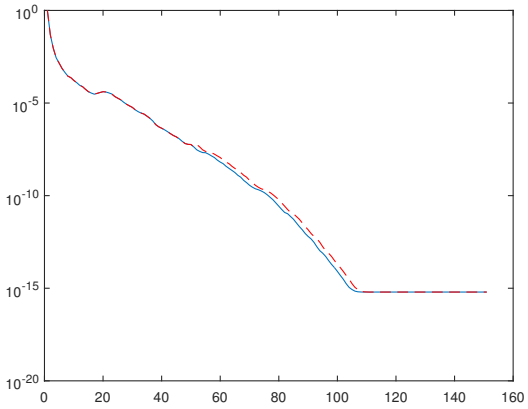

FIG. 5.1. *kuu, incomplete Cholesky preconditioner, relative true residual norms when flipping bit 60 in* (r_k, z_k) *at iteration 50 with recovery*

and cheap to implement. When using a preconditioner it is nevertheless preferable to use the predictor-corrector variant of CG. Numerical experiments show that the new criterion is robust and reliable. One of the advantages of our method, besides its simplicity, is that the check for errors can be done at every iteration, minimizing the extra computations that have to be done for the recovery.

Appendix: CG local orthogonality. Using (3.2) it can be shown that

$$|(r_k, r_{k-1})| \leq \kappa(A) \frac{\|r_{k-1}\|}{\|p_{k-1}\|} \left[C^A_{k-1} u \frac{\|r_{k-1}\|}{\|p_{k-1}\|} \frac{\|r_{k-1}\|^2}{\|r_{k-2}\|^2} + \|r_{k-1}\| \, \|\delta^p_{k-2}\| \right] + \|\delta^r_{k-1}\| \, \|r_{k-1}\|, \tag{6.1}$$

where $\kappa(A)$ is the condition umber of A and C^A_{k-1} is a constant involved in the bound $|(Ap_{k-2}, p_{k-1})| \leq \lambda_n C^A_{k-1} u$ where λ_n is the largest eigenvalue of A.

The three terms in the right-hand side of inequality (6.1) are small provided that the ratios are bounded and $\kappa(A)$ is not too large. Local orthogonality is, in general, well satisfied.

Acknowledgments. The author thanks Erin Carson for interesting comments and suggestions.

REFERENCES

[1] E. AGULLO, S. COOLS, E. FATIH-YETKIN, L. GIRAUD, N. SCHENKELS AND W. VANROOSE, *On soft errors in the Conjugate Gradient method: Sensitivity and robust numerical detection*, SIAM J. Sci. Comput., v 42, n 6 (2020), pp. C-335-C358.

[2] P.G. BRIDGES, K.B. FERREIRA, M.A. HEROUX AND M. HOEMMEN, *Fault-tolerant linear solvers via selective reliability*, arXiv preprint arXiv:1206.1390, (2012).

[3] G. BRONEVETSKY AND B. DE SUPINSKI, *Soft error vulnerability of iterative linear algebra methods*, in Proceedings of the 22nd Annual International Conference on Supercomputing, ICS'08, ACM, New York, USA, (2008), pp. 155-164.

[4] T. CHEN AND E. CARSON, *Predict-and-recompute conjugate gradient variants*, SIAM J. Sci. Comput., v 42 n 5 (2020), pp. A3084-A3108.

[5] Z. Chen, *Online-ABFT: An online algorithm based fault tolerance scheme for soft error detection in iterative methods*, in Proc. 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP'13, (2013), pp. 167-176.

[6] J. Elliott and M. Hoemmen, *Quantifying the impact of single bit flips in GMRES*, CSRI Summer Proceedings 2013, (2014), pp. 10-31.

[7] J. Elliott, M. Hoemmen and F. Mueller, *Evaluating the impact of SDC on the GMRES iterative solver*, 2014 IEEE 28th International Parallel and Distributed Processing Symposium, IEEE, (2014).

[8] J. Elliott, M. Hoemmen and F. Mueller, *Resilience in numerical methods: a position on fault models and methodologies*, arXiv preprint arXiv:1401.3013, (2014).

[9] J. Elliott, M. Hoemmen and F. Mueller, *A numerical soft fault model for iterative linear solvers*, Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing, (2015).

[10] J. Elliott, M. Hoemmen and F. Mueller, *Exploiting data representation for fault tolerance*, Research Report SAND-2016-0354J, Sandia National Laboratory, (2016).

[11] J. Elliott, F. Mueller, F. Stoyanov and C. Webster, *Quantifying the impact of single bit flips on floating point arithmetic*, Report North Carolina State University, Dept. of Computer Science, (2013).

[12] M. Fasi, J. Langou, Y. Robert and B. Uçar, *A backward/forward recovery approach for the preconditioned conjugate gradient method*, J. Comput. Sci., v 17 (2016), pp. 522-534.

[13] N. J. Higham, *Accuracy and stability of numerical algorithms*, SIAM (2002).

[14] K.-H. Huang and J.A. Abraham, *Algorithm-based fault tolerance for matrix operations*, IEEE transactions on computers, v 100 n 6 (1984), pp. 518-528.

[15] G. Kestor, B.O. Mutlu, J. Manzano, O. Subasi, O. Unsal and S. Krishnamoorthy, *Comparative analysis of soft-error detection strategies: A case study with iterative methods*, in Proceedings of the 15th ACM International Conference on Computing Frontiers (2018), pp. 173-182.

[16] G. Meurant, *Multitasking the conjugate gradient method on the CRAY X-MP/48*, Parallel Computing, v 5, (1987), pp. 267-280.

[17] G. Meurant, *The Lanczos and Conjugate Gradient algorithms, from theory to finite precision computations*, SIAM (2006).

[18] G. Meurant and P. Tichý, *Approximating the extreme Ritz values and upper bounds for the A-norm of the error in CG*, Numer. Algorithms, v 82 n 3 (2019), pp. 937-968.

[19] B.O. Mutlu, G. Kestor, J. Manzano, O. Unsal, S. Chatterjee, S. and S. Krishnamoorthy, *Characterization of the impact of soft errors on iterative methods*, in 2018 IEEE 25th International Conference on High Performance Computing (HiPC), pp. 203-214.

[20] Z. Rubenstein, H. Fujita, Z. Zheng and A. Chien, *Error checking and snapshot-based recovery in a preconditioned conjugate gradient solver*, Technical Report TR-2013-11, Department of Computer Science, University of Chicago, (2013).

[21] Y. Saad, *Practical use of polynomial preconditionings for the conjugate gradient method*, SIAM J. Sci. Stat. Comput., v 6 n 4 (1985), pp. 865-881.

[22] P. Sao and R. Vuduc, *Self-stabilizing iterative solvers*, in Proceedings of the Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, (2013).

[23] A. Schöll, C. Braun, M.A. Kochte and H.-J. Wunderlich, *Low-overhead fault-tolerance for the preconditioned conjugate gradient solver*, in Proceedings of the International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS'15), Amherst, Massachusetts, USA, (2015).

[24] M. Shantharam, S. Srinivasmurthy and P. Raghavan, *Fault tolerant preconditioned conjugate gradient for sparse linear system solution* in Proceedings of the 26th ACM international conference on Supercomputing, ICS'12, New York, USA, (2012), pp. 69-78.