On soft errors in the conjugate gradient method: sensitivity and robust numerical detection

Emmanuel Agullo, Siegfried Cools, Emrullah Fatih-Yetkin, Luc Giraud, Nick Schenkels, Wim Vanroose

ON SOFT ERRORS IN THE CONJUGATE GRADIENT METHOD: SENSITIVITY AND ROBUST NUMERICAL DETECTION

EMMANUEL AGULLO*, SIEGFRIED COOLS†, EMRULLAH FATIH YETKIN‡, LUC GIRAUD*, NICK SCHENKELS*, AND WIM VANROOSE†

Abstract. The conjugate gradient (CG) method is the most widely used iterative scheme for the solution of large sparse systems of linear equations when the matrix is symmetric positive definite. Although more than sixty year old, it is still a serious candidate for extreme-scale computations on large computing platforms. On the technological side, the continuous shrinking of transistor geometry and the increasing complexity of these devices affect dramatically their sensitivity to natural radiation, and thus diminish their reliability. One of the most common effects produced by natural radiation is the single event upset which consists in a bit-flip in a memory cell producing unexpected results at application level. Consequently, the future extreme scale computing facilities will be more prone to errors of any kind, including bit-flips, during their calculations. These numerical and technological observations are the main motivations for this work, where we first investigate through extensive numerical experiments the sensitivity of CG to bit-flips in its main computationally intensive kernels, namely the matrix-vector product and the preconditioner application. We further propose numerical criteria to detect the occurrence of such soft errors and assess their robustness through extensive numerical experiments.

1. Introduction. The flexibility offered by Von Neumann machines for programming complex algorithms motivated the quest for the design of many innovative numerical algorithms in the late 40's. Computers being considered as unreliable machines, it was not rare that the presentation of those algorithms was accompanied with advanced numerical considerations for distinguishing "normal rounding-off errors" due to digital computation from "errors of the computers", using the expressions from [**?**], which introduced the Conjugate Gradient (CG) algorithm, a seminal paper of that unprecedented fertile period. This numerical technique is still a method of choice for solving large sparse linear systems of equations involving a symmetric positive definite (SPD) matrix. While the study of numerical correctness since led to a continuous, productive research field [**?**], the tremendous progress in hardware design progressively reduced the interest of detecting *soft* errors (using the modern expression for characterizing errors that do not lead to an immediate failure of a program). At some extra energy cost, hardware mechanisms such as Error Correcting Codes (ECC) were indeed able to correct most soft errors so that, from a numerical design point of view, they could almost be considered as – and indeed often were – a non-existent problem.

The advent of distributed-memory platforms and networks of heterogeneous workstations in the 90's once again drew the attention of the computational science community to faults leading to errors, as long running parallel executions were regularly aborted because of the crash or the non response of only a single processing unit. For this reason, early high-level parallel libraries for programming these parallel platforms such as the Parallel Virtual Machine (PVM) [**?**] proposed primitives (`hostfailentry()` in the revision 3 of the PVM standard for instance) allowing an application to design tailored schemes for recovering from such *hard* errors (using modern terminology). From the numerical perspective, one major breakthrough was the introduction of algorithm-based fault tolerance (ABFT) schemes [**?**] to detect and correct errors when

*Inria, France
†Applied Mathematics Group, University of Antwerp, Belgium
‡Management Information Systems Department, Kadir Has University, Turkey

matrix operations such as addition, multiplication, scalar product, LU-decomposition, and transposition are performed using multiple processor systems. The proposed method could "detect and correct any failure within a single processor in a multiple processor system", using the words of the authors. On the other hand, the operating system community developed efficient checkpoint-restart (CPR) mechanisms. Together with the progress of interconnect network technologies, their ability to handle those faults in a transparent way for the application once again allowed programmers to view (parallel) computers as reliable computing platforms. As a consequence, in spite of solid proposals (such as [**?**]), the most widely employed interface for programming distributed-memory machines today, the Message Passing Interface (MPI) standard, does not provide support allowing applications to design customized resilient schemes, even in its revision 3.1 [**?**], the most recent standard at the time of writing the present article.

As the size of transistors continues to reduce and the number of components continues to increase, soft errors in supercomputers become more and more common. In the fault tolerance literature, many techniques have been proposed to detect and/or correct soft errors. The best-known general technique to detect soft errors is the double modular redundancy (DMR) approach. This approach either uses two different pieces of hardware to perform the same computation at the same time or performs the same computation on the same hardware twice, then compares the two results to detect whether errors occur or not. The most well-known general technique to correct single soft errors is the triple modular redundancy (TMR) approach. TMR either performs the same computation on three different pieces of hardware or uses the same hardware to perform the same computation three times, then compares and selects the results obtained consistently at least twice as the correct result. While DMR and TMR are very general, their overhead is high - up to 100% overhead to detect errors and 200% overhead to correct errors. To protect memory against corruption, ECC memory has been widely used by many computer vendors. Although today's ECC memory can detect and correct bit-flips in memory, it brings significant overhead in space, time, and energy. Furthermore, ECC memory is not able to handle computational (*i.e.*, arithmetic) errors that are caused by faults in logic units. New types of unreliable hardware, such as *in-memory* computers [**?**] (which are no longer Von Neumann machines), are emerging as serious competitors (or, more likely, accelerators) to traditional processors, as they are expected to achieve a much higher energy efficiency.

As the preconditioned version of CG, PCG, remains the Krylov subspace method of choice for solving large sparse SPD linear systems, the goal of this paper is to study its sensitivity to soft errors and propose numerical remedies to detect them. Although much truncated, the above very brief journey through half a dozen decades of the modern computing era shows that our concern for the reliability of machines has been slowly but constantly evolving. This motivates us to characterize these errors from a high-level point of view, disregarding low-level hardware details, may the underneath silicon be an exascale machine, the initial motivation for this work, or, potentially, an embedded computer in high altitude [**?**], or any other type of unreliable digital computer (such as [**?**]). We therefore only assume that soft errors may occur, corrupting data or computation, without the hardware or system detecting them and notifying the application that an error occurred. We call this type of soft error, "silent data corruption" (SDC) (after [**?**]), which may cause a simulation to silently return an incorrect answer that does not reflect any sensitivity of the solution to the input

data and consequently could lead to a misleading analysis of the computed outcome. We focus on such errors in the present study and we refer the reader to [**?**] and the references therein for a recent overview on the resilience and fault tolerance issues in HPC.

The first contribution of this paper is to study the sensitivity of PCG to such soft errors. We mainly focus on its main computational kernels, *i.e.*, the matrix-vector product and the preconditioner application. In particular, we investigate the sensitivity to soft errors for various space and time locations.

The second contribution is the proposition of two numerical criteria to detect these soft errors. The PCG method is a very sophisticated and elegant numerical scheme that has many properties induced by the symmetric positive definiteness of the matrix A. Unfortunately, most characteristic properties are no longer valid in finite precision calculation. On the other hand, a lot of work has been devoted to study PCG in finite precision [**?**, **?**]. We therefore consider some of the finite precision results to define a first possible numerical soft error detection mechanism based on the residual gap. We consider an additional criterion, which was already proposed in the original paper on CG [**?**, Thm 5.5], based on a bound (as opposed to a strict equality), hence expected to be less prone to defection due to finite precision calculation. We study the respective quality of both these criteria and show that, combined together, they are extremely efficient to detect soft errors, occurring not only in the matrix-vector product and preconditioner application, but also in all the operations involved in PCG.

The rest of the paper is organized as follows. In Section 2, we briefly present the PCG algorithm, we review the main classical error detection techniques and we detail round-off and soft error modelling. In Section 3, we then study the sensitivity of PCG to soft errors occurring in its main computational kernels that are the matrix-vector product and the preconditioner application. In particular, we investigate the dependency on the bit and temporal location of the error. In Section 4, we propose numerical criteria to detect soft errors in those two kernels and assess their robustness through extensive numerical experiments. Finally, we summarize the contributions of this paper in Section 5 and draw up some perspectives for future works.

2. Background.

2.1. Preconditioned Conjugate Gradient algorithm. PCG is a Krylov subspace method for solving a large linear system

$$Ax = b$$

where $A \in \mathbb{R}^{n\times n}$ is symmetric positive definite (SPD), the right-hand side $b \in \mathbb{R}^n$ and the solution $x \in \mathbb{R}^n$. In exact arithmetic, the method converges within at most n iterations. In practice with finite precision calculation, this property does not hold and preconditioning [**?**, **?**] is often needed to accelerate the convergence. While in the context of Krylov subspace methods, PCG is a sophisticated, elegant and powerful numerical algorithm [**?**, **?**, **?**], its algorithm looks fairly simple and can be written as depicted in Algorithm 1, where M defines the preconditioner.

2.2. Double Modular Redundancy. Although potentially costly, DMR is certainly the best-known general agnostic technique for detecting soft errors. It consists in duplicating both the operations and data, and checking the equality of the output of the redundant calculations. For classical PCG, soft error detection with such a full duplication technique can be implemented as described in Algorithm 2.

Algorithm 1 Preconditioned Conjugate Gradient

1: $r_0 := b - Ax_0$; $u_0 = M^{-1}r_0$; $p_0 := u_0$; $\gamma_0 := r_0^T u_0$
2: **for** $i = 0, \ldots$ **do**
3: $\quad s_i := Ap_i$
4: $\quad \alpha_i := \gamma_i / s_i^T p_i$
5: $\quad x_{i+1} := x_i + \alpha_i p_i$
6: $\quad r_{i+1} := r_i - \alpha_i s_i$
7: $\quad u_{i+1} := M^{-1} r_{i+1}$
8: $\quad \gamma_{i+1} := r_{i+1}^T u_{i+1}$
9: $\quad \beta_{i+1} := \gamma_{i+1} / \gamma_i$
10: $\quad p_{i+1} := u_{i+1} + \beta_{i+1} p_i$
11: **end for**

Algorithm 2 Preconditioned Conjugate Gradient with DMR detection

1: $r_0^1 := b - Ax_0$; $p_0^1 := r_0^1$; $r_0^2 := b - Ax_0$; $p_0^2 := r_0^2$
2: **for** $i = 0, \ldots$ **do**
3: $\quad s^1 := Ap_i^1$; $s^2 := Ap_i^2$; check$(s^1 == s^2)$
4: $\quad \alpha^1 := (r_i^1, r_i^1)/(s^1, p_i^1)$; $\alpha^2 := (r_i^2, r_i^2)/(s^2, p_i^2)$; check$(\alpha^1 == \alpha^2)$
5: $\quad x_{i+1}^1 := x_i^1 + \alpha^1 p_i^1$; $x_{i+1}^2 := x_i^2 + \alpha^2 p_i^2$; check$(x_{i+1}^1 == x_{i+1}^2)$
6: $\quad r_{i+1}^1 := r_i^1 - \alpha^1 s^1$; $r_{i+1}^2 := r_i^2 - \alpha^2 s^2$; check$(r_{i+1}^1 == r_{i+1}^2)$
7: $\quad u_{i+1}^1 := {M^{-1}}^1 r_{i+1}^1$; $u_{i+1}^2 := {M^{-1}}^2 r_{i+1}^2$; check$(u_{i+1}^1 == u_{i+1}^2)$
8: $\quad \beta^1 := (r_{i+1}^1, u_{i+1}^1)/(r_i^1, u_i^1)$; $\beta^2 := (r_{i+1}^2, u_{i+1}^2)/(r_i^2, u_i^2)$; check$(\beta^1 == \beta^2)$
9: $\quad p_{i+1}^1 := u_{i+1}^1 + \beta^1 p_i^1$; $p_{i+1}^2 := u_{i+1}^2 + \beta^2 p_i^2$; check $(p_{i+1}^1 == p_{i+1}^2)$
10: **end for**

This duplication process is simple, generic and effective. However, the price for this detection technique may not be affordable with respect to computational and storage costs. Level-1 BLAS operations have a moderate computational cost so that their duplication may be considered acceptable under certain conditions. On the other hand, duplication of the two most computationally intensive kernels that are the matrix-vector product and preconditioning may not be affordable. Alternatively, checksum techniques can be employed to protect and check the correctness of these last two kernels.

2.3. Checksum techniques. Detecting soft errors in matrix calculation with checksum techniques was first proposed in [**?**] and further investigated for applications with iterative methods in [**?**]. To protect a matrix-vector product the main idea relies on the following equality that holds in exact arithmetic for any vector $v \in \mathbb{R}^n$:

$$\mathbb{I}^T(Av) = (\mathbb{I}^T A)v$$

where $\mathbb{I}$ is the vector of $\mathbb{R}^n$ with all entries equal to one. Each entry ℓ of the row vector $c^T = (\mathbb{I}^T A)$ is the checksum of the ℓ^{th} column of A that can be securely computed before starting PCG. At each PCG iteration, checking the correctness of the matrix-vector product reduces to test the equality:

$$c^T p_{i-1} = \mathbb{I}^T s_{i-1}.$$

The left-hand side requires an extra dot-product calculation and the right-hand side is simply the sum of all the entries of the output vector s_{i-1}. Because of the symmetry of

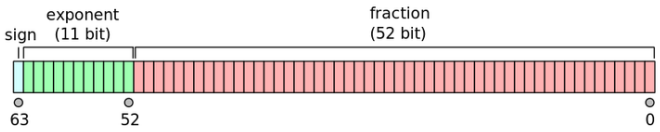


Figure 2.1: IEEE 754 double-precision binary floating-point format.

A the checksum vector can also be computed by $c = A\mathbb{I}$ so that the idea can be applied in a matrix-free context as well as for the preconditioner application calculation.

In finite precision arithmetic, there may not be a bitwise equality between $c^T p_{i-1}$ and $\mathbb{I}^T s_{i-1}$ because of the non-associativity of the sum in presence of round-off errors [**?**]. To assess the correctness of the matrix-vector product, we cannot simply check the bitwise equality and instead need to consider that the calculation is correct if

$$\frac{|c^T p_{i-1} - \mathbb{I}^T s_{i-1}|}{|c^T p_{i-1}|} \leq \tau, \tag{2.1}$$

where the threshold τ has to be safely defined. To prevent the occurrence of too many false detection instances (referred to as false-positives), the threshold τ needs to be chosen large enough. However, τ cannot be chosen arbitrarily large; otherwise actual errors would be missed. Consequently, the checksum-based detection methods need some tuning of the threshold τ with respect to round-off effects to achieve effectiveness, as further discussed in Section 4.2.1.

2.4. Round-off and soft error models.

2.4.1. Round-off errors. Round-off errors are conservatively modelled with the IEEE 754 specification. In particular, we rely on the IEEE 754 double-precision binary floating-point format, referred to as binary64 and illustrated in Figure 2.1. Let us denote $(b_i)_{i=0}^{63}$ the sequence of bits defining a double-precision number d where $(b_i)_{i=0}^{51}$ defines the mantissa/fraction, $(b_i)_{i=52}^{62}$ the exponent and b_{63} the sign so that

$$d = (-1)^{b_{63}} 2^{e-1023} (1+m), \text{with } e = \sum_{i=52}^{62} b_i 2^{i-52} \text{ and } m = \sum_{i=0}^{51} b_i 2^{i-52}. \tag{2.2}$$

The accuracy and stability of numerical algorithms in the presence of round-off errors is still an intense field of research [**?**] and we will rely on such finite precision results to define a first possible soft error detection mechanism in Section 4.1.1.

2.4.2. Soft errors. In addition to round-off errors (the "normal rounding-off errors" in [**?**]), soft errors (the "errors of the computers" in [**?**]) may occur. As discussed in the introduction of the present paper, a brief journey through the modern computing era motivates us to characterize soft errors from a high-level point of view, disregarding low-level hardware details. While there is a substantial literature for characterizing errors based on their hardware characteristics, much less effort has been devoted to characterize them from a high-level point of view. However, quoted in [**?**], Rees argued back in 1997 that "failure is a matter of function only [and is thus] related to purpose, not to whether an item is physically intact or not". M. Hoemmen and M. Heroux proposed a fault characterization in that perspective [**?**]. In [**?**], J. Elliot, M. Hoemmen and F. Mueller also consider a type of fault they do characterize from a high-level

point of view, *i.e.*, in their case, "*a fault that silently introduces bad data, while not persistently tainting the data that was used in the calculation. For example, let* $a = 2$ *and* $b = 2$*, then* $c = a + b = 10$." They acknowledge that "*while simplistic, this model presumes no knowledge of the nature of the fault, only that* c *is incorrect. This model assumes that the machine is unreliable in an unpredictable way, and therefore [they] are skeptical of the output it presents.*"

We consider that model referred to as silent data corruption (SDC) in [**?**] and refine it, depending on (1) whether the input data also gets tainted once the operation has completed, (2) at which stage the perturbation occurs and (3) how the perturbation is incurred. First, we distinguish whether the input data (a and b in the above example) is tainted or not once the operation has completed. Assume, for instance, that a gets tainted (say, $a = 8$ instead of 2) during the execution of the operation, eventually leading to the perturbation of c ($c = 10$). If the perturbation on a occurred while a was in a *persistent* memory (such as the main memory), a remains tainted once the operation has completed. We refer to this case as a *persistent* soft error. In the above example, the final state would be: $a = 8$, $b = 2$, $c = 10$. This type of error can be detected if the input data have been stored redundantly without the need of performing the computation redundantly. On the contrary, if the perturbation on a occurred while a was in *transient* memory (such as a cache), a does not remain tainted once the operation has completed. Following [**?**, **?**] terminology, we refer to this case as a *transient* soft error. In the example, the final state would be: $a = 2$, $b = 2$, $c = 10$. This second type of error cannot be detected relying only on input data redundancy and is thus more critical. We therefore focus only on such transient soft errors in the remainder of the paper (we present results for an analogous study on persistent soft errors in [**?**]).

The second refinement we make explicit with respect to the model proposed in [**?**] is the stage at which the perturbation occurs. As we do not aim at considering low-level details, we consider the logical operation at which an error occurs as opposed to an hardware instruction or a floating point operation. In our case, we thus refer to an instruction in Algorithm 1 such as the matrix-vector product (step 3) or the application of the preconditioner (step 7). We will focus mainly on both these operations as they are the most time consuming and DMR may thus be expensive to apply. Nonetheless, we will eventually cover all CG steps in the concluding remarks of our study on detection mechanisms (see Section 4.2.4 and Figure 4.5 in particular).

Third, we consider soft errors occurring as bit-flips on the operands of the considered step. Note that this choice introduces a bias as the considered steps may be composed of multiple atomic hardware operations. A bit-flip occurring on the input data (a and b in the above example) may thus be viewed as an early error while a bit-flip occurring on the output data (c) may be viewed as a late error. In our model, we thus do not consider the intermediates between those. Note once again that in the case of transient errors (focus of this paper), an early bit-flip on a (for instance changing $a = 2$ into $a = 8$) will affect the output data ($c = 10$) and the input data a is set back to its original value. Note that a single transient bit-flip injected on the input data is likely to induce multiple bit-flips on the output data.

We now provide a brief overview on bit-flip arithmetic. If a bit-flip occurs on an operand of value d, its value becomes $d + \delta d$ with a relative perturbation that depends both on the index ℓ of the affected bit and on the original value b_ℓ of this ℓ^{th} bit in the definition of d in (2.2). The possible relative perturbation $|\delta d|/|d|$ and associated bounds are summarized in Table 2.1. Note that the largest relative perturbations are

	$\lvert\delta d\rvert \,/\, \lvert d\rvert$		
original value	$\ell = 63$	$52 \leq \ell \leq 62$	$0 \leq \ell \leq 51$
$b_\ell = 0$	2	$2^{2^{\ell-52}} - 1$	$\dfrac{2^{\ell-52}}{1+m} \leq \dfrac{1}{2}$
$b_\ell = 1$	2	$\dfrac{1}{2} \leq 1 - 2^{-2^{\ell-52}} < 1$	$\dfrac{-2^{\ell-52}}{1+m} \leq \dfrac{1}{2}$

Table 2.1: Relative perturbation and associated bounds with a bit-flip on the ℓ^{th} bit depending on its original value b_ℓ.

obtained when the bit-flip affects a bit in the exponent that was originally equal to zero ($b_\ell = 0$).

3. Study of the sensitivity of PCG to soft errors.

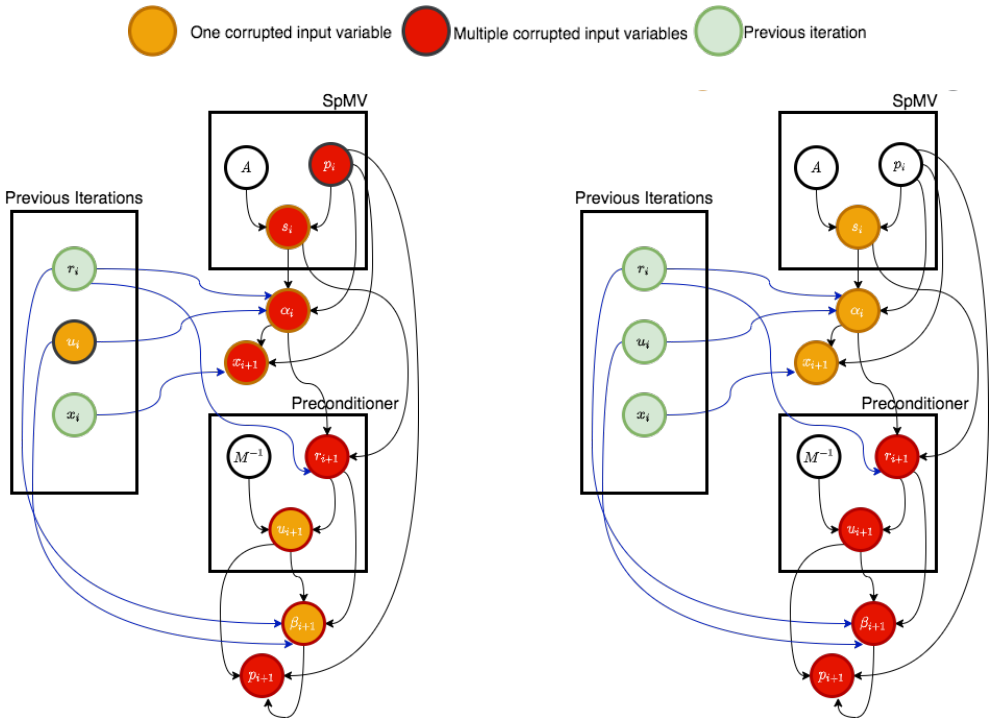
3.1. Propagation of bit-flips in PCG. As discussed above, we are interested in the possible impact of a transient bit-flip that might occur on data computed by the algorithm. Because the most computationally intensive numerical kernels are the matrix-vector product and the preconditioner application we only consider bit-flips in these two key steps of the algorithm. An additional motivation to focus on those two kernels is that the other calculations are mainly cheaper BLAS-1 operations that could be protected by DMR at an affordable extra computational cost. The propagation of the bit-flip in Algorithm 1 is as follows:

1. **Transient error in the matrix-vector calculation.** Assuming a transient error occurs at step 3 of Algorithm 1 during iteration i, after that step, the quantity s_i gets altered, while A and p_i are not. It implies that the current iterate (x_{i+1}) is computed using a corrupted scalar (α_i) and a non-corrupted vector (p_i) whereas the computed residual (r_{i+1}) is computed using both a corrupted scalar (α_i) and vector (s_i).
2. **Transient error in the preconditioner application.** Assuming a transient error at step 7 of Algorithm 1 during iteration i, the next updated iterate (x_{i+2}) in iteration $i+1$ is computed using a corrupted scalar (α_{i+1}) and a consistently corrupted vector (p_{i+1}). Similarly, the iteratively computed residual (r_{i+2}) is updated using the corrupted scalar (α_{i+1}) and corrupted vector (s_{i+1}).

The propagation of transient errors located in the matrix-vector product and in the preconditioner application have therefore a different impact on the quantities computed by CG, possibly introducing different effects on its numerical behavior.

Figure 3.1 shows the data flow in the main PCG loop as well as associated corrupted quantities when a transient soft error appears in the matrix-vector product (left) or preconditioning (right). In these graphs a vertex corresponds to a computing task and an edge to data dependencies between those. The orange color indicates that the task is performed with one corrupted input variable and red is used when more than one input variable is affected by a previous error.

3.2. Bit-flip injection protocol. A transient bit-flip in the matrix-vector product or the application of the preconditioner can happen anywhere in the computation kernel. For simplicity, however, we only consider early bit-flip injections in the input vector (p_i or r_{i+1}), which will be reset to its original value after the calculations with



(a) Transient error in matrix-vector product.

(b) Transient error in preconditioner application.

Figure 3.1: Propagation of transient errors in the PCG algorithm.

this vector are performed, or late bit-flip injections in the output vector (s_i or u_{i+1}). This approach can be seen as considering the two extreme cases of the effect that a single bit-flip can have: modifying the output vector directly implies that only one entry of s_i or u_{i+1} will be erroneous, but modifying the input vector p_i or r_{i+1} implies that multiple – and possibly every – entry of s_i or u_{i+1} will be erroneous.

We use the following protocol in order to create a database of PCG runs with and without the injection of a bit-flip:

1. Given a SPD $A \in \mathbb{R}^{n \times n}$, we generate a vector $x \in \mathbb{R}^n$ with random entries in $[-1, 1]$ and calculate the corresponding right-hand side b.
2. We solve the resulting linear system $Ax = b$ for x: once without the injection of a bit-flip and once with the injection of a bit-flip.
3. The injected bit-flip is randomly generated by varying the following parameters:
   - The iteration at which the error occurs: we use nine sample locations ranging from 10% up to 90% of the iterations required for PCG to converge without bit-flip.
   - The vector affected by the bit-flip: s_i or p_i to model a bit-flip in the matrix-vector product, and u_{i+1} or r_{i+1} to model a bit-flip in the preconditioner.
   - The entry of the vector that is affected by the bit-flip.
   - The bit in the 64-bit sequence of the IEEE-754 double precision representation that will be flipped.

ID	Name	Size	Norm	Cond	AMG iterations	Jacobi iterations	No preconditioner iterations
1	bodyy5	18,589	8.04e3	7.87e3	16 (2)	443 (52)	752 (112)
3	bundle1	10,581	6.43e12	1.00e3	16 (2)	58 (8)	242 (2)
5	crystm02	13,965	1.76e-12	2.50e2	10 (1)	58 (8)	146 (24)
7	crystm03	24,696	9.79e-13	2.64e2	10 (1)	58 (8)	151 (25)
9	Dubcova1	16,129	4.80	9.97e2	39 (5)	174 (23)	170 (28)
11	fv1	9,604	4.51	8.81	5 (1)	32 (4)	33 (6)
13	fv3	9,801	4.00	2.03e3	6 (1)	223 (30)	229 (38)
15	jnlbrng1	40,000	1.84e1	1.83e2	6 (1)	127 (16)	140 (22)
17	Kuu	7,102	5.41e1	1.58e4	110 (14)	581 (74)	712 (111)
19	minsurfo	40,806	8.10	81.11	6 (1)	91 (11)	100 (16)
21	obstclae	40,000	8.20	4.10e1	6 (1)	68 (9)	70 (11)
23	torsion1	40,000	8.20	4.10e1	6 (1)	68 (9)	70 (11)
25	wathen100	30,401	3.70e2	5.82e3	18 (2)	50 (7)	333 (54)
27	wathen120	36,441	3.69e2	2.58e3	18 (2)	50 (7)	379 (57)

Table 3.1: Some numerical properties of the matrices used in the experiments. The matrices with even IDs $2n$ correspond to the normalized versions of the matrices with IDs $2n-1$. We also list the average number of iterations (and standard deviations in parantheses) required by the linear systems in order to converge to threshold $\varepsilon_2 = 10^{-10}$ when no fault occurs (see (3.1)).

4. For the given matrix A, this process will be repeated multiple times – each time with a new random vector x, corresponding right-hand side b and new random bit-flip.

A sketch of this protocol is given in Figure 3.2.

The matrices we used to generate our data are listed in Table 3.1. In order to see the effect of the norm of A, we also consider the versions of these matrices scaled by $\|A\|_2$.The reason for this scaling will become clear in Section 4.1.1, where this norm will play a role in one of our detection criteria. Furthermore, we will consider CG runs without a preconditioner, with a Jacobi preconditioner (JAC) [**?**] and with an algebraic multigrid preconditioner (AMG) [**?**, **?**], and use the stopping criterium

$$\frac{\|b - Ax_{i+1}\|_2}{\|b\|_2} \leq \varepsilon_{\ell=1,2} \tag{3.1}$$

with two threshold values: low accuracy with $\varepsilon_1 = 10^{-5}$ and high accuracy with $\varepsilon_2 = 10^{-10}$. Finally, we generate two datasets: one with early bit-flips (in p_i or r_{i+1}), and one with late bit-flips (in s_i or u_{i+1}). For each of these, per combination of matrix, preconditioner, and ε_ℓ, we performed 100,000 runs without a bit-flip, and the 100,000 corresponding runs with a bit-flip.

3.3. Numerical experiments. We now study the sensitivity of the PCG algorithm with respect to soft error locations in time and space. As bit-flip injections impact the convergence behavior of PCG, the stopping criterion may be reached in a different number of iterations or even be prevented. We thus consider that a faulty execution converges if it achieves the same stopping criterion as the non-faulty execution with some authorized delay in term of number of iterations. In this work, we allow for 50% extra iterations (with respect to the non-faulty execution) to decide whether or not the soft error has prevented PCG to converge, see also Figure 3.2. We mention that we varied this extra iteration bound and it did not significantly change

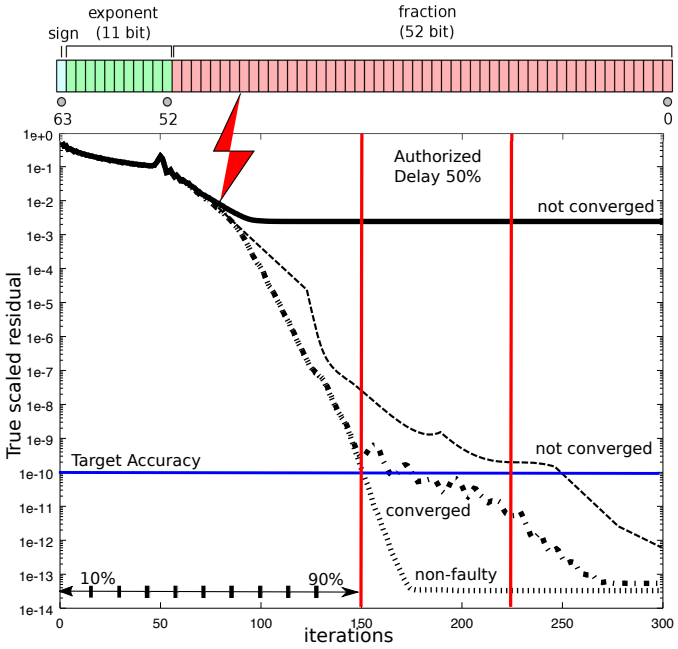


Figure 3.2: The error is injected early or late in vector at any entry, any bit and at any time in the iterative process for either the matrix-vector product or the preconditioner calculation.

the statistical results as the number of non converging cases would remain roughly the same. Some of the bit-flips translate into NaN (Not a Number). Such a situation is considered as a non-converged run in the proposed sensitivity analysis (Section 3) and as a detected fault in the study of detection mechanisms (Section 4). Finally, because we did not observe any significant differences between the results for early or late bit-flips, we only show the results for the experimetns with early bit-flips in this section. The results for late bit-flips can be found in [**?**, Appendix A].

3.3.1. Effect of the bit-flip value. Figure 3.3 shows the percentage of successfully converged executions of PCG when a soft error is injected in the matrix-vector product as a function of the index of the flipped bit in the 64-bit sequence of a double precision floating point number. We report both results on low ($\varepsilon = 10^{-5}$ in (3.1), top rows of the figure) and high ($\varepsilon = 10^{-10}$, bottom rows) target accuracy. An extra distinction is made in the rows between the experiments based on the original value of bit, *i.e.*, zero or one, and refer to them as "$0 \to 1$" or "$1 \to 0$" respectively. In the column we differentiate between different preconditioners: an AMG precondtioner, a Jacobi preconditioner, and no precondtioner, *i.e.*, $M = I$.

As could have been expected from the bound on the relative perturbation size reported in Table 2.1, one can notice that flipping a bit in the exponent from zero to one damages convergence more than a bit moving from one to zero. This trend is particularly visible in the first two rows of the graphs, for the less stringent stopping criterion. In the rest of our experiments we will, however, not make this "$0 \to 1$" or "$1 \to 0$" distiction.

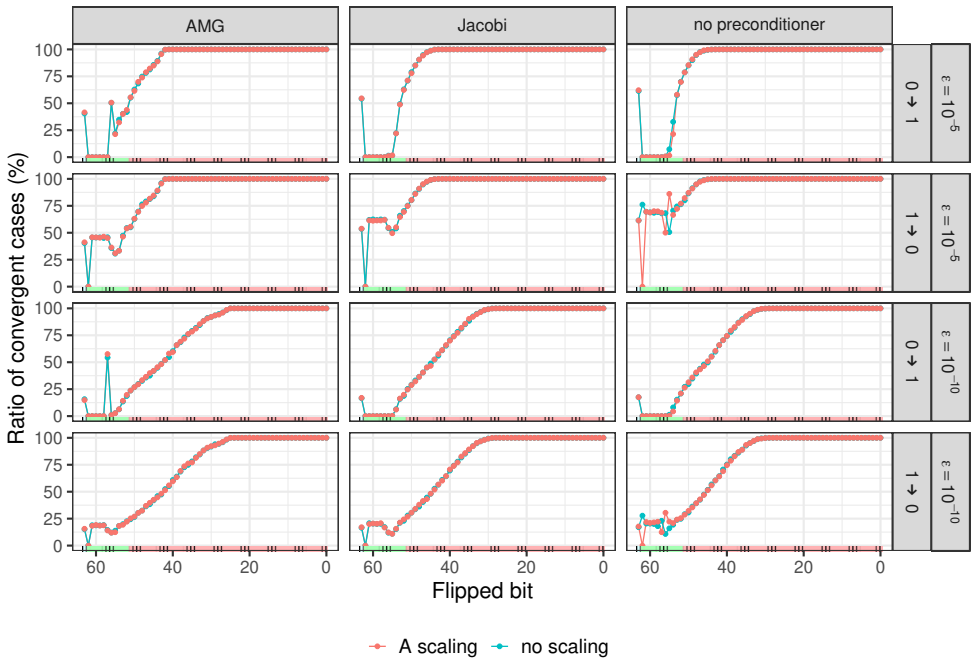


Figure 3.3: Comparison of the impact on convergence of the bit-flips at originally zero or one bits. The 64-bit indices of the IEEE 754 floating point numbers are displayed between each graph; from left to right, the sign (blue), exponent (green) and mantissa (red) bits are represented, see also Figure 2.1.

3.3.2. Soft errors in the matrix-vector product. Figure 3.4 shows the percentage of successfully converged executions of PCG when a soft error is injected in the matrix-vector product as a function of the index of the flipped bit. The bits flipped in the exponent or in high order bits of the mantissa very often prevent CG to converge, especially when a high accuracy is targeted. We also observe that bit-flips in the low-order bits in the mantissa do not prevent CG to converge for any of the two threshold values ε_ℓ; of course the index of the bit from which no effect is observed is lower for large value of ε_ℓ.

We also depict the influence of the bit-flip injection time in Figure 3.5. Those results show that early errors have a larger impact than late ones; this behavior is somehow coherent with the results presented in [**?**, **?**, **?**] in the context of inexact Krylov solvers where the inexactness in the matrix-vector calculation can grow as the inverse of the residual norm.

3.3.3. Soft errors in the preconditioner application. A similar experimental study was conducted for assessing the impact of transient errors in the preconditioner application. The impact of the index of the flipped bit is reported in Figure 3.6, while the influence of the bit-flip injection time is reported in Figure 3.7. We recall that no preconditioner means $M = I$, and that in this context a bit-flip in the preconditioner is just a bit-flip in the vector $u_{i+1} = r_{i+1}$.

One interesting observation is the lower negative impact of soft errors in the

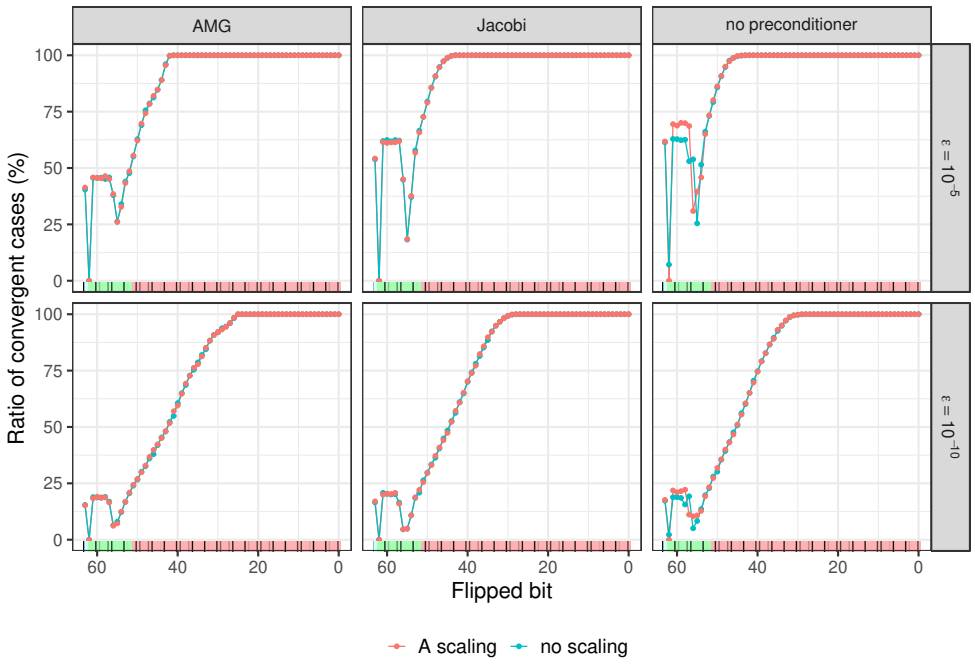


Figure 3.4: Impact of the index of the flipped bit in the matrix-vector product on PCG convergence success. The 64-bit indices of the IEEE 754 floating point numbers are displayed between each graph; from left to right, the sign (blue), exponent (green) and mantissa (red) bits are represented.

preconditioner application compared to errors in the matrix-vector product. For most of the experiments, a flipped bit in the mantissa does not prevent PCG to converge. To confirm this observation, we performed experiments using the identity as a preconditioner, which reduces to unpreconditioned CG when no bit-flip is injected. Comparing the subsequent results, reported in the last column of Figure 3.6 with those displayed in the last column of Figure 3.4, it can be seen that the two graphs exhibit very different behavior. The reason for this significant difference is that the propagation flows of a transient error occurring in the matrix-vector product and in the preconditioner application are very different (see Figure 3.1 and related discussion, above). A deeper theoretical analysis would certainly deserve to be undertaken to better understand this behavior; however, this analysis is out of the scope of the present study. Finally, applying PCG on matrices having a norm equal to one makes the numerical method generally slightly more robust to bit-flip in the exponent. A possible explanation is that in such cases, the values of the entries of the vectors are mostly lower than one, which corresponds to a large number of bits equal to one in their exponents [**?**]. This trend is more visible for bit-flips in the preconditioner application (see Figure 3.6) than in the matrix-vector product (see Figure 3.4).

3.3.4. Concluding remarks. A few supplementary comments on the experiments presented so far can be made. The first one is that PCG is rather robust and still converges in many cases even when bit-flips occur on exponent or large digits of the

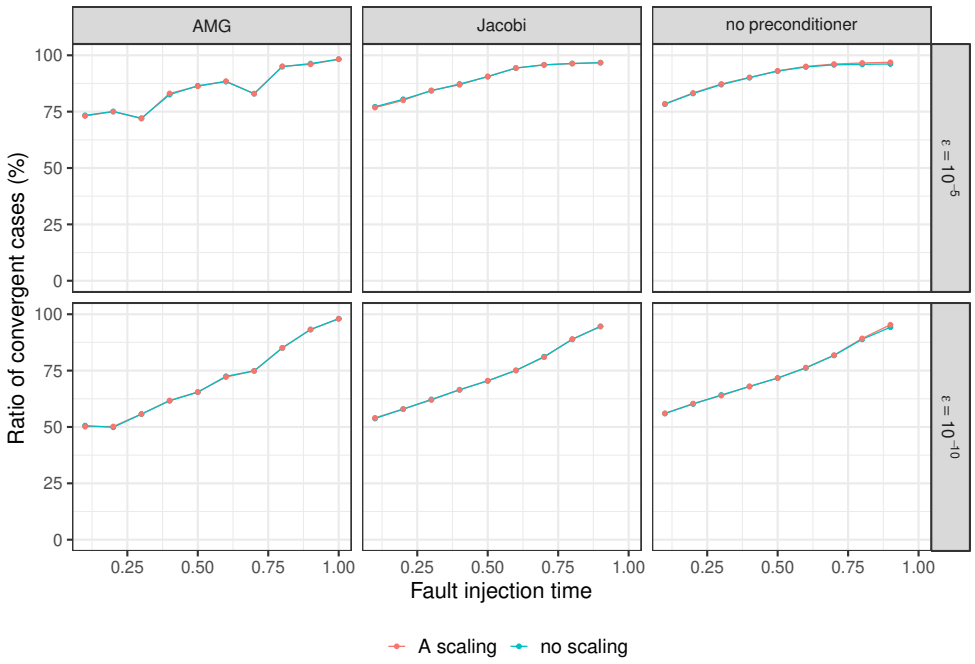


Figure 3.5: Impact of the bit-flip injection time (as a proportion of the number of iterations with respect to the non-faulty execution) in the matrix-vector product on PCG convergence success.

mantissa; obviously the less stringent the convergence threshold the more robust PCG is. A concurrent study performed in [**?**] shows similar results. Much more surprising is that PCG is significantly more sensitive to transient soft errors occurring in the matrix-vector product, than to those appearing in the preconditioner application. A second remark is that scaling the matrices with their 2-norm seems to have little to no effect on the ratio of convergent cases. Finally, in accordance with existing results on inexact Krylov, the earlier the soft error appears in the convergence process the larger the impact on the numerical behavior is and large errors close to the convergence might not prevent it to eventually converge. This is again something that was also observed in [**?**].

4. Numerical criteria for detecting soft errors in PCG.

4.1. Numerical criteria. The PCG method is a very sophisticated and elegant numerical scheme that has many properties induced by the symmetric positive definiteness of the matrix A, which, in particular, can be used to define a vectorial norm. Among those properties, we can recall the orthogonality between the residual r_i at each iteration or the A-orthogonality of the descent directions p_i. Unfortunately, those properties, as associated characteristic equalities, are no longer valid in finite precision calculation. On the other hand a lot of work has been devoted to study PCG in finite precision, see [**?**, **?**] and references therein. We consider some of the finite precision results and use them to define a potential soft error detection mechanism based on the residual gap.

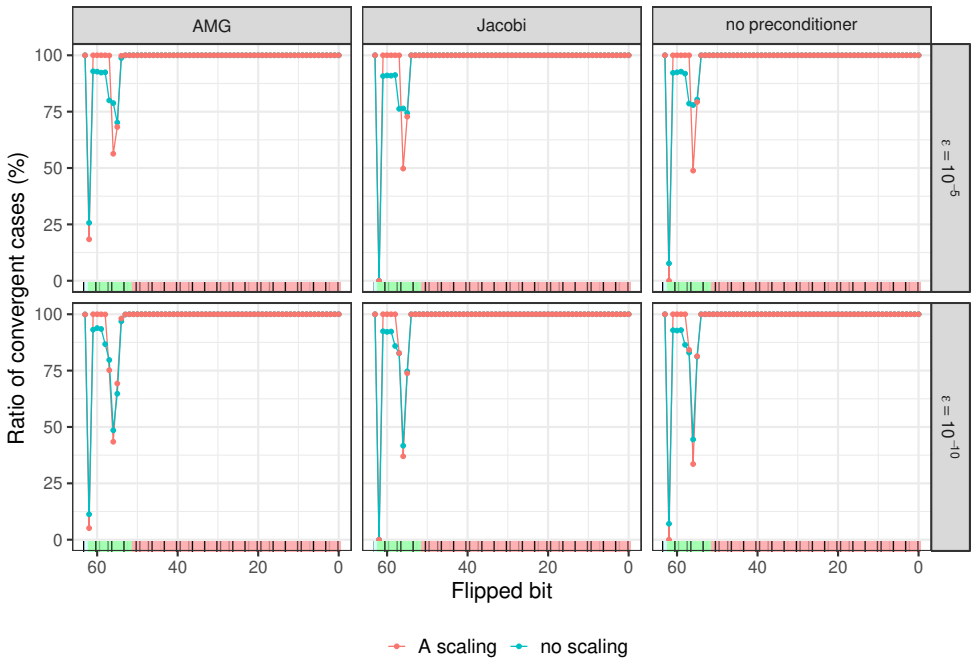


Figure 3.6: Impact of the index of the flipped bit in the preconditioner application on PCG convergence success. Recall that no preconditioner means $M = I$.

4.1.1. Residual gap-based detection. In exact arithmetic, the iteratively computed residual r_i is equal to the true residual defined by $b - Ax_i$ associated with the current iterate x_i; that is

$$r_i - (b - Ax_i) = 0. \tag{4.1}$$

In finite precision calculation, the computed quantities (denoted with a bar) differ from their exact mathematical values. A first consequence is that $f_i \equiv \bar{r}_i - (b - A\bar{x}_i)$ is no longer zero and defines the gap between the true and the iteratively computed residual referred to as the residual gap, which determines the maximal attainable accuracy. Using the rounding error analysis performed in [**?**, **?**], given an initial guess $\bar{x}_0$, the computed vectors satisfy

$$\bar{p}_0 = \bar{r}_0 = b - A\bar{x}_0 + f_0$$

and

$$\bar{x}_i = \bar{x}_{i-1} + \bar{\alpha}_{i-1}\bar{p}_{i-1} + \delta x_i, \tag{4.2}$$
$$\bar{r}_i = \bar{r}_{i-1} - \bar{\alpha}_{i-1}A\bar{p}_{i-1} + \delta r_i, \tag{4.3}$$
$$\bar{p}_i = \bar{r}_i - \bar{\beta}_i\bar{p}_{i-1} + \delta p_i, \tag{4.4}$$

where the individual δ-terms account for the local round-off errors associated with the different iterative updates.

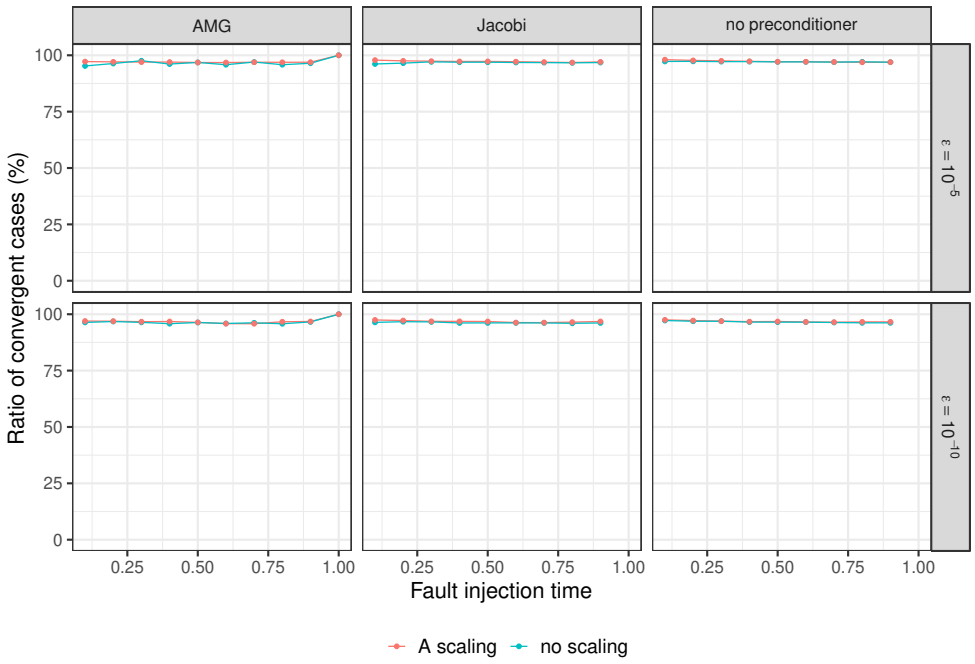


Figure 3.7: Impact of the bit-flip injection time in the preconditioner application on PCG convergence success. Recall that no preconditioner means $M = I$.

Using (4.2) and (4.3), a recurrence on the residual gap can be derived that accounts for the accumulation of the local round-off errors

$$\begin{aligned} f_i &= \bar{r}_i - (b - A\bar{x}_i) \\ &= \bar{r}_{i-1} - \bar{\alpha}_{i-1} A\bar{p}_{i-1} + \delta r_i - (b - A(\bar{x}_{i-1} + \bar{\alpha}_{i-1}\bar{p}_{i-1} + \delta x_i)) \\ &= f_{i-1} + \delta r_i + A\delta x_i \end{aligned}$$

so that we have

$$f_i = f_0 + A\sum_{\ell=1}^{i} \delta r_\ell + A\sum_{\ell=1}^{i} \delta x_\ell.$$

The norm of the residual gap between the true and the computed residuals has been intensively studied [**?**, **?**, **?**]. Assuming the standard model of floating point arithmetic with machine precision ϵ, see, e.g. [**?**, **?**, **?**], it is shown in [**?**] that the following upper bound holds for the norm of the residual gap

$$\|f_i\| \leq \epsilon \left(m||A|| \sum_{\ell=0}^{i} ||\bar{x}_\ell|| + \sum_{\ell=0}^{i} ||r_\ell|| \right), \tag{4.5}$$

where m corresponds to the maximal number of non-zero entries in the rows of the matrix A. This bound on the residual gap can be used to detect soft errors that have larger effects than the one predicted by the worse case scenario of the rounding error

analysis. It can be assessed on a periodic basis, for instance every other *CheckPeriod* iterations, as illustrated at lines 11 to 16 in Algorithm 3. Finally we note that fact that the bound (4.5) depends on $\|A\|$ is the motivation for us to include the scaled versions of all the matrices in our numerical experiments.

4.1.2. α-based detection. Among the numerous relationships that exist between the quantities computed by PCG, there is one that is possibly less prone to defection due to finite precision calculation because it is a bound and not a strict equality as for the orthogonality or A-orthogonality properties. It was already presented in the original paper on CG [**?**, Thm 5.5],

$$\forall i \quad \frac{1}{\lambda_{\max}} < \alpha_i < \frac{1}{\lambda_{\min}} \tag{4.6}$$

where $\lambda_{\max}$ and $\lambda_{\min}$ denote the largest and smallest eigenvalues of A, or the preconditioned matrix. From a practical view point, the calculation or the tight approximation of $\lambda_{\min}$ is generally expensive while $\lambda_{\max}$ can often be cheaply approximated using for instance randomized techniques [**?**]. Consequently, we only consider the lower bound to define our α-based detection mechanism, which is cheap to check at each iteration to possibly detect that an error occurred, as illustrated at lines 6 to 8 in Algorithm 3. The overall PCG algorithm equipped with both residual gap and α-based detection is given in Algorithm 3.

Algorithm 3 PCG enhanced with both residual gap-based and α-based detection

Require: **A**, b, x_0,**M**, λ_{max}, *CheckPeriod*.
1: $r_0 := b - Ax_0$; $u_0 = M^{-1}r_0$; $p_0 := r_0$
2: $f_0 = \epsilon(||r_0|| + m||A||||x_0||)$
3: **for** $i = 0, \ldots$ **do**
4: $s_i := Ap_i$
5: $\alpha_i := r_i^T u_i / s_i^T p_i$
6: **if** $\alpha_i < \frac{1}{\lambda_{max}}$ **then**
7: CreateDetectionAlert()
8: **end if**
9: $x_{i+1} := x_i + \alpha_i p_i$
10: $r_{i+1} := r_i - \alpha_i s_i$
11: $f_{i+1} = f_i + \epsilon(||r_{i+1}|| + m||A||||x_{i+1}||)$
12: **if** $mod(i, CheckPeriod) == 0$ **then**
13: **if** $||r_{i+1} - (b - Ax_{i+1})|| > f_{i+1}$ **then**
14: CreateDetectionAlert()
15: **end if**
16: **end if**
17: $u_{i+1} = M^{-1}r_{i+1}$
18: $\beta_{i+1} := r_{i+1}^T u_{i+1} / r_i^T u_i$
19: $p_{i+1} := u_{i+1} + \beta_{i+1} p_i$
20: **end for**

4.2. Numerical experiments. In this section, we aim at evaluating the robustness and genericity of the numerical detection mechanisms described in the previous sections (we do not consider the full DMR technique). We use the same data as in Section 3.3, and show the results for both early and late bit-flips. However, because we

Color	Term	Explanation
	true positive	A bit-flip occurred, it prevented convergence, and the detector raised an alert.
	false negative	A bit-flip occurred, it prevented convergence, but the detector did not raise an alert.
	true negative	No bit-flip occurred, and the detector did not raise an alert.
	false positive	No bit-flip occurred and the detector raised an alert.
	special negative	A bit-flip occurred, but it did not prevent convergence, and the detector did not raise an alert.
	special positive	A bit-flip occurred, but it did not prevent convergence, and the detector raised an alert.
	special critical	A bit-flip occurred, it prevented convergence, and it resulted in a NaN.

Table 4.1: Corresponding terms for color codes.

want to evaluate the different detection criteria, we will only take into account the experiments with a bit-flip and slightly broaden the classical terminology to characterize soft error detection mechanisms.

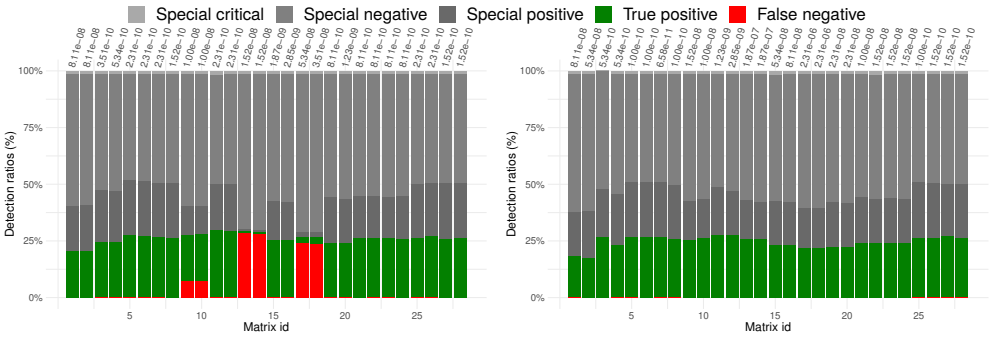
Table 4.1 discusses the color codes used in this section. The first four rows correspond to the classical taxonomy of the outcomes of any kind of decision methodology, that we extend with the last three rows for a finer analysis. These final rows correspond to situations where the detector raises an alert (or not) after a soft error occurred that did not prevent PCG from converging, or if the bit-flip translated into NaN. If, in a more binary setting, one only wishes to detect bit-flips that prevent convergence, the special positive, negative and critical cases could for example be interpreted as false positives, true negatives and true positives respectively. Depending on the goal of the detection, other interpretations are possible.

4.2.1. Checksum-based detection. Although the well-known checksum technique can be applied for protecting both the matrix-vector product and the preconditioner application, we focus on its capabilities in the case of bit-flip in the matrix-vector calculation. As indicated in Section 2.3, a threshold τ has to be chosen that should comply with two conflicting constraints: be large enough to reduce the false positives and be small enough to limit the number of false negatives. We do this independently for each combination of matrix, ε_ℓ and preconditioner by minimizing the following cost function:

$$\text{cost}(\tau) = \omega \frac{n_1}{n_1 + n_2} \text{FP}(\tau) + (1 - \omega) \frac{n_2}{n_1 + n_2} \text{FN}(\tau). \tag{4.7}$$

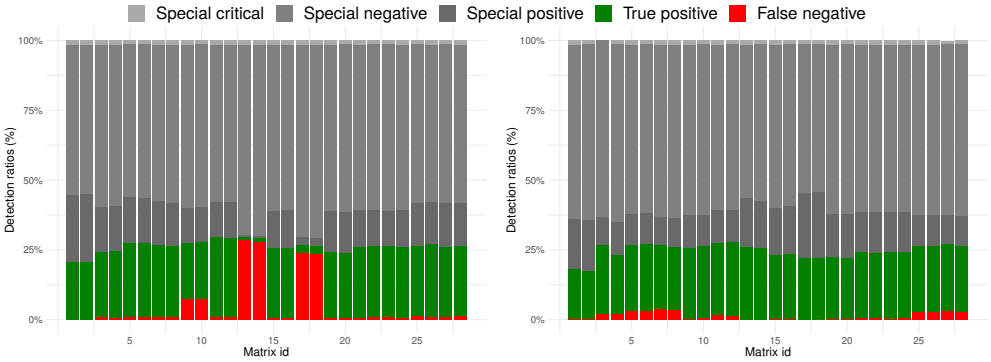
Here, n_1 and n_2 are the number of PCG runs for the given matrix, ε_ℓ and preconditioner without and with a bit-flip, $\text{FP}(\tau)$ and $\text{FN}(\tau)$ are the number of false positives and false negatives for the given value of τ, and $\omega \in [0, 1]$ is used to balance the two terms.

In Figure 4.1, we report on experiments where the checksum threshold parameter has been optimally tuned for each individual matrix, value of ε_ℓ, and preconditioner in order to minimize the cost function. The abscissa corresponds to the matrix index as defined in Table 3.1 and the values on top of the bars are the individual threshold values τ. In Figure 4.2 we report the same results, but here we used a non-optimal value



(a) Early bit-flips. (b) Late bit-flips.

Figure 4.1: Outcome of the checksum-based detection for the faulty runs when using the optimal thresholds τ (top of the bars) for early (in p_i) and late (in s_i) bit-flips, considering only the experiments with a Jacobi preconditioner and $\varepsilon =$ 1e-10 ($\omega = 0.5$).



(a) Early bit-flips ($\tau =$ 1.20e-7). (b) Late bit-flips ($\tau =$ 1.49e-7).

Figure 4.2: Outcome of the checksum-based detection for the faulty runs when using the non-optimal thresholds τ for early (in p_i) and late (in s_i) bit-flips, considering only the experiments with a Jacobi preconditioner and $\varepsilon =$ 1e-10 ($\omega = 0.5$).

for τ: the mean value of the optimal values of τ for all experiments with the same ε_ℓ. The former figure, on the one hand, illustrates the potential of the checksum critirion, but the optimization procedure it requires may often be prohibitive in practice. The latter figure, on the other hand, represents a more realistic usage of the checksum.

It should be noted that this average non-optimal value is by no means a terrible choice for τ. The difficulty lies in finding optimal values for every matrix (or a set of matrices). Furthermore, this value can vary greatly if we change ε or the preconditioner. We refer to [**?**, Appendix B] for more information on this. So while the checksum criterium is very generic and can be computed easily, it does require a careful tuning of the threshold parameter τ.

4.2.2. Residual gap-based detection. In this section we investigate the robustness of the soft error detection mechanism based on the residual gap. As discussed in Section 1 and illustrated in Figure 3.1, one could expect that soft errors in the matrix-vector product will very likely create a larger residual gap than predicted by the theoretical bound that only accounts for the worse case induced by round-off. The errors in the preconditioner calculation generate corrupted quantities that similarly affect the computed residual and the current iterate, so that the corresponding error mostly vanishes in the residual gap. Consequently, we first consider the experiments where the bit-flips are injected in the matrix-vector product. Furthermore, because we want to design criteria able to detect soft errors that prevent PCG to converge, we only consider non converging runs in our analysis below.

It would also not make sense to check the residual gap at each iteration, as the required matrix-vector product would be as costly as DMR to protect the matrix-vector calculation. We therefore only check this criterion periodically (every other $CheckPeriod = 10$ iterations in our experiments, as in Algorithm 3) and when exiting PCG (after convergence or reaching the maximal number of iterations). We also compare the behaviour of the residual gap detection with that of the checksum criterion for the detection of soft errors in the matrix-vector product (with optimal and not optimal τ). This is shown in Figure 4.3, where we once again present the proportion of undetected critical soft errors. The checksum strategy with an optimal threshold may be very efficient for late bit-flips, but it performs poorly for early bit-flips. Additionally, as discussed above, the optimization procedure may be prohibitive in practice, and when the checksum criterion is used with non-optimal parameter τ the detection is significantly less efficient, even for late bit-flips. The residual gap criterion, however, performs very well. This confirms the robustness of the residual gap criterion combined with periodic and final checks as a practical detection, since it achieves a very high rate of detection while not requiring any optimization procedure.

4.2.3. α-based detection. In this section we study the robustness and reliability of the α-based criterion (given by Equation (4.6)) to detect possible soft errors in the preconditioner application as it proved to be not effective to detect errors in the matrix-vector calculation. We display the ratio of undetected soft errors as a function of the bit-flip index as well as the ratio for the residual gap criterion in Figure 4.4. Both methods only seem to miss errors caused by bit-flips in the low order bits of the exponent, but the α-based detection criterion appears to be much more robust than the residual gap-based detection. This is to be expected, since due to the consistent propagation of the error in the iterations and residual updates, the residual gap criterion is going to be less effective when it comes to detecting errors in the preconditioner calculation.

4.2.4. Concluding remarks. In this section, we have assessed three numerical criteria to detect soft errors: the classical checksum mechanism, a bound on the norm of the residual gap and a bound on the α value. The numerical experiments have revealed the difficulty to define the threshold associated with the checksum in finite precision calculation. In that context, the rounding error analysis of the residual gap measurement provides a robust criterion. This criterion is particularly effective to detect soft errors in the matrix-vector product. Because the residual gap deviation bound is based on solid theoretical results, no false-positives can exist in non-faulty executions. Finally, the criterion based on α allows the detection of most of the errors in the preconditioner application. The combination of these last two criteria enables us to equip the PCG algorithm with numerical techniques to detect errors that do

not suffer from false-positives; consequently they are robust and reliable. A possible drawback of the α criterion is that it requires knowledge of the largest eigenvalue; we note that some numerically scalable multi-level preconditioning techniques provide this information (see [**?**] and reference therein). Finally, we remark that scaling the matrices with their 2-norm doesn't seem to have a large effect on either the residual gap-based detection – which motiviated this scaling – or the α-based detection.

One could naturally wonder whether these two detection mechanisms, which are robust criteria for detecting soft errors in the matrix-vector product and preconditioner application, would also be a robust criterion to detect errors occurring in the other steps of the PCG algorithm. In that respect, we performed an additional extensive set of experiments by injecting bit-flips in all steps of Algorithm 1. In Figure 4.5 we report the outcome of these experiments based on the step in which the errors were injected for the residual gap criterion, the α-based criterion, and the combination of both criteria as expressed in Algorithm 3. Several comments can be made. First, regarding the sensitivity of bit-flips occurring in step 10; altering an entry of the descent direction rarely prevents PCG from converging. Although possibly surprising, this is consistent with the observation made in Section 3.3.3, that is, the weak sensitivity of PCG to soft-errors in the calculation of the preconditioned residual. Second, the ability of the residual gap detection to catch a soft error occurring in the computation of the iterate update (step 5) is remarkable.

5. Conclusion and perspectives. In this paper we have experimentally investigated the robustness of PCG to transient soft-errors in its (usually) most time consuming kernels: the preconditioner and matrix-vector product. As could have been expected, we observed that soft errors affecting the exponent have a more detrimental impact on the convergence of PCG than low order bits in the mantissa. Surprisingly, we noticed that PCG was more robust to soft errors in the preconditioner that in the matrix-vector calculation. Based on these observations we proposed numerical criteria that aim at detecting soft errors that prevent PCG to converge. In particular, we illustrated that the classical checksum approach, based on equalities in exact arithmetic, may lack robustness in practice where a priori tuning procedure may be prohibitive. Alternatively, criteria based on the residual gap and on the range of validity of the values of α, which do not require any optimization procedure, allow for the definition of robust mechanisms. Future works will consist in using those criteria to design a self-correcting (or *self-stabilizing* using the terminology of [**?**]) PCG algorithm and in investigating similar approaches for modern variants of PCG such as pipelined PCG [**?**] as well as extending our work to non-symmetric Krylov subspace solvers.

Acknowledgments. This work has been funded by the EXA2CT European Project on Exascale Algorithms and Advanced Computational Techniques, which receives funding from the EU's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 610741. Experiments presented in this paper were carried out using the PlaFRIM experimental testbed, supported by Inria, CNRS (LABRI and IMB), Université de Bordeaux, Bordeaux INP and Conseil Régional d'Aquitaine (see https://www.plafrim.fr/). Siegfried Cools acknowledges funding by the Research Foundation Flanders (FWO) under grand number 12H4617N.

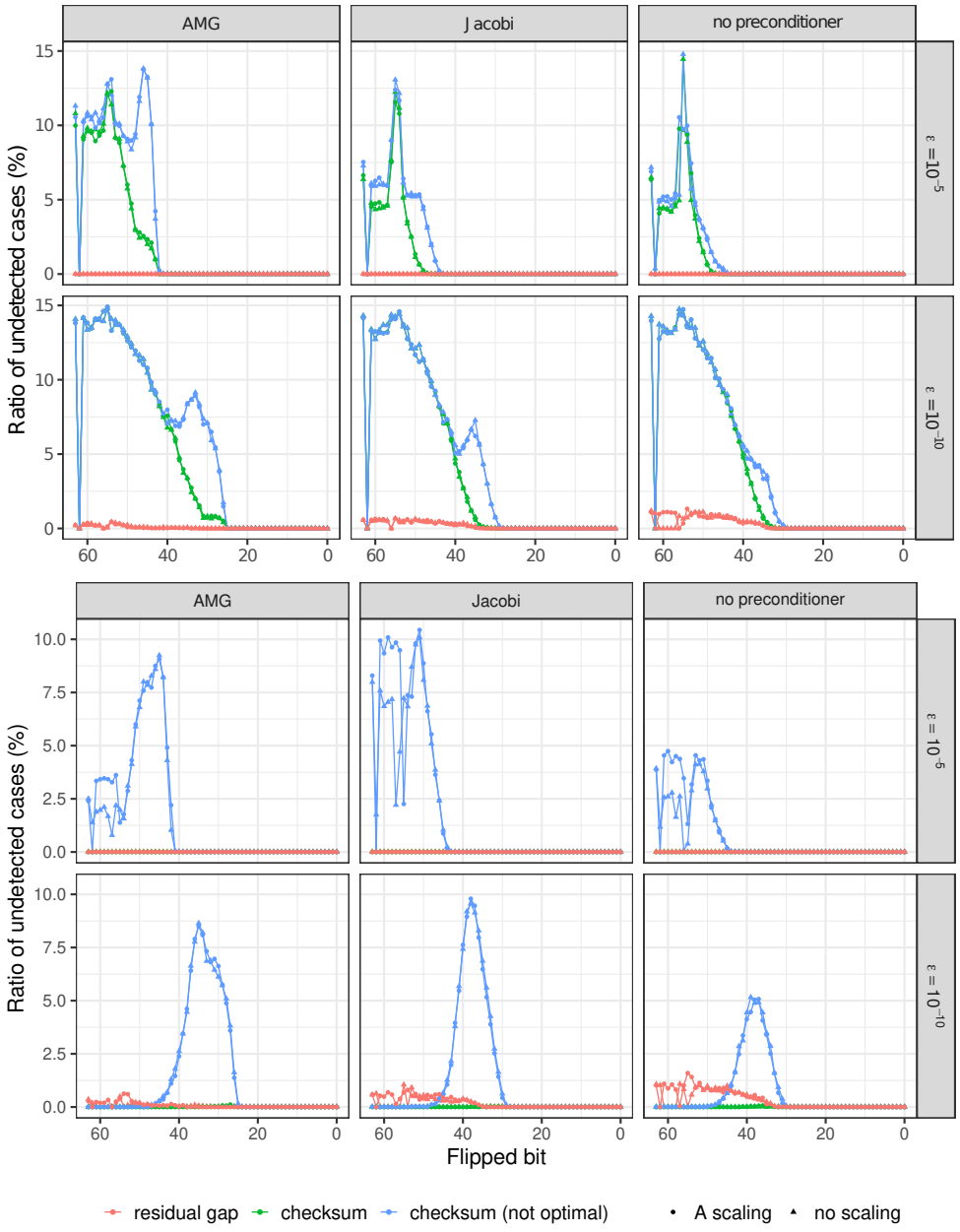


Figure 4.3: Detection performance of gap deviation and checksum-based methodologies for soft errors in the matrix-vector calculation. (top) Early bit-flips (in p_i). (bottom) Late bit-flips (in s_i).

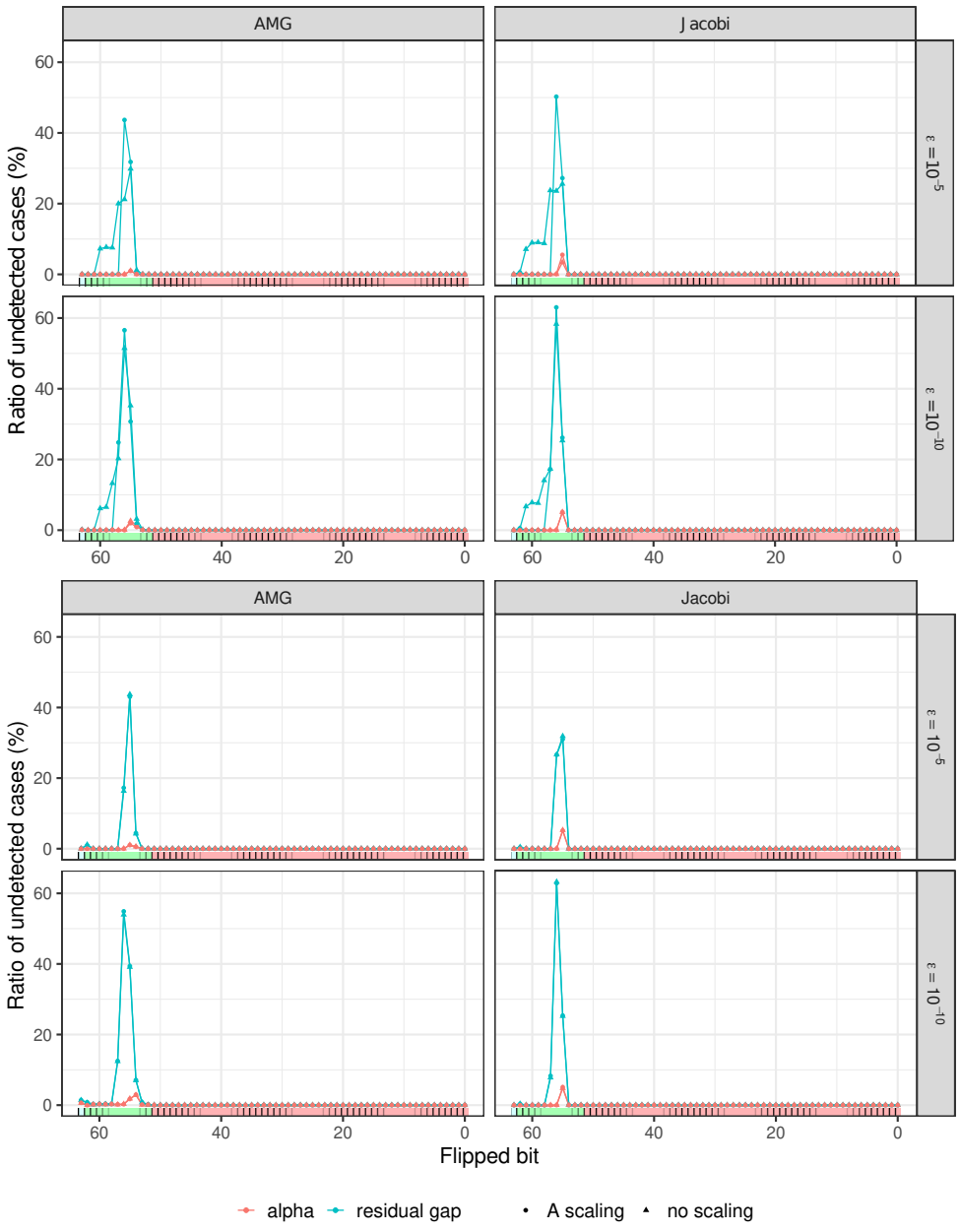


Figure 4.4: Detection performance of the residual gap and α-based methodologies for soft errors in the preconditioner calculation. (top) Early bit-flips (r_{i+1}). (bottom) Late bit-flips (u_{i+1}).

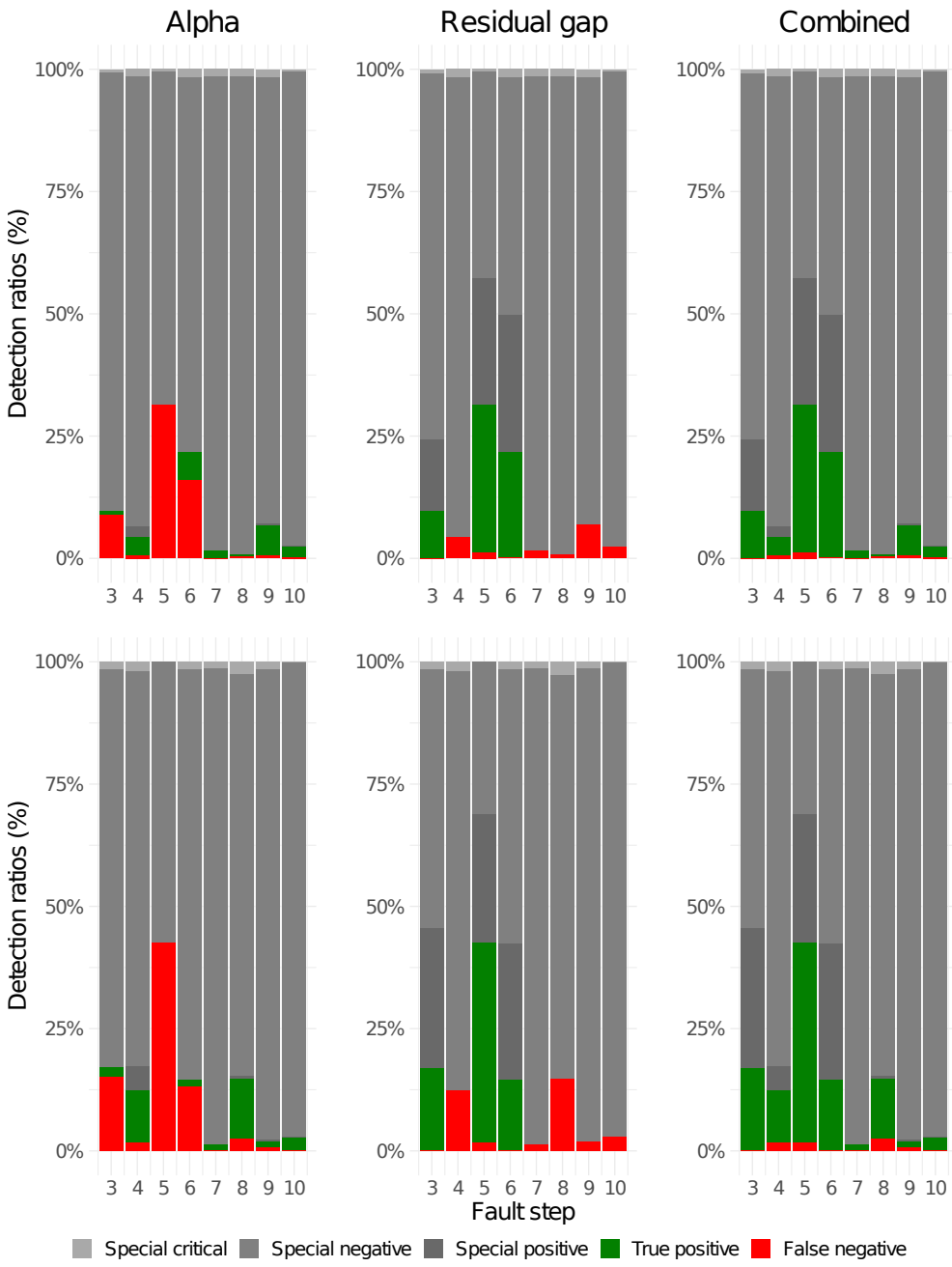


Figure 4.5: Comparison of the detection success of the alpha and residual gap based methodologies, and their combination for bit-flips in all possible steps of PCG. The set PCG runs used for this experiment is different from the other experiments, but was created using the same methodology described in Section 3.2. The only difference is the possible locations for the bit-flip. (top) Early bit-flips. (bottom) Late bit-flips.