

# Lightweight and Accurate Silent Data Corruption Detection in Ordinary Differential Equation Solvers

Pierre-Louis Guhur,<sup>1,2</sup> Hong Zhang,<sup>1</sup> Tom Peterka,<sup>1</sup> Emil Constantinescu,<sup>1</sup> and Franck Cappello<sup>1</sup>

<sup>1</sup> Argonne National Laboratory (USA)  
{hongzh, emconsta, tpeterka, cappello}@mcs.anl.gov

<sup>2</sup> ENS de Cachan (France)  
pierre-louis.guhur@ens-cachan.fr

**Abstract** Silent data corruptions (SDCs) are errors that corrupt the system or falsify results while remaining unnoticed by firmware or operating systems. In numerical integration solvers, SDCs that impact the accuracy of the solver are considered significant. Detecting SDCs in high-performance computing is necessary because results need to be trustworthy and the increase of the number and complexity of components in emerging large-scale architectures makes SDCs more likely to occur. Until recently, SDC detection methods consisted in replicating the processes of the execution or in using checksums (for example algorithm-based fault tolerance). Recently, new detection methods have been proposed relying on mathematical properties of numerical kernels or performing data analysis of the results modified by the application. None of those methods, however, provide a lightweight solution guaranteeing that all significant SDCs are detected. We propose a new method called *Hot Rod* as a solution to this problem. It checks and potentially corrects the data produced by numerical integration solvers. Our theoretical model shows that all significant SDCs can be detected. We present two detectors and conduct experiments on streamline integration from the WRF meteorology application. Compared with the algorithmic detection methods, the accuracy of our first detector is increased by 52% with a similar false detection rate. The second detector has a false detection rate one order of magnitude lower than these detection methods while improving the detection accuracy by 23%. The computational overhead is lower than 5% in both cases. The model has been developed for an explicit Runge-Kutta method, although it can be generalized to other solvers.

**Keywords:** resilience, fault tolerance, Runge-Kutta, numerical integration solvers, HPC, SDC

## 1 Introduction

Ensuring trustworthy results has always been a critical challenge for scientists. In numerical simulations, results can be impaired by silent data corruptions

(SDCs). Because of an ever increasing number of processes, exascale reports [18] project an increase in the SDC rate in future systems. The origins of SDC are diverse. Examples of SDC sources are electromagnetic interferences [15], ionizing radiation [1], and aging of hardware components.

Replication [10] can detect SDCs by duplicating the same program (or other versions in  $n$ -version programming [7]) and comparing their results. In a deterministic program, all duplications must provide exactly the same result; otherwise the results are considered corrupted. The protection of linear algebra results in algorithm-based fault tolerance (ABFT) [12] and the error-correction code memory (ECC) [9] are both based on checksums: ABFT computes and performs detection inside the software, and ECC memory is inside the hardware. All these methods are generic (although ABFT is limited to certain numerical kernels), and only a few percents of SDCs are undetected. However, these methods may be too computationally expensive (replication), or they do not protect each component. ABFT covers only the data used in the kernel and not the other data used by the application. ECC protects only memory, caches, and registers; it usually does not protect the CPU control logic or its functional units.

In the context of iterative, time-stepping methods, new detection techniques compare the results of the numerical method with results produced by a surrogate function. Because previous steps of the numerical method have already been validated, the surrogate function can use these values as trusted references to compute its own results for the current step. In the adaptive impact-driven (AID) detector [8], the surrogate function computes value predictions for the current step by extrapolation from several past steps of the numerical method. If the difference between the numerical method results and the surrogate function predictions is outside a certain confidence interval, an SDC is reported. AID uses different extrapolation methods from order 0 to 2 and selects dynamically the one that minimizes the prediction error. The confidence interval is built from the acceptable bound (given by the user) upon which SDCs are considered as impacting the results, the number of false positives, and the maximum error of extrapolation. Following a different direction, Benson et al. [3] propose a more complex surrogate function by computing an error estimate. As with the AID detector, the estimate is compared to a predicted value. If the estimate for the current step is not similar to previous estimates, an SDC is reported. Their detector is called *BSS14*. While the two approaches are different, they both use extrapolation and thus rely on the smoothness property of the data set (AID) or of the estimate (BSS14) to perform accurate detection. As shown in Sect. 4, they do not guarantee that all SDCs impacting the accuracy of the iterative methods are detected, in particular when the data set (AID) or the estimate (BSS14) presents stiff variations.

Our objective is to design and develop a new SDC detection technique that presents a high detection accuracy (also called recall or true positive rate, TPR) and a low false detection rate (also called false positive rate, FPR), and does not rely on extrapolation. *We mathematically show that all significant SDCs are detected.* In the context of numerical integration solvers, a solver is chosen because

its approximation error is acceptable with respect to the required accuracy of the results. We consider that an SDC is significant when the introduced error is bigger than the approximation error of the solver. We built two new detectors relying on mathematical properties of the ordinary differential equation (ODE) integration method. Our detection technique compares two estimates of this approximation error. We chose estimates that are similar if and only if no significant SDC occurs. A confidence interval on the similarity, established from a simple machine learning algorithm, controls the SDC detection. If an SDC is detected, the correction is done by recomputing the step. The two detectors present different tradeoffs. One has a high accuracy (we call it *Hot Rod HR*, for resilient ODE high recall) and small false detection rate. The other has a false detection rate lower than 1% but also a lower accuracy (we call it *Hot Rod LFP*, for low false positive). We designed the two detectors for Cash-Karp's method [5], a fourth-order Runge-Kutta method with a fixed-step size. However, our technique can be applied to other ODE integration methods as discussed in Sect. 5.

Because all significant SDCs are detected, our detectors improve the trustworthiness of the results while avoiding wasting of resources to recover from insignificant SDCs. We performed experiments on a streamline integrator used for visualizing of WRF meteorology application results [17].

Section 2 explains background. In Sect. 3, our method for detecting SDC is detailed, and proof is given that all significant SDC are detected. In Sect. 4, our SDC detectors are tested in a meteorology application and compared with replication and the AID and BSS14 detectors.

## 2 Background

An ODE is a differential equation of one independent variable and its derivatives. Because numerical integration solvers are widely used, trust in their results is critical. An initial value problem can be formulated as

$$x'(t) = f(t, x(t)), \quad x(t_0) = x_0,$$

with  $t_0 \in \mathbb{R}, x_0 \in \mathbb{R}^m, x : \mathbb{R} \rightarrow \mathbb{R}^m$ , and  $f : \mathbb{R} \times \mathbb{R}^n \rightarrow \mathbb{R}^m$ ;  $f$  is  $L$ -Lipschitz continuous.

### 2.1 Runge-Kutta methods

For each  $n = 1, \dots, N$  with  $N$  the total number of steps, Runge-Kutta methods (RKMs) provide an approximation  $x_n$  of  $x(t_n)$ , where  $t_n = t_0 + nh$ ,  $h \in \mathbb{R}_+^*$  the step size, and  $x(t_n)$  the exact solution of the ODE at time  $t_n$ . An  $s$ -stage explicit RKM is defined by

$$\forall i \leq s, k_i = f \left( t_n + c_i h, x_n + h \sum_{j=1}^{i-1} a_{ij} k_j \right); \quad x_{n+1} = x_n + h \sum_{i=1}^s b_i k_i.$$

Here  $(k_i)_i$  are called the stage slopes and represent the most computationally expensive part of the method. The local truncation error (LTE) is the approximation error introduced at a step  $n + 1$ , and it can be defined by  $\text{LTE}_{n+1} = x_{n+1} - \tilde{x}(t_{n+1}, x_n)$ , with  $\tilde{x}(t, x_n)$  the exact solution of the problem  $\tilde{x}'(t, x_n) = f(t, \tilde{x}(t, x_n))$ ,  $\tilde{x}(t_n, x_n) = x_n$ . The global truncation error (GTE) is the absolute difference between the correct value  $\tilde{x}(t_n, x_0)$  and the approximated value  $x_n$ . An ODE integration method is said to have an order  $p$  if  $\text{LTE}_n = O(h^{p+1})$  and  $\text{GTE}_N = O(h^p)$ , where  $N$  is the last step.

In the following, we focus on a RKM called Cash-Karp's method [5], while generalization is discussed in Sect. 5. Cash-Karp's method has an order 4 and computes six stages. Although two more stages than the classical fourth-order Runge-Kutta method are required, Cash-Karp's method allows us to compute the embedded method that is used in BSS14, in our detectors, and in the adaptive integration scheme.

## 2.2 Embedded methods

LTE can be estimated with *embedded methods*. These methods compute two results  $x_n^p$  and  $x_n^q$  from two RKMs with orders  $p$  and  $q$  (in general  $|q - p| = 1$ ). The solution is propagated by one of these results, while its stages (and possibly extra stages) are reused to compute the other result in order to achieve a low overhead. In the case of Cash-Karp's method,  $p = 4$  and  $q = 5$ . If  $\text{LTE}_n^q$  has a higher order than does  $\text{LTE}_n^p$ , the difference between  $x_n^p$  and  $x_n^q$  estimates  $\text{LTE}_n^p$ :

$$\begin{aligned} \mathcal{E}_n &= x_n^p - x_n^q \\ &= x_n^p - \tilde{x}(t_n, x_{n-1}) - (\tilde{x}(t_n, x_{n-1}) - x_n^q), \\ &= \text{LTE}_n^p - \text{LTE}_n^q, \\ &= \text{LTE}_n^p + O(h^{q+1}). \end{aligned}$$

## 2.3 Radau's quadrature

Another way of estimating LTE is suggested by Stoller and Morrison [19] and extended by Ceschino and Kuntzmann [6]. Relying on Radau's quadrature and Taylor's expansion, Ceschino and Kuntzmann give an expression of the LTE of a method given its order  $p \leq 5$ . The estimate  $\mathcal{R}_n$ , called here *Radau's estimate*, does not require the computation of any extra stage, but it checkpoints previous stages and solutions. Therefore, it has a memory overhead, rather than a computational overhead as does the embedded method. Since  $\mathcal{E}_n$  is a sixth-order estimate, we use the following estimate  $\mathcal{R}_n$  presented by Butcher and Johnston [4]:

$$\begin{aligned} \mathcal{R}_n &= \frac{h}{10} [3f(t_{n-2}, x_{n-2}) + 6f(t_{n-1}, x_{n-1}) + f(t_n, x_n)] \\ &\quad - \frac{1}{30} [x_{n-3} + 18x_{n-2} - 9x_{n-1} - 10x_n] \\ &= \text{LTE}_n^p + O(h^{p+2}). \end{aligned}$$

### 3 Proposed Hot Rod method

Our method relies on a surrogate function  $\Delta_n$  that is the difference between two estimates:  $\Delta_n = \mathcal{A}_n - \mathcal{B}_n$ . For Cash-Karp's method, we use the embedded estimate  $\mathcal{A}_n = \mathcal{E}_n$  and Radau's estimate  $\mathcal{B}_n = \mathcal{R}_n$ . In the absence of SDC, the surrogate function becomes  $O(h^{p+2})$ :

$$\Delta_n = (\text{LTE}_n + O(h^{p+2})) - (\text{LTE}_n + O(h^{p+2})) = O(h^{p+2}).$$

In Hot Rod HR, the surrogate function is compared with a certain confidence interval centered over zero. When the surrogate function is outside the confidence interval, an SDC is reported. We show that all significant SDCs are detected. However, Hot Rod HR may have a false positive rate of a few percents. In Hot Rod LFP, we chose a larger confidence interval, and its false positive rate remains below 1 percent.

#### 3.1 First detector: Hot Rod HR

In regular cases, our surrogate function is one order higher than that of the LTE. In presence of an SDC,  $\Delta_n$  is outside the confidence interval, as shown in the following paragraph. Hence, SDCs whose introduced errors are even smaller than the LTE are expected to be detected. We show that all significant SDCs are detected by Hot Rod HR.

**Detection of significant SDCs** An SDC is detected when  $|\Delta_n^c| \geq \mathcal{C}_n$  with  $\mathcal{C}_n$  the half-length of the confidence interval at step  $n$ . It is all the more difficult to detect when  $\Delta_n^o = 0$ . We show that the minimum injected error  $\epsilon_{min}$  that can be detected is of the same order as that of the approximation error.

We study the case of a corrupted stage  $k_i$ ; the case of a corrupted result  $x_n$  itself is similar. Here,  $k_i^c = \epsilon - k_i^o$ , where  $^c$  (resp.  $^o$ ) denotes corrupted (resp. uncorrupted) data:

$$\Delta_n^c - \Delta_n^o = \mathcal{E}_n^c - \mathcal{E}_n^o - (\mathcal{R}_n^c - \mathcal{R}_n^o) = h\epsilon \left[ \hat{b}_i + b_i \left( \frac{1}{30} - \frac{3\delta_{i,1}}{10} \right) \right],$$

where  $\delta_{ij}$  is defined by  $\delta_{ij} = 1$  if  $i = j$ ; otherwise  $\delta_{ij} = 0$ , ( $b_i$ ) (resp. ( $\hat{b}_i$ )) are the coefficients of the order 4 (resp. 5) in Cash-Karp's method.

The minimum error  $\epsilon_{min}$  that we can detect corresponds to the case  $|\Delta_n^c - \Delta_n^o| = \mathcal{C}_n - 0$ . We note that  $B = \hat{b}_i + b_i \left( \frac{1}{30} - \frac{3\delta_{i,1}}{10} \right)$ . This leads to

$$\epsilon_{min} = \frac{\mathcal{C}_n}{hB} = O\left(\frac{\mathcal{C}_n}{h}\right).$$

When  $x_n$  is corrupted instead of a stage, one can derive that  $\epsilon_{min} = O(\mathcal{C}_n)$ . If  $\mathcal{C}_n$  has the same order as  $\Delta_n$ , then (1)  $\epsilon_{min} = O(h^{p+1})$  when an error is injected inside a stage and (2)  $\epsilon_{min} = O(h^{p+2})$  when an error is injected inside a result. In other words, the threshold of detection has the same order as (or better than) the LTE of Cash-Karp's method. This guarantees that all significant SDCs are detected.

**Confidence interval** Because  $\Delta_n = O(h^{p+2})$ , one can assume that  $\Delta_n$  acts as a random variable, with a zero-mean in the absence of SDC. Its standard deviation can be estimated from a training set  $T$  composed of  $N_s$  samples with the unbiased sample standard deviation

$$\sigma = \sqrt{\frac{1}{N_s - 1} \sum_{n=1}^{N_s} \Delta_n^2}. \quad (1)$$

Assuming that  $(\Delta_n)_n$  follows a normal distribution, the “three sigma rule” [14] suggests choosing  $\mathcal{C}_n = 3\sigma$ . Thus, we expect that 99.7% of uncorrupted  $(\Delta_n)_n$  fall within the confidence interval, or in other words a false positive rate of 0.3%. The normal distribution is a natural choice for modeling the repartition of training samples.

Because items from  $T$  are not labeled as trusted or untrusted samples, the evaluation of  $\sigma$  might be corrupted. It thus would jeopardize the confidence interval and thus the SDC detector. To improve reliability, we weighted each  $\Delta_n$  with its own value. Equation (1) becomes

$$\Sigma = \sum_{n=1}^{N_s} \exp(-\Delta_n^2); \quad \sigma = \sqrt{\frac{1}{(N_s - 1)\Sigma} \sum_{n=1}^{N_s} \exp(-\Delta_n^2) \Delta_n^2}.$$

**Adaptive control** The hypothesis of a normal distribution may be invalidated. We therefore developed a correction of the confidence interval based on false positives.

When an SDC is reported, the current step is recomputed. If the result has the same value, we can assume that it was a false positive and not an SDC. Because of the “three sigma rule,” the FPR is expected to be 0.3%. If the FPR is an order of magnitude higher, at 3%, for  $k$  times, the confidence interval is increased with a certain coefficient  $1 + \alpha$ .  $\mathcal{C}_n$  becomes  $\mathcal{C}_n = (1 + \alpha)^k \times 3\sigma$ , where  $\alpha$  fixes the rate of the adaptive control. Because  $(1 + \alpha)^k = 1 + \alpha k + O(\alpha^2)$ ,  $\alpha$  is taken as  $1/(\max(\text{FPR}) \times N)$ , where  $N$  is the number of steps in the application and  $\max(\text{FPR})$  is the maximum acceptable false positive rate. Because a false positive requires the recomputation of a noncorrupted step, we suggest setting  $\max(\text{FPR})$  at 5% to limit the computational overhead. In our experiments, we have  $N = 1000$ ; thus  $\alpha = 0.02$ .

Thanks to the adaptive control, the training set requires only a few steps. In our experiments, we have found that  $N_s = 5$  samples are sufficient to initialize the confidence interval.

### 3.2 Second detector: Hot Rod LFP

If the cost of a false positive is too high, Hot Rod HR is not suitable. Hence, we designed a second detector with a larger confidence interval. Nonetheless, all significant SDCs must still be detected.

This new confidence interval is defined by  $\mathfrak{C}_n = 10C_{99}(|\Delta| \in T)$ , with  $C_{99}$  the 99th percentile of the training set. The interval can be interpreted as a threshold that is an order of magnitude bigger than the surrogate functions in the training set. Because this threshold is higher than the previous one, this detector's recall is lower. Because the estimates are at order  $p = 4$  for Cash-Karp's method, the LTE at step  $n$  can be expressed as  $\text{LTE}_n = Ch^{p+1} + O(h^{p+2})$ . We show that the GTE at the last step  $N$  is still an order  $p$ , since it used to be without corruption. We assume the probability that an SDC occurs and is accepted as small enough to guarantee that at most only one SDC will be accepted. The worst case is when this SDC is accepted at the first step,  $n = 1$ , and when  $\mathfrak{C}_n = \Delta_n$ . Hence, the introduced error is  $\text{LTE}_1 = 10Ch^{p+1} + O(h^{p+2})$ . Because  $\text{GTE}_1 = \text{LTE}_1$ ,  $\text{GTE}_1 = 10Ch^{p+1} + O(h^{p+2})$ .

With  $\tilde{x}(t, x_n)$  the notation in Sect. 2.1,  $x(t) = \tilde{x}(t, x_0)$ , and one can write that the GTE at a step  $0 < n < N$  is

$$\begin{aligned} |\text{GTE}_{n+1}| &= |x(t_{n+1}) - \tilde{x}(t_{n+1}, x_n) + \tilde{x}(t_{n+1}, x_n) - x_{n+1}|, \\ &\leq |x(t_{n+1}) - \tilde{x}(t_{n+1}, x_n)| + |x_{n+1} - \tilde{x}(t_{n+1}, x_n)|. \end{aligned}$$

Because  $f$  is  $L$ -Lipschitz continuous, the Gronwall's inequality simplifies the first term to

$$|x(t_{n+1}) - \tilde{x}(t_{n+1}, x_n)| \leq |\tilde{x}(t_n, x_0) - \tilde{x}(t_n, x_{n-1})| e^{Lh} = |\text{GTE}_n| e^{Lh}.$$

The second term,  $|x_{n+1} - \tilde{x}(t_{n+1}, x_n)|$ , is the LTE at step  $n+1$  and so is evaluated at  $Ch^{p+1} + O(h^{p+2})$ . Denoting  $\gamma = e^{Lh}$ , we obtain

$$\frac{|\text{GTE}_{n+1}|}{\gamma^n} \leq \frac{|\text{GTE}_n|}{\gamma^{n-1}} + \frac{Ch^{p+1}}{\gamma^n} \leq \dots \leq |\text{GTE}_1| + Ch^{p+1} \sum_{i=1}^n \frac{1}{\gamma^i}.$$

Because  $\sum_{i=1}^N 1/\gamma^i = (\gamma^N - 1)/\gamma^N(\gamma - 1)$  and  $\gamma - 1 \geq Lh$ , noting  $\tau = Nh$ , we obtain

$$|\text{GTE}_{n+1}| \leq 10Ch^{p+1} + \frac{Ch^p}{L} (e^{L\tau} - 1) + O(h^{p+2}).$$

At the last step, we have verified that  $\text{GTE}_N = O(h^p)$ . The order of GTE is unchanged: the SDC is insignificant.

### 3.3 Algorithm

We presented two detectors and showed their efficiency. They differ in their tradeoffs: Hot Rod HR has a higher TPR, and Hot Rod LFP has a lower FPR. We saw that undetected SDCs have no impact on the accuracy of the ODE method. They require fixing the parameter  $\alpha$ , but simple indications are given. We can thus derive two scenarios. If an SDC is likely to happen (it could be the case when the processor is not protected from SDC by ECC memory or other protection system), then Hot Rod HR is employed. Otherwise, employing Hot Rod LFP allows us to detect all significant SDCs with fewer false positives. The schema is illustrated in Algorithm 1 for a given detector.

```

while learning do
  step  $\leftarrow$  simulation(prev. step) ;
   $\Delta \leftarrow |\mathcal{A}(step, prev.steps) - \mathcal{B}(step, prev.steps)|$  ;
  TraininigSet.push( $\Delta$ ) ;
end
while new step do
  step  $\leftarrow$  simulation(prev. step) ;
   $\Delta \leftarrow |\mathcal{A}(step, prev.steps) - \mathcal{B}(step, prev.steps)|$  ;
  if (Detector == Hot Rod HR and  $\Delta \leq C_n$ ) or (Detector == Hot Rod LFP
  and  $\Delta \leq \mathfrak{C}_n$ ) then
    report("no error") ;
    accept step ;
  end
  else
    step  $\leftarrow$  simulation(prev. step) ;
     $\Delta' \leftarrow |\mathcal{A}(step, prev.steps) - \mathcal{B}(step, prev.steps)|$  ;
    if  $\Delta' = \Delta$  then
      report("false positive") ;
      if FPR > 3% then
        k++ ;
      end
    end
    accept step ;
  end
end

```

**Algorithm 1:** Pseudocode for the execution of our detectors

## 4 Experiments and results

We have shown theoretically that all significant SDCs are detected with Hot Rod. In this section, we evaluate the SDC detectors with a meteorology application.

### 4.1 Environment

Experiments were computed on a machine with four Intel Xeon E5620 CPUs (each with 4 cores and 8 threads), 12 GB RAM, and one NVIDIA Kepler K40 GPU with 12 GB memory. It was programmed in C++11 using CUDA. The application is particle tracing for streamline flow visualization [17],[11], [16]. The solver integrates a velocity field to compute the streamline. It stops when the streamline goes outside the velocity field. Uncorrected streamlines can thus be shorter than they were supposed to be.

### 4.2 SDC injection methodology

An SDC can arise from many sources in hardware and software [2], [13], and these sources may change with new versions and generations of hardware and software. We do not attempt to evaluate exhaustively the coverage of our approach because of space limitations. SDCs are simulated by flipping bits in data items. SDCs

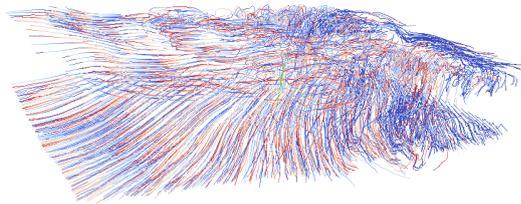


Figure 1. Streamlines computed by the application. The color gradient starts in red at seeds; 1,408 streamlines are computed.

affect one or several bits in the same data item, called respectively singlebit and multibit corruption. We experimented with both cases. In multibit corruption, we chose the number of bit-flips  $N_{flips}$  from a uniform distribution. Other distributions such as normal and beta distributions were tested with several different parameters, but the results were not significantly different from those reported below. Corruption can affect data items in any stage (or even directly in the result). The position of a bit-flip is drawn from a uniform distribution. In multibit corruption, we have forced the  $N_{flips}$  bit-flips to be applied on  $N_{flips}$  different positions. Some SDC have no impact on the results. In a third scenario, we inject only significant singlebit corruptions. We considered that an SDC is significant when the difference between the corrupted result and the safe result is higher than the mean LTE.

### 4.3 Benchmark

We compared our approach with similar methods presented in Sect. 1: replication, AID and BSS14 detectors. Those methods need to be parametrized. We compared results with a set of parameters and selected the parameters that provide the best results in our application. Using the same notation as in [8], we configure AID with  $\theta r = 1$ . Results were improved if the confidence interval is taken as  $(1 + \alpha)^k (\epsilon + \theta r)$  with  $\alpha = 0.2$  and  $k$  defined in Sect. 3.1. Concerning BSS14, five parameters should be set, but no indication is detailed in [3] about two of them. With the notation of [3], the considered values are  $\tau_j = 1e^{-5}$ ,  $\tau_v = 0.02$ ,  $I = 1.4$ ,  $\gamma = 0.95$ , and  $p = 10$ .

### 4.4 Results

Table 1 presents results from our benchmark. We did not compare each detector with a solver with no detector. We compared each detector with a perfect detector that returns the ground truth. For computational overhead, we divided the execution time of each detector with that of the perfect detector. Our detectors have a computational overhead lower than 5%, as do the BSS14 and AID detectors. It is 20 times less computationally expensive than replication. But

Table 1. Benchmark of our detectors Hot Rod (H.R.) LFP and HR, replication, AID and BSS14. Values in the column “IRE 95%” are the injected relative errors (IRE) that were detected 95% of the time.

Detector	TPR (%)			FPR (%)	IRE 95%	Overheads (%)	
	Singlebit	Multibit	Significant			Comp.	Memory
Replication	100.0	100.0	100.0	100.0	0.0	+100	+100
AID	14.3	43.2	86.7	1.6	$7e^{-6}$	+4.6	+50
BSS14	18.8	49.5	91.2	0.6	$4e^{-6}$	+3.7	+13
H.R. LFP	23.1	64.6	99.9	0.01	$7e^{-8}$	+3.8	+50
H.R. HR	28.6	69.6	99.9	1.2	$5e^{-9}$	+4.4	+50

unlike the AID detector, our detectors have to employ an embedded integration method that computes more stages than does another Runge-Kutta method of the same order.

Our detectors have a higher memory cost than does the BSS14 detector, but a smaller memory cost than does replication. For estimating memory overheads, we counted the number of stored vectors, such as solutions  $(x_n)_n$ , stage slopes  $(k_i)_i$  and estimates. Cash-Karp’s method requires computing and storing two additional stage slopes than does Runge-Kutta 4, but the same number as the other embedded fourth-order methods. Cash-Karp’s method requires storing 6  $(k_i)_i$  (among them  $f(x_{n-1})$ ), and  $x_n$ ;  $x_{n-1}$  is stored to allow a rollback in case of SDC detection; when  $f(x_{n-1})$  is employed in the Radau estimation,  $f(x_n)$  can be computed at the position (the result is employed at the next step if the step is accepted). Thus in total, 8 data elements are stored by the perfect detector, whereas  $\mathcal{E}$  ( $\mathcal{R}$  can use the same storage as  $\mathcal{E}$ ),  $f(x_{n-2})$ ,  $x_{n-2}$  and  $x_{n-3}$  are stored for our detectors; AID stores  $x_{n-2}$ ,  $x_{n-3}$ ,  $x_{n-4}$ , and the extrapolated solution; and BSS14 stores  $\mathcal{E}$ .

The true positive rate (TPR) shows that our detectors detect perfectly (at 99.9%) significant SDCs. Replication does as well, but the BSS14 and AID detectors have a TPR of 91.2% and 86.7% of significant SDCs, respectively. For BSS14 and AID, some SDCs can thus be undetected while affecting the accuracy of the solvers. Moreover, the “IRE 95%” value of our detectors is smaller than the mean local error estimate ( $1.5e^{-6}$ ) by a factor of 100. Because all significant SDCs are detected, SDCs undetected by Hot Rod are sure to have no impact. The undetected 76.9% of SDCs by Hot Rod LFP are thus insignificant and do not need to be corrected: correcting these insignificant SDCs would not improve results and would demand extra computation. Figure 2 shows the LTE of the solver in the confidence interval in the absence of SDC. It represents the approximation error. As defined in Sect. 1, significant SDCs inject errors that are higher than this error. Because the streamlines of the AID and BSS14 detectors are pushed outside the confidence interval at SDC injections, they do not detect those SDCs. On the other hand, Hot Rod HR and LFP’s streamlines are not affected by SDCs:

these detectors protected the solver. This result is consistent with the fact that the IRE 95% of Hot Rod is two orders of magnitudes less than the approximation error.

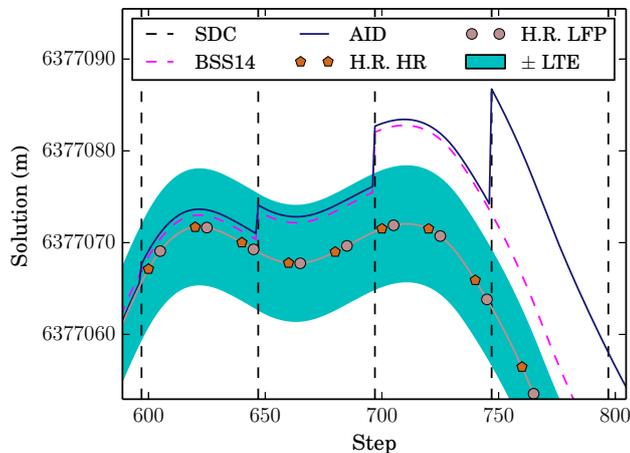


Figure 2. One streamline computed by the different detectors. Singlebit injection is made every 50 steps. In the window, the position of the bit-flip varies from 31 to 35 in IEEE754 doubleprecision. The interval “ $\pm$  LTE ” represents the approximation error. Significant SDCs shift the solution outside this interval. In the application, the origin is the center of the Earth.

## 5 Conclusion

This study presented our SDC detection method Hot Rod for ODE integration solvers. Both experimental and theoretical results show that all significant SDCs are detected. Except for replication, no other tested SDC detectors achieve these results. More specifically, compared with the algorithmic detection SDC detectors, the true positive rate is improved by 52% for singlebit corruptions; whereas compared with replication, the computational overhead is reduced by 20 times. Moreover, users need only to fix the maximum false positive rate, as explained in Sect. 3.

Our detectors were employed for one of the ODE integration methods. Other embedded Runge-Kutta methods can be directly employed. Radau’s estimates have a general expression in the case of adaptive step size; see the work of Butcher and Johnston [4]. For implicit methods or linear multisteps, Richardson’s estimates can also be used. In future work, we plan to investigate detection in partial differential equation solvers.

**Acknowledgments** We express our gratitude to Julie Bessac for assistance with the algorithm and Gail Pieper for comments that greatly improved the manuscript. We also gratefully acknowledge the use of the services and facilities of the Decaf project at Argonne National Laboratory, supported by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357, program manager Lucy Nowell. We also thank the anonymous reviewers for their helpful comments.

## References

1. Bagatin, M., Gerardin, S.: *Ionizing Radiation Effects in Electronics: From Memories to Imagers. Devices, Circuits, and Systems*, CRC Press (2015)
2. Bairavasundaram, L.N., Goodson, G.R., Pasupathy, S., Schindler, J.: An analysis of latent sector errors in disk drives. In: *ACM SIGMETRICS Performance Evaluation Review*. vol. 35, pp. 289–300 (2007)
3. Benson, A.R., Schmit, S., Schreiber, R.: Silent error detection in numerical time-stepping schemes. *International Journal of High Performance Computing Applications* (2014)
4. Butcher, J., Johnston, P.: Estimating local truncation errors for Runge-Kutta methods. *Journal of Computational and Applied Mathematics* 45(1), 203–212 (1993)
5. Cash, J.R., Karp, A.H.: A variable order Runge-Kutta method for initial value problems with rapidly varying right-hand sides. *ACM TOMS* 16(3), 201–222 (1990)
6. Ceschino, F., Kuntzmann, J.: *Numerical solution of initial value problems* (1966)
7. Chen, L., Avizienis, A.: N-version programming: A fault-tolerance approach to reliability of software operation. In: *Digest of Papers FTCS-8*. pp. 3–9 (1978)
8. Di, S., Cappello, F.: Adaptive impact-driven detection of silent data corruption for HPC applications. *IEEE Transactions on Parallel and Distributed Systems* (2016)
9. Ghosh, S., Basu, S., Toubia, N.A.: Selecting error correcting codes to minimize power in memory checker circuits. *Journal of Low Power Electronics* pp. 63–72 (2005)
10. Guerraoui, R., Schiper, A.: Software-based replication for fault tolerance. *Computer* (4), 68–74 (1997)
11. Guo, H., He, W., Peterka, T., Shen, H.W., Collis, S.M., Helmus, J.J.: Finite-time lyapunov exponents and lagrangian coherent structures in uncertain unsteady flows. *IEEE TVCG (Proc. PacificVis 16)* 22 (2016), to appear
12. Huang, K.H., Abraham, J., et al.: Algorithm-based fault tolerance for matrix operations. *IEEE Transactions on computers* 100(6), 518–528 (1984)
13. Hwang, A.A., Stefanovici, I.A., Schroeder, B.: Cosmic rays don't strike twice: understanding the nature of DRAM errors and the implications for system design. In: *ACM SIGPLAN Notices*. vol. 47, pp. 111–122 (2012)
14. Krishnamoorthy, K., Mathew, T.: *Statistical tolerance regions: theory, applications, and computation*, vol. 744. John Wiley & Sons (2009)
15. Lapinsky, S.E., Easty, A.C.: Electromagnetic interference in critical care. *Journal of Critical Care* 21(3), 267–270 (2006)
16. McLoughlin, T., Laramée, R.S., Peikert, R., Post, F.H., Chen, M.: Over two decades of integration-based, geometric flow visualization. In: *Eurographics 2009 State of the Art Report*. pp. 73–92. Munich, Germany (2009)
17. Peterka, T., Ross, R., Nouanesengsy, B., Lee, T.Y., Shen, H.W., Kendall, W., Huang, J.: A study of parallel particle tracing for steady-state and time-varying flow fields. In: *IPDPS*. pp. 580–591. IEEE (2011)
18. Snir, M., Wisniewski, R.W., Abraham, J.A., Adve, S.V., Bagchi, S., Balaji, P., Belak, J., Bose, P., Cappello, F., Carlson, B., et al.: Addressing failures in exascale computing. *International Journal of High Performance Computing Applications* (2014)
19. Stoller, L., Morrison, D.: A method for the numerical integration of ordinary differential equations. *Mathematical Tables and Other Aids to Computation* 12 pp. 269–272 (1958)