Math.Comput.Sci. (2011) 5:359–375
DOI 10.1007/s11786-011-0103-4

Mathematics in Computer Science

Stochastic Arithmetic in Multiprecision

Stef Graillat · Fabienne Jézéquel · Shiyue Wang · Yuxiang Zhu

Received: 3 November 2010 / Revised: 26 April 2011 / Accepted: 31 May 2011 / Published online: 17 November 2011

Abstract Floating-point arithmetic precision is limited in length the IEEE single (respectively double) precision format is 32-bit (respectively 64-bit) long. Extended precision formats can be up to 128-bit long. However some problems require a longer floating-point format, because of round-off errors. Such problems are usually solved in arbitrary precision, but round-off errors still occur and must be controlled. Interval arithmetic has been implemented in arbitrary precision, for instance in the MPFI library. Interval arithmetic provides guaranteed results, but it is not well suited for the validation of huge applications. The CADNA library estimates round-off error propagation using stochastic arithmetic. CADNA has enabled the numerical validation of real-life applications, but it can be used in single precision or in double precision only. In this paper, we present a library called SAM (Stochastic Arithmetic in Multiprecision). It is a multiprecision extension of the classic CADNA library. In SAM (as in CADNA), the arithmetic and relational operators are overloaded in order to be able to deal with stochastic numbers. As a consequence, the use of SAM in a scientific code needs only few modifications. This new library SAM makes it possible to dynamically control the numerical methods used and more particularly to determine the optimal number of iterations in an iterative process. We present some applications of SAM in the numerical validation of chaotic systems modeled by the logistic map.

Keywords Stochastic arithmetic · Multiprecision · MPFR · MPFI · Interval arithmetic

Mathematics Subject Classification (2010) Primary 68N01; Secondary 68U20

This work has partially been carried out during S. Wang's and Y. Zhu's training periods at LIP6.

S. Graillat (✉) · F. Jézéquel · S. Wang · Y. Zhu
Laboratoire d'Informatique de Paris 6, UPMC Univ Paris 06, UMR 7606, 4 place Jussieu, 75252 Paris Cedex 05, France
e-mail: Stef.Graillat@lip6.fr

F. Jézéquel
e-mail: Fabienne.Jezequel@lip6.fr

S. Wang
e-mail: Shiyue.Wang@etu.upmc.fr

Y. Zhu
e-mail: Hareton.Zhu@gmail.com

1 Introduction

The increasing power of current computers enables one to solve more and more complex problems. Then it is necessary to perform a high number of floating-point operations, each one leading to a round-off error. Because of round-off error propagation, some problems must be solved with a longer floating-point format. This is the case, especially, for applications which carry out very complicated and enormous tasks in scientific fields, for example (see http://crd.lbl.gov/~dhbailey/dhbpapers/hpmpd.pdf):

- Quantum field theory
- Supernova simulation
- Semiconductor physics
- Planetary orbit calculations
- Experimental and computational mathematics.

Even if other techniques, methods or algorithms are employed to increase the accuracy of numerical results, some extended precision is still required to avoid severe numerical inaccuracies. Therefore some versions of scientific software carry out multiprecision computation. For instance, XBLAS routines [1] consist of a multiprecision version of basic linear algebra routines (BLAS). Moreover some libraries implement arbitrary precision arithmetic. MPFR [2] is an arbitrary precision library developed by INRIA in C language. Freely available on various platforms, MPFR uses very efficient algorithms.

Nevertheless, even with multiprecision, there are still some rounding errors and so it is still necessary to get some information about the numerical quality of the results. Validation of numerical computations has been studied for a long time (see for example [3–6]). Several approaches exist to control round-off error propagation: backward analysis [3] (used in LAPACK), stochastic backward analysis [6] (used in PRECISE), interval arithmetic [7–9], stochastic arithmetic [10,11], and abstract interpretation-based static analysis [12]. A survey on rounding errors analysis is given in [13]. Based on MPFR, the MPFI library [14] provides arbitrary precision interval arithmetic. The problem is that it is difficult to validate huge scientific codes with interval arithmetic. If such a code is not rewritten to deal with intervals, its results would often be so overestimated that they would provide no information at all.

In this paper, we present the SAM (Stochastic Arithmetic in Multiprecision) library which is based on MPFR and extends the features of discrete stochastic arithmetic by enabling arbitrary precision computation. It can be used to validate real-life scientific codes with few modifications of the code.

In Sect. 2, we briefly describe interval arithmetic as well as the MPFR and MPFI multiprecision libraries. Section 3 is devoted to Discrete Stochastic Arithmetic and the CADNA library, its implementation in IEEE single and double precision. In Sect. 4, we describe the SAM library as well as how to use it. In Sect. 5, we present some numerical experiments carried out using the SAM library. We study Rump's polynomial, Muller's sequence, the dynamic control of integral computation as well as the logistic map. We also provide some performance tests. Conclusions and perspectives are given in Sect. 6.

2 MPFR, Interval Arithmetic and MPFI

In order to improve the accuracy of numerical results, the working precision can be increased. For this purpose, some multiprecision libraries have been developed. One can divide the libraries into three categories.

- Arbitrary precision libraries using a *multiple-digit* format where a number is expressed as a sequence of digits coupled with a single exponent. Examples of this format are Bailey's MPFUN/ARPREC [15], Brent's MP [16] or MPFR [2].
- Arbitrary precision libraries using a *multiple-component* format where a number is expressed as unevaluated sums of ordinary floating-point words. Examples using this format are Priest's [17] and Shewchuk's [18] libraries.

- Extended fixed precision libraries using the *multiple-component* format but with a limited number of components. Examples of this format are Bailey's *double-double* [19] (double-double numbers are represented as an unevaluated sum of a leading double and a trailing double) and *quad-double* [20].

In the sequel, we will focus on the MPFR arbitrary precision library, since it provides the four IEEE 754-1985 [21] rounding modes and correct rounding for all the operations and mathematical functions it implements. Written in the ISO C language, MPFR is based on the GNU MP library [22] (GMP for short). The internal representation of a floating-point number x by MPFR is a mantissa m, a sign s and a signed exponent e. If the precision of x is p, then the mantissa m has p significant bits. The mantissa m is represented by an array of native unsigned integers. The semantic in MPFR is as follows: for each instruction $a = f(b, c)$ the variables may have different precisions. In MPFR, the data b and c are considered with their full precision and a correct rounding to the full precision of a is computed. Applications using MPFR inherit the same properties as programs using the IEEE 754 standard (portability, well-defined semantics, possibility to design robust programs and prove their correctness) with no significant slowdown on average with respect to arbitrary precision libraries with ill-defined semantics.

The presentation of interval arithmetic in this paragraph heavily relies on [13]. Interval arithmetic [8,23] is defined not on real numbers, but on closed intervals. The result of an arithmetic operation between two intervals, $X = [\underline{x}, \overline{x}]$ and $Y = [\underline{y}, \overline{y}]$, is defined as the set of all values that can be obtained by performing this operation on elements from each interval. The resulting arithmetic operations are listed below.

$$X + Y = [\underline{x} + \underline{y},\ \overline{x} + \overline{y}]. \tag{2.1}$$

$$X - Y = [\underline{x} - \overline{y},\ \overline{x} - \underline{y}]. \tag{2.2}$$

$$X \times Y = [\min(\underline{x} \times \underline{y}, \underline{x} \times \overline{y}, \overline{x} \times \underline{y}, \overline{x} \times \overline{y}),\ \max(\underline{x} \times \underline{y}, \underline{x} \times \overline{y}, \overline{x} \times \underline{y}, \overline{x} \times \overline{y})]. \tag{2.3}$$

$$X^2 = [\min(\underline{x}^2, \overline{x}^2),\ \max(\underline{x}^2, \overline{x}^2)] \quad \text{if } 0 \notin [\underline{x},\ \overline{x}],\ [0, \max(\underline{x}^2, \overline{x}^2)] \text{ otherwise.}$$

$$1/Y = [\min(1/\underline{y}, 1/\overline{y}),\ \max(1/\underline{y}, 1/\overline{y})] \quad \text{if } 0 \notin [\underline{y}, \overline{y}]. \tag{2.4}$$

$$X/Y = [\underline{x},\ \overline{x}] \times (1/[\underline{y},\ \overline{y}]) \quad \text{if } 0 \notin [\underline{y},\ \overline{y}]. \tag{2.5}$$

Actually we can also deal with empty intervals and unbounded intervals. But for simplicity we only mention operations on nonempty bounded intervals. As a remark, arithmetic operations can also be applied to interval vectors and interval matrices by performing scalar interval operations componentwise.

Interval arithmetic enables one to control rounding errors automatically. On a computer, a real value which is not machine representable can be approximated to a floating-point number. It can also be enclosed by two floating-point numbers. Real numbers can therefore be replaced by intervals with machine representable bounds. An interval operation can be performed using directed rounding modes, in such a way that the rounding error is taken into account and the exact result is necessarily contained in the computed interval. For instance, the computed results, with guaranteed bounds, of the addition and the subtraction between two intervals $X = [\underline{x}, \overline{x}]$ and $Y = [\underline{y}, \overline{y}]$ are

$$X + Y = [\nabla(\underline{x} + \underline{y}), \Delta(\overline{x} + \overline{y})] \supseteq \{x + y | x \in X, y \in Y\} \tag{2.6}$$

$$X - Y = [\nabla(\underline{x} - \overline{y}), \Delta(\overline{x} - \underline{y})] \supseteq \{x - y | x \in X, y \in Y\} \tag{2.7}$$

where ∇ (respectively Δ) denotes the downward (respectively upward) rounding mode.

The main advantage of interval arithmetic is its reliability. But the intervals obtained may be too large. The intervals width regularly increases with respect to the intervals that would have been obtained in exact arithmetic. With interval arithmetic, rounding error compensation is not taken into account. As the classical numerical algorithms can lead to over-pessimistic results in interval arithmetic, specific algorithms, suited for interval arithmetic, have been proposed. Interval arithmetic has been implemented in several libraries or software. For instance, a C++ class library, C-XSC,[1] and a Matlab toolbox, INTLAB,[2] are freely available.

MPFI is a portable library written in C for arbitrary precision interval arithmetic where intervals are represented using MPFR floating-point numbers. The purpose of MPFI is twofold: it is to get guaranteed results thanks to the

[1] http://www.xsc.de.

[2] http://www.ti3.tu-harburg.de/rump/intlab.

reliability of interval computation, but also to get accurate results, thanks to MPFR multiple precision arithmetic. In the sequel, we have compared MPFI with the SAM library, because both are based on MPFR.

3 Discrete Stochastic Arithmetic

Based on a probabilistic approach, the CESTAC method [11] allows the estimation of round-off error propagation which occurs with floating-point arithmetic. This method uses a random rounding mode: at each elementary operation, the result is rounded up or down with the probability 0.5. With this random rounding mode, the same program run several times provides different results, due to different round-off errors.

It has been proved [24] that a computed result R is modelled to the first order in 2^{-p} as:

$$R \approx Z = r + \sum_{i=1}^{n} g_i(d) 2^{-p} z_i \tag{3.1}$$

where r is the exact result, $g_i(d)$ are coefficients depending exclusively on the data and on the code, p is the number of bits in the mantissa and z_i are independent uniformly distributed random variables on $[-1, 1]$.

From Eq. (3.1), we deduce that:

1. the mean value of the random variable Z is the exact result r,
2. the distribution of Z is a quasi-Gaussian distribution.

Then by identifying R and Z, i.e. by neglecting all the second order terms, Student's test can be used to estimate the accuracy of R. From N samples R_i, $i = 1, 2, \ldots, N$,

$$\forall \beta \in [0, 1], \exists \tau_\beta \in \mathbb{R} \quad \text{such that} \quad P\left(\left| \overline{R} - r \right| \leq \frac{\sigma \tau_\beta}{\sqrt{N}} \right) = 1 - \beta \tag{3.2}$$

where

$$\overline{R} = \frac{1}{N} \sum_{i=1}^{N} R_i \quad \text{and} \quad \sigma^2 = \frac{1}{N-1} \sum_{i=1}^{N} \left(R_i - \overline{R} \right)^2. \tag{3.3}$$

τ_β is the value of Student's distribution for $N - 1$ degrees of freedom and a probability level $1 - \beta$.

Therefore the number of decimal significant digits common to $\overline{R}$ and r can be estimated with the following equation.

$$C_{\overline{R}} = \log_{10} \left(\frac{\sqrt{N} \left| \overline{R} \right|}{\sigma \tau_\beta} \right). \tag{3.4}$$

Thus the implementation of the CESTAC method in a code providing a result R consists in:

- performing N times this code with the random rounding mode, which is obtained by using randomly the upward or downward rounding mode; we then obtain N samples R_i of R
- choosing as the computed result the mean value $\overline{R}$ of R_i, $i = 1, \ldots, N$
- estimating with Eq. (3.4) the number of exact decimal significant digits of $\overline{R}$.

In practice $\beta = 0.05$ and $N = 3$. Indeed, it has been shown [25,10] that $N = 3$ is in some reasonable sense the optimal value. The estimation with $N = 3$ is more reliable than with $N = 2$ and increasing the size of the sample does not improve the quality of the estimation. The complete theory can be found in [10,11].

Equations (3.1) and (3.4) hold if two main hypotheses are verified. These hypotheses are:

1. the round-off errors α_i are independent, centered uniformly distributed random variables,
2. the approximation to the first order in 2^{-p} is legitimate.

Concerning the first hypothesis, with the use of the random arithmetic, round-off errors α_i are random variables, however, in practice, they are not rigorously centered and in this case Student's test gives a biased estimation of the computed result. It has been proved [25] that, with a bias of a few σ, the error on the estimation of the number of exact significant digits of $\overline{R}$ is less than one decimal digit. Therefore even if the first hypothesis is not rigorously satisfied, the estimation obtained with Eq. (3.4) is still correct up to one digit.

Concerning the second hypothesis, the approximation to the first order only concerns multiplications and divisions. Indeed the round-off error generated by an addition or a subtraction does not contain any term of higher order. It has been shown [24,10] that, if a computed result becomes insignificant, i.e. if the round-off error it contains is of the same order of magnitude as the result itself, then the first order approximation may be not legitimate. In practice the validation of the CESTAC method requires a dynamical control of multiplications and divisions, during the execution of the code. This leads to the synchronous implementation of the method, i.e. to the parallel computation of the N results R_i, and also to the concept of computational zero, also named informatical zero [26,27].

Definition 3.1 During the run of a code using the CESTAC method, an intermediate or a final result R is a computational zero, denoted by @.0, if one of the two following conditions holds:

- $\forall i,\ R_i = 0$,
- $C_{\overline{R}} \leq 0$.

Any computed result R is a computational zero if either $R = 0$, R being significant, or R is insignificant. In other words, a computational zero is a value that cannot be differentiated from the mathematical zero because of its rounding error. As a remark, in a program, a variable R that is insignificant at a point can become significant later. For instance, in a single precision program (with 24-bit mantissa length) if $R = (10^{-16}, 2.10^{-16}, 3.10^{-16})$, the three samples representing R have no common significant digits. After the instruction $R = R + A$ with $A_1 = A_2 = A_3 = 2$, the value of each sample representing R is 2, so the computed result $\overline{R}$ is 2 with the maximal number of exact significant digits for the precision used. An insignificant result is, like any stochastic number, represented by three samples. The only difference is that it is printed as @.0.

From this new concept of zero, one can deduce new order relationships that take into account the accuracy of intermediate results. For instance,

Definition 3.2 X is stochastically strictly greater than Y if and only if:

$$\overline{X} > \overline{Y} \quad \text{and} \quad X - Y \neq @.0.$$

or

Definition 3.3 X is stochastically greater than or equal to Y if and only if:

$$\overline{X} \geq \overline{Y} \quad \text{or} \quad X - Y = @.0.$$

Stochastic relational operators ensure that in a branching statement the same sequence of instructions is performed for all the samples which represent a variable.

The joint use of the CESTAC method and these new definitions is called DSA (Discrete Stochastic Arithmetic). DSA enables to estimate the impact of rounding errors on any result of a scientific code and also to check that no anomaly occurred during the run, especially in branching statements. The CADNA software [10,28–31] is a library which implements DSA in any code written in C++ or in Fortran and allows to use new numerical types: the stochastic types. The library contains the definition of all arithmetic operations and order relations for the stochastic types. The control of the accuracy is performed only on variables of stochastic type. When a stochastic variable is printed, only its exact significant digits appear. For a computational zero, the string "@.0" is printed. In contrast to interval arithmetic, that computes guaranteed results, the CADNA software provides, with the probability 95% the number of exact significant digits of any computed result. Because all operators are overloaded, the use of CADNA in a program requires only a few modifications. The CADNA software has been successfully used for the numerical validation of real-life applications [32–36].

4 The SAM Library

The SAM library implements in arbitrary precision the features of DSA: the stochastic types, the concept of computational zero and the stochastic operators. The particularity of SAM (compared to CADNA) is the arbitrary precision of stochastic variables. The SAM library with 24-bit (resp. 53-bit) mantissa length is similar to CADNA in single (resp. double) precision, except the range of the exponent is only limited by the machine memory. In SAM, the number of exact significant digits of any stochastic variable is estimated with the probability 95%, whatever its precision. Like in CADNA, the arithmetic and relational operators in SAM take into account round-off error propagation. All numerical instabilities which occur at run time are detected. Such instabilities are usually generated by an operation involving a computational zero.

The SAM library is written in C++ and is based on MPFR. In the SAM library, all operators are overloaded. Consequently for a program written in C++ to be used with SAM, only a few modifications are needed: mainly changes in type declarations. Classical variables have to be replaced by multiprecision stochastic variables (of `mp_st` type) which consist of three variables of MPFR type and one integer variable to store the accuracy. In SAM, for each stochastic operation, three MPFR operations are performed using different rounding modes and the numerical instability that may be generated is detected. As a remark, in order to avoid using the same rounding mode for the three operations, the first two operations are performed using a rounding mode randomly chosen and the third rounding mode is the opposite of the second one. For instance, if the result of the second operation is rounded up, then the result of the third operation is rounded down.

The use of the SAM library in a C++ program involves five steps described below.

1. The SAM library has to be included in the C++ program using the `#include "sam.h"` directive.
2. The `sam_init` initialization function has to be called once. In the following instruction
   `sam_init(nb_instabilities, nb_bits);`
   `nb_instabilities` is the maximum number of numerical instabilities to be detected and `nb_bits` is the mantissa length in bits. If the value of the first argument is -1, all the instabilities will be detected.
3. In variable declarations, the `float` or `double` type has to be replaced by the `mp_st` stochastic type.
4. The `strp` function returns a string which contains the exact significant digits of a stochastic result, i.e. its significant digits not affected by round-off errors. If the result is insignificant, the `strp` function returns "@.0". Therefore output statements should be modified to print stochastic results with their accuracy using the `strp` function.
5. The `sam_end` function has to be called to print a report on the numerical instabilities which possibly occurred during the execution.

As an example, a simple program which uses the SAM library is given in 5.1.

5 Numerical Experiments

5.1 Computing A Rational Function of Two Variables

In the following example proposed by Rump [37], the rational function

$$f(x, y) = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + \frac{x}{2y}$$

is computed with $x = 77617$, $y = 33096$. The associated program written using the SAM library is:

```
#include ``sam.h''
#include <stdio.h>
int main() {
sam_init(-1,122);
mp_st x = 77617, y = 33096, res;
```

```
res=333.75*y*y*y*y*y*y*y+x*x*(11*x*x*y*y-y*y*y*y*y*y*y
-121*y*y*y*y-2.0)+5.5*y*y*y*y*y*y*y*y+x/(2*y);
printf(``res=%s\n'',strp(res));
sam_end();
}
```

Because the first argument in the `sam_init` function is -1, the SAM library will detect all the numerical instabilities which will possibly occur at run time. Using SAM with 122 bits, one obtains:

```
-----------------------------------------------------------------
SAM software --- University P. et M. Curie --- LIP6
Self-validation detection: ON
Mathematical instabilities detection: ON
Branching instabilities detection: ON
Intrinsic instabilities detection: ON
Cancellation instabilities detection: ON
-----------------------------------------------------------------
res=-0.827396059946821368141165095479816292
-----------------------------------------------------------------
SAM software --- University P. et M. Curie --- LIP6
No instability detected
-----------------------------------------------------------------
```

Using SAM with 53 bits, the result obtained has no exact significant digits. Furthermore SAM detects a cancellation, which is a severe loss of accuracy due to the subtraction of two nearly equal numbers. Let us decompose the expression into three terms: $f(x, y) = a + b + c$ with $a = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2)$, $b = 5.5y^8$ and $c = x/(2y)$. SAM with 53 bits provides two opposite results for a and b:

`a = -0.791711134066896E+037`
`b = 0.791711134066896E+037.`

The computed results a and b have the same 15 exact significant digits. Their sum is insignificant, because it is only due to rounding errors. The three samples representing this sum are:

`-1.1805916207174113E+021, -1.1805916207174113E+021, -0.0000000000000000E+000.`

The computed result c has 15 exact significant digits:

`c= 0.117260394005317E+001.`

The final result $a + b + c$ is insignificant. It is represented by the three samples:

`-1.1805916207174113E+021, -1.1805916207174112E+021, +1.1726039400531785E+000.`

Table 1 shows that if computations carried out with different precisions lead to similar approximations, it does not mean that the results have a good accuracy. Indeed, it is shown that the evaluations of Rump's polynomial f in single, double and extended precision share the same first digits. But in fact, the correct evaluation is totally different from that result. This is detected by SAM. Until 121 bits, SAM says that the result has no exact significant digits. It is only starting with 122-bit mantissa length that SAM says that we have full accuracy. It is verified by

Table 1 Computation of $f(x, y) = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$ with $x = 77{,}617$ and $y = 33{,}096$

Single precision	1.172603
Double precision	1.1726039400531
Extended precision	1.172603940053178
Variable precision	[−0.827396059946821368141165095479816292005,
Interval arithmetic	−0.827396059946821368141165095479816291986]
SAM, ≤121 bits	@.0
SAM, 122 bits	−0.827396059946821368141165095479816292

computing the exact result with the Maple computer algebra system. It is noticeable that the result provided by SAM with 122-bit mantissa length is included in the interval computed using variable precision interval arithmetic.

5.2 Computing A Second Order Recurrent Sequence

This sequence was proposed by Muller [38]. The first 30 iterations of the following recurrent sequence are computed:

$$U_{n+1} = 111 - \frac{1130}{U_n} + \frac{3000}{U_n U_{n-1}}$$

with $U_0 = 5.5$ and $U_1 = \frac{61}{11}$. The exact limit is 6.

Using IEEE double precision arithmetic with rounding to the nearest, one obtains:

```
U(3) = +5.590163934426237e+00
(...)
U(11) = +5.861018785996283e+00
U(12) = +5.882524608269310e+00
U(13) = +5.918655323805488e+00
U(14) = +6.243961815306110e+00
U(15) = +1.120308737284091e+01
U(16) = +5.302171264499677e+01
U(17) = +9.473842279276452e+01
U(18) = +9.966965087355071e+01
U(19) = +9.998025776093678e+01
U(20) = +9.999882245337588e+01
(...)
U(30) = +1.000000000000000e+02
```

Until the 13th iteration, the sequence seems to converge to 6. Then it seems to converge to 100, although the exact limit is 6.

Using SAM in double precision (with 53-bit mantissa length), one obtains:

```
----------------------------------------------------------------
SAM software --- University P. et M. Curie --- LIP6
Self-validation detection: ON
Mathematical instabilities detection: ON
Branching instabilities detection: ON
Intrinsic instabilities detection: ON
Cancellation instabilities detection: ON
----------------------------------------------------------------
U(3) = 0.5590163934426E+1
U(4) = 0.563343108504E+1
U(5) = 0.56746486205E+1
U(6) = 0.571332905E+1
U(7) = 0.57491209E+1
U(8) = 0.5781811E+1
U(9) = 0.581131E+1
U(10) = 0.58376E+1
U(11) = 0.586E+1
U(12) = 0.59E+1
U(13) = 0.6E+1
U(14) = @.0
U(15) = @.0
U(16) = @.0
U(17) = @.0
U(18) = 0.9E+2
U(19) = 0.99E+2
U(20) = 0.999E+2
(...)
U(30) = 0.100000000000000E+3
```

```
------------------------------------------------------------------
SAM software --- University P. et M. Curie --- LIP6
CRITICAL WARNING: the self-validation detects major problem(s).
The results are NOT guaranteed.
There are 12 numerical instabilities
9 UNSTABLE DIVISION(S)
3 UNSTABLE MULTIPLICATION(S)
------------------------------------------------------------------
```

First the sequence seems to converge to 6, losing regularly exact significant digits to such an extent that iterates 14 to 17 are insignificant. Then the sequence converges to 100, improving regularly its numerical quality. The accuracy of the 30th iterate (15 exact significant digits) is optimal for a computation carried out in double precision. Unfortunately results from the 18th iteration are incorrect. The instabilities reported at the end of the execution are unstable divisions and unstable multiplications which may invalidate the first order approximation in the CESTAC method. A warning informs the user that the results may not be reliable.

Using SAM with 100 bits, one obtains:

```
------------------------------------------------------------------
SAM software --- University P. et M. Curie --- LIP6
Self-validation detection: ON
Mathematical instabilities detection: ON
Branching instabilities detection: ON
Intrinsic instabilities detection: ON
Cancellation instabilities detection: ON
------------------------------------------------------------------
U(3) = 0.55901639344262295081967213 1E+1
(...)
U(22) = 0.5979E+1
U(23) = 0.598E+1
U(24) = 0.59E+1
U(25) = @.0
U(26) = @.0
U(27) = @.0
U(28) = @.0
U(29) = 0.9E+2
U(30) = 0.99E+2
U(31) = 0.999E+2
(...)
U(52) = 0.10000000000000000000000000000E+3
------------------------------------------------------------------
SAM software --- University P. et M. Curie --- LIP6
CRITICAL WARNING: the self-validation detects major problem(s).
The results are NOT guaranteed.
There are 12 numerical instabilities
9 UNSTABLE DIVISION(S)
3 UNSTABLE MULTIPLICATION(S)
------------------------------------------------------------------
```

The sequence obtained using SAM with 100 bits has the same behavior as the previous one (computed with 53 bits). The sequence seems to converge to 6, then four iterates have no exact significant digits. These insignificant results appear later, at iterations 25–28. From the 29th iteration, the sequence converges to 100, improving regularly its accuracy. Like in the previous execution, a self-validation of SAM has been performed during the run. Because unstable divisions and unstable multiplications occurred, the results are not reliable.

5.3 Dynamical Control of Integral Computation

Previous work had been carried out on the dynamical control of approximation methods using the CADNA library in single or in double precision [34,35,39–42]. We show that the strategies previously proposed can be extended to

arbitrary precision. The theoretical results recalled here require the notion of significant digits common to two real numbers. Therefore we need the following definition.

Definition 5.1 Let a and b be two real numbers, the number of significant digits that are common to a and b can be defined in $\mathbb{R}$ by

1. for $a \neq b$, $C_{a,b} = \log_{10} \left| \frac{a+b}{2(a-b)} \right|$,
2. $\forall a \in \mathbb{R},\ C_{a,a} = +\infty$.

Then $|a-b| = \left|\frac{a+b}{2}\right| 10^{-C_{a,b}}$. For instance, if $C_{a,b} = 3$, the relative difference between a and b is of the order of 10^{-3} which means that a and b have three significant digits in common.

Remark 5.2 The value of $C_{a,b}$ can seem surprising if we consider the decimal notations of a and b. For example, if $a = 2.4599976$ and $b = 2.4600012$, then $C_{a,b} \approx 5.8$. The difference due to the sequences of "0" or "9" is illusive. The significant decimal digits of a and b are really different from the sixth position.

A dynamical control of integral computation can be performed using DSA (Discrete Stochastic Arithmetic). Theorem 5.3 has been established from the truncation error expansion for closed Newton-Cotes quadrature rules. Its proof can be found in [42]. It can apply, for instance, to the trapezoidal rule (of order 2) or to Simpson's rule (of order 4).

Theorem 5.3 *Let I_n be the approximation to $I = \int_a^b f(x)dx$ by a composite closed Newton-Cotes quadrature rule of order p with the step $\frac{b-a}{2^n}$. If $f \in \mathcal{C}^{p+2}[a,b]$ and $f^{(p-1)}(b) \neq f^{(p-1)}(a)$, then*

$$C_{I_n,I_{n+1}} = C_{I_n,I} + log_{10}\left(\frac{2^p}{2^p-1}\right) + \mathcal{O}\left(\frac{1}{4^n}\right). \tag{5.1}$$

If the convergence zone is reached, i.e. if the term $\mathcal{O}\left(\frac{1}{4^n}\right)$ becomes negligible, the significant digits common to two successive approximations I_n and I_{n+1} are also in common with the exact result I, up to one bit. Indeed the term $\log_{10}(2^p/(2^p-1))$ decreases as p increases and it corresponds to one bit for methods of order 1. Let us assume that the sequence (I_n) is computed in DSA and that the convergence zone is reached. If the difference between I_n and I_{n+1} is only due to round-off errors, further iterations are useless. Therefore if computations are performed until the difference $I_n - I_{n+1}$ has no exact significant digits, the optimal iterate I_{n+1} can be dynamically determined at run time. Furthermore, from Theorem 5.3, its significant bits which are not affected by round-off errors are in common with the exact result I, up to one.

A similar strategy can be used with Romberg's method, which is based on Richardson's extrapolation on results of the trapezoidal method. Let f be a real function over $[a,b]$ and I be the exact value of $\int_a^b f(x)dx$. Let $T_1(h)$ be the approximation to I computed using the trapezoidal method with step $h = \frac{b-a}{M}$ ($M \geq 1$). Romberg's method consists in computing the following triangular table:

$$\begin{array}{llllll}
T_1(h) & T_1(\frac{h}{2}) & \ldots\ldots & T_1(\frac{h}{2^{n-3}}) & T_1(\frac{h}{2^{n-2}}) & T_1(\frac{h}{2^{n-1}}) \\
T_2(h) & T_2(\frac{h}{2}) & \ldots\ldots & T_2(\frac{h}{2^{n-3}}) & T_2(\frac{h}{2^{n-2}}) & \\
T_3(h) & T_3(\frac{h}{2}) & \ldots\ldots & T_3(\frac{h}{2^{n-3}}) & & \\
\vdots & \vdots & & & & \\
T_{n-1}(h) & T_{n-1}(\frac{h}{2}) & & & & \\
T_n(h) & & & & &
\end{array}$$

The first row of the table represents approximations to I computed using the trapezoidal method with step $\frac{h}{2^j}$ ($j = 0, \ldots, n-1$). Rows 2 to n are computed using the following formula.

For $r = 2, \ldots, n$ and $j = 0, \ldots, n-r$,

$$T_r\left(\frac{h}{2^j}\right) = \frac{1}{4^{r-1}-1}\left(4^{r-1}T_{r-1}\left(\frac{h}{2^{j+1}}\right) - T_{r-1}\left(\frac{h}{2^j}\right)\right). \tag{5.2}$$

The sequence of approximations with Romberg's method $T_1(h), \ldots, T_n(h)$ converges exponentially to I.

The dynamical control of Romberg's method is based on Theorem 5.4. Its proof can be found in [41].

Theorem 5.4 *Let us assume that f is a real function which is $\mathcal{C}^k$ over $[a, b]$ where $k \geq 2n+1$ and that $f^{(2n-1)}(a) \neq f^{(2n-1)}(b)$. Let $T_n(h)$ be the approximation to $I = \int_a^b f(x)dx$ computed with n iterations of Romberg's method using the initial step $h = \frac{b-a}{M}$ ($M \geq 1$). Then*

$$C_{T_n(h),T_{n+1}(h)} = C_{T_n(h),I} + \mathcal{O}\left(\frac{h^2}{n^2}\right).$$

In practice, it means that, when the convergence zone is reached, the significant digits common to $T_n(h)$ and $T_{n+1}(h)$ are also common to I. Therefore the first significant digits of the exact value I can be obtained from only two consecutive values of $T_n(h)$. Let us assume that the sequence $(T_n(h))$ is computed in DSA and that the convergence zone is reached. If computations are performed until the difference $T_n(h) - T_{n+1}(h)$ has no exact significant digits, the significant digits of the approximation obtained which are not affected by round-off errors are in common with the exact result I.

The evaluation of the integral I defined in Eq. (5.3) is a problem which has been posed in [43].

$$I = \int_0^1 \frac{\arctan(\sqrt{2+t^2})}{(1+t^2)\sqrt{2+t^2}}dt \tag{5.3}$$

Its exact value has been indicated in [44]: $I = \frac{5\pi^2}{96}$. Let I_n be the approximation to I computed by using the composite trapezoidal rule or the composite Simpson's rule with the step $\frac{1}{2^n}$, or by performing n iterations of Romberg's method with the initial step $h = 1$. Starting from I_0 (the approximation obtained with no partition of the integration interval), approximations I_n have been computed using the SAM library until the difference $I_n - I_{n+1}$ is a computational zero (has no exact significant digits). Table 2 presents the approximations obtained using different precisions. In every sequence, only the exact significant digits (i.e. not affected by round-off errors) of the last iterate, estimated using SAM, are reported.

In agreement with Theorems 5.3 and 5.4, the exact significant digits of each approximation obtained are in common with I, up to one. The number of iterations required for the stopping criterion to be satisfied may depend

Table 2 Approximations to I, its first 40 exact digits being 0.5140418958900707613976297395768828716309

#bits	trapezoidal method
24	I_9 = 0.51404
53	I_{20} = 0.514041895890
70	I_{26} = 0.5140418958900708
#bits	Simpson's method
24	I_4 = 0.514042
53	I_{10} = 0.51404189589007
70	I_{14} = 0.5140418958900707614
100	I_{21} = 0.514041895890070761397629739
#bits	Romberg's method
24	I_3 = 0.514041
53	I_7 = 0.51404189589007
70	I_8 = 0.5140418958900707614
100	I_{11} = 0.514041895890070761397629739 6

on the precision chosen, but also on the quadrature method used. Indeed the convergence speed of the computed sequence and the numerical quality of the result obtained vary according to the quadrature method.

5.4 Computation of A Chaotic Dynamical System

It is always difficult to simulate chaotic dynamical systems [45,46] with a computer because rounding errors can change dramatically their behavior. Nevertheless, some results can be proved on chaotic systems using rigorous numerical computations and in particular interval computations (see for example [47–49]). In this subsection, we study the computation of the logistic map using stochastic arithmetic with SAM and we compare with MPFI the results obtained.

Let us consider the logistic iteration [45] defined by $x_{n+1} = ax_n(1 - x_n)$ with $a > 0$ and $0 < x_0 < 1$.

- When $a < 3$ this sequence converges to a unique fixed point, whatever the initial condition x_0 is.
- When $3 \leq a \leq 3.57$ this sequence is periodic, whatever the initial condition x_0 is, the periodicity depending only on a. Furthermore the periodicity is multiplied by 2 for some values of a called "bifurcations".
- When $3.57 < a < 4$ this sequence is usually chaotic, but there are certain isolated values of a that appear to show periodic behavior.
- Beyond $a = 4$, the values eventually leave the interval [0,1] and diverge for almost all initial values.

The logistic map has been computed with $x_0 = 0.6$ using SAM and MPFI. In stochastic arithmetic, iterations have been performed until the current iterate is a computational zero, i.e. all its digits are affected by round-off errors. In interval arithmetic, iterations have been performed until the two bounds of the interval have no common significant digits. In Tables 3 and 4, we report the number N of iterations performed for two ways of computing the logistic map. We also report the execution times. With SAM two types of executions have been performed:

- without the detection of numerical instabilities
- with the detection of all numerical instabilities which may occur at run time.

In Table 3, we have computed the logistic map using the formula

$$x_{n+1} = ax_n(1 - x_n) \quad \text{with} \quad x_0 = 0.6. \tag{5.4}$$

During the computation performed using SAM with $a = 3.57$, no computational zero has been detected. The program has been stopped after one million iterations. The run time reported in Table 3 has been measured for one million iterations. Whereas using MPFI with the same value for a, at a certain iteration the stopping criterion is satisfied: the bounds of the last interval have no common digits. Using 500,000 bits, fewer than 300,000 iterations are performed in about 6 h. Since the number N of iterations performed increases linearly as the precision increases, about 2-million bits would be required to perform one million iterations. If $a = 3.575$, $a = 3.6$ or $a = 3.7$, more iterations are performed with SAM than with MPFI for the stopping criterion to be satisfied. However it must be pointed out that SAM is based on an estimation of round-off errors. Results obtained with MPFI are more pessimistic, but are guaranteed. If $a = 10$, the sequence diverges. Similar results are obtained with SAM and MPFI. The run times measured with SAM are longer than those measured with MPFI. The main reason is the number of iterations, higher with SAM than with MPFI. In the numerical experiment presented in 5.5, the same computation is carried out using SAM and MPFI in order to compare their performance. The execution time is higher with SAM when the detection of instabilities is enabled. However, in this application, the cost of this detection remains reasonable.

In Table 4, we have computed the logistic map using the formula

$$x_{n+1} = -a\left(x_n - \frac{1}{2}\right)^2 + \frac{a}{4} \quad \text{with} \quad x_0 = 0.6. \tag{5.5}$$

Concerning SAM, the results are very similar to those in Table 3 (similar N). Nevertheless, the results are better with MPFI compared to Table 3. This can be explained by the so-called *dependency problem* which is a major drawback of interval arithmetic. Indeed, if an interval occurs several times in an expression, each occurrence is

Table 3 Logistic map: number N of iterations performed with SAM and MPFI, $x_{n+1} = ax_n(1 - x_n)$ with $x_0 = 0.6$

a		# bits	N	Execution time (in s)	
3.57	SAM			without detection	with all detections
		24	$> 10^6$	6.668	12.093
		53	$> 10^6$	7.192	13.565
		100	$> 10^6$	9.045	17.485
		200	$> 10^6$	11.565	23.277
		2000	$> 10^6$	160.39	360.65
	MPFI	24	11	$< 10^{-3}$	
		53	27	$< 10^{-3}$	
		100	53	$< 10^{-3}$	
		200	108	0.004	
		2000	1088	0.040	
		5 000	2 722	0.316	
		50 000	27 232	97.270	
		500 000	272 341	20901.6	
3.575	SAM			without detection	with all detections
		24	110	0.008	0.008
		53	324	0.012	0.012
		100	722	0.016	0.020
		200	1526	0.020	0.044
		2000	15896	2.585	5.804
	MPFI	24	12	$< 10^{-3}$	
		53	27	$< 10^{-3}$	
		100	53	$< 10^{-3}$	
		200	108	0.004	
		2000	1087	0.032	
3.6	SAM			without detection	with all detections
		24	52	$< 10^{-3}$	0.008
		53	150	0.008	0.008
		100	338	0.011	0.016
		200	718	0.012	0.028
	MPFI	24	12	$< 10^{-3}$	
		53	27	$< 10^{-3}$	
		100	53	$< 10^{-3}$	
		200	107	0.004	
3.7	SAM			without detection	with all detections
		24	37	$< 10^{-3}$	0.008
		53	97	0.004	0.008
		100	193	0.008	0.012
		200	387	0.012	0.020
	MPFI	24	11	$< 10^{-3}$	
		53	27	$< 10^{-3}$	
		100	52	$< 10^{-3}$	
		200	105	0.004	
10	SAM			without detection	with all detections
		24	21	$< 10^{-3}$	0.004
		53	29	0.008	0.008
		100	29	0.008	0.008
		200	29	0.008	0.008
	MPFI	24	29	$< 10^{-3}$	
		53	29	$< 10^{-3}$	
		100	29	$< 10^{-3}$	
		200	29	$< 10^{-3}$	

taken independently and then can lead to an unwanted over-estimation of the resulting interval. This is the case in Eq. (5.4) where the variable x_n appears twice whereas x_n appears only once in Eq. (5.5). As a remark, if $a = 3.57$, no computational zero has been detected using SAM. The run time reported in Table 4 has been measured for 1-million iterations. Using MPFI with $a = 3.57$, more than 15,000 bits are required to perform 1-million iterations.

Although SAM has been designed for arbitrary precision, it can be compared with CADNA if the chosen precision is 24 or 53 bits. The results are, in general, similar with CADNA in double precision (53 bits) and SAM with 53 bits. It is normal since the operations are correctly rounded with the same precision in both cases. When there is a difference, it is due to the fact that the exponent of the floating-point numbers is not limited in SAM, whereas it is limited to 1,023 in double precision. The same explanation applies for single precision and SAM with 24 bits (here the exponent in single precision is limited to 127).

Table 4 Logistic map: number N of iterations performed with SAM and MPFI, $x_{n+1} = -a(x_n - \frac{1}{2})^2 + \frac{a}{4}$ with $x_0 = 0.6$

a		# bits	N	Execution time (in s)	
3.57	SAM			without detection	with all detections
		24	$> 10^6$	10.017	20.357
		53	$> 10^6$	10.565	22.669
		100	$> 10^6$	12.665	28.706
		200	$> 10^6$	15.389	37.154
		2000	$> 10^6$	164.85	541.14
	MPFI	24	394	$< 10^{-3}$	
		53	1990	0.004	
		100	5044	0.016	
		200	11310	0.036	
		2000	123220	5.376	
		5 000	311 828	55.691	
		15 000	935 556	913.40	
		16 500	$> 10^6$	1064.9	
		50 000	$> 10^6$	5818.7	
3.575	SAM			without detection	with all detections
		24	110	0.008	0.012
		53	354	0.012	0.016
		100	722	0.016	0.032
		200	1548	0.032	0.068
		2000	15890	2.636	8.673
	MPFI	24	108	$< 10^{-3}$	
		53	324	$< 10^{-3}$	
		100	722	$< 10^{-3}$	
		200	1534	0.008	
		2000	15882	0.696	
3.6	SAM			without detection	with all detections
		24	56	$< 10^{-3}$	0.008
		53	152	0.008	0.012
		100	342	0.008	0.020
		200	722	0.020	0.036
	MPFI	24	56	$< 10^{-3}$	
		53	152	$< 10^{-3}$	
		100	338	$< 10^{-3}$	
		200	722	0.004	
3.7	SAM			without detection	with all detections
		24	39	0.004	0.008
		53	106	0.008	0.008
		100	198	0.012	0.012
		200	396	0.016	0.024
	MPFI	24	39	$< 10^{-3}$	
		53	103	$< 10^{-3}$	
		100	197	$< 10^{-3}$	
		200	389	0.004	
10	SAM			without detection	with all detections
		24	21	$< 10^{-3}$	0.008
		53	29	0.008	0.008
		100	29	0.012	0.012
		200	29	0.012	0.020
	MPFI	24	29	$< 10^{-3}$	
		53	29	$< 10^{-3}$	
		100	29	$< 10^{-3}$	
		200	29	$< 10^{-3}$	

5.5 Performance Test

The performance of SAM has been compared with that of MPFI. The matrix multiplication $M * M$, where $M_{i,j} = i + j - 1 (1 \leq i \leq N, 1 \leq j \leq N)$ has been computed using SAM and MPFI with different precisions and different values of N. Three types of executions have been performed using SAM:

- without the detection of numerical instabilities
- with the detection of instabilities that may occur in multiplications, divisions or calls to the power function and may invalidate the CESTAC method, as mentioned in Sect. 3; this detection enables a self-validation of the SAM library
- with the detection of all instabilities.

Table 5 Run time (in seconds) for a matrix multiplication of size 100

# Bits	24	53	100	500	1,000	5,000
MPFI	0.292	0.320	0.432	0.504	0.648	2.216
SAM without detection	0.892	0.936	1.076	1.120	1.372	2.616
SAM with self-validation	0.896	0.940	1.092	1.160	1.380	2.624
SAM with all detections	7.168	8.357	10.461	27.254	69.588	903.528
SAM with self-validation/MPFI	3.07	2.94	2.53	2.30	2.13	1.18

The run times reported in Table 5 have been measured for $N = 100$ on an Intel Q9550 2.83 GHz processor using the g++ 4.4 compiler.

From Table 5, it is noticeable that MPFI performs better than SAM. When self-validation is activated, the time ratio varies from 1.2 to 3.1, depending on the working precision. The cost of self-validation in SAM (like in CADNA) is negligible. The detection of all numerical instabilities is an interesting feature of SAM, unfortunately it may be very costly, depending on the application. This cost is mainly due to the detection of cancellations which may occur in additions or subtractions. From tests carried out for various values of N (from 50 to 1,000) it is noticeable that the time ratio between SAM and MPFI is independent of N. Anyway our main objective was to show that the resources required by SAM with self-validation are of the same order of magnitude as MPFI. This makes SAM useful for the validation of huge numerical codes.

6 Conclusion and Future Work

In this article, we have demonstrated that SAM can be very useful to study the behavior of chaotic dynamical system. We have only studied the behavior of the logistic map. We plan to do a similar work with other systems like the Hénon map [50] or the Lorenz attractor [51].

We have also compared in terms of efficiency and computing times the SAM library with MPFI. As mentioned previously, SAM and MPFI do not give the same answer since MPFI leads to a certified answer whereas SAM gives an answer true within a given probability. Anyway, it is sometimes sufficient to know the answer only with high probability. The main advantage of SAM is that it can be used to validate huge numerical scientific codes.

A perspective is the adaptive refinement of the precision in programs using SAM when the accuracy of the results is not satisfactory. Such a strategy would be based on an appropriate threshold for the number of exact significant digits lost during the computation.

Acknowledgments The authors sincerely wish to thank the reviewers for their constructive comments.

References

1. Li, X.S., Demmel, J.W., Bailey, D.H., Henry, G., Hida, Y., Iskandar, J., Kahan, W., Kang, S.Y., Kapur, A., Martin, M.C., Thompson, B.J., Tung, T., Yoo, D.J.: Design, implementation and testing of extended and mixed precision BLAS. ACM Trans. Math. Softw. **28**(2), 152–205 (2002)
2. Fousse, L., Hanrot, G., Lefèvre, V., Pélissier, P., Zimmermann, P.: MPFR: A multiple-precision binary floating-point library with correct rounding. ACM Trans. Math. Softw. **33**(2), 13:1–13:15 (2007). (http://www.mpfr.org)
3. Wilkinson, J.H.: Rounding errors in algebraic processes. Prentice-Hall Inc., Englewood Cliffs (1963)
4. Higham, N.J.: Accuracy and stability of numerical algorithms. 2nd edn. Society for Industrial and Applied Mathematics (SIAM), Philadelphia (2002)
5. Einarsson, B., et al.: Accuracy and Reliability in Scientific Computing. Software-Environments-Tools. SIAM, Philadelphia (2005)
6. Chaitin-Chatelin, F., Frayssé, V.: Lectures on Finite Precision Computations. Society for Industrial and Applied Mathematics, Philadelphia (1996)
7. Moore, R.: Interval analysis. Prentice Hall, Saddle River (1966)
8. Alefeld, G., Herzberger, J.: Introduction to interval analysis. Academic Press, New York (1983)

9. Moore, R., Kearfott, R., Cloud, M.: Introduction to interval analysis. Society for Industrial and Applied Mathematics, Philadelphia (2009)
10. Chesneaux, J.M.: L'arithmétique stochastique et le logiciel CADNA. Habilitation à diriger des recherches Université Pierre et Marie Curie, Paris (1995)
11. Vignes, J.: A stochastic arithmetic for reliable scientific computation. Math. Comput. Simul. **35**, 233–261 (1993)
12. Goubault, E., Putot, S., Baufreton, P., Gassino, J.: Static analysis of the accuracy in control systems: Principles and experiments. In: Proceedings of Formal Methods in Industrial Critical Systems, LNCS 4916, Springer, Berlin (2007)
13. Chesneaux, J.M., Graillat, S., Jézéquel, F.: Rounding Errors. In: Encyclopedia of Computer Science and Engineering, vol. 4, pp. 2480–2494. Wiley, New York (2009)
14. Revol, N., Rouillier, F.: MPFI (Multiple Precision Floating-point Interval library) (2009). (Available at http://gforge.inria.fr/projects/mpfi).
15. Bailey, D.H.: A Fortran 90-based multiprecision system. ACM Trans. Math. Softw. **21**(4), 379–387 (1995)
16. Brent, R.P.: A fortran multiple-precision arithmetic package. ACM Trans. Math. Softw. **4**(1), 57–70 (1978)
17. Priest, D.M.: Algorithms for arbitrary precision floating point arithmetic. In Kornerup, P., Matula, D.W., (eds.) Proceedings of the 10th IEEE Symposium on Computer Arithmetic (Arith-10), Grenoble, France, pp. 132–144. IEEE Computer Society Press, Los Alamitos (1991)
18. Shewchuk, J.R.: Adaptive precision floating-point arithmetic and fast robust geometric predicates. Discret. Comput. Geom. **18**(3), 305–363 (1997)
19. Bailey, D.H.: A Fortran-90 double-double library (2001). (Available at http://crd.lbl.gov/~dhbailey/mpdist/index.html)
20. Hida, Y., Li, X.S., Bailey, D.H.: Algorithms for quad-double precision floating point arithmetic. In: Proceedings of 15th IEEE Symposium on Computer Arithmetic, pp. 155–162. IEEE Computer Society Press, Los Alamitos (2001)
21. IEEE Computer Society, New York: IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard, pp. 754–1985 (1985). (Reprinted in SIGPLAN Notices **22**(2), 9–25 (1987))
22. Grandlund, T.: GNU MP: The GNU Multiple Precision Arithmetic Library. (http://gmplib.org)
23. Kulisch, U.: Advanced Arithmetic for the Digital Computer. Springer, Wien (2002)
24. Chesneaux, J.M.: Study of the computing accuracy by using probabilistic approach. In: Ullrich, C. (ed.) Contribution to Computer Arithmetic and Self-Validating Numerical Methods, pp. 19–30. IMACS, New Brunswick (1990)
25. Chesneaux, J.M., Vignes, J.: Sur la robustesse de la méthode CESTAC. C. R. Acad. Sci. Paris Sér. I Math. **307**, 855–860 (1988)
26. Vignes, J.: Zéro mathématique et zéro informatique. C. R. Acad. Sci. Paris Sér. I Math. **303**, 997–1000 (1986)
27. Vignes, J.: Zéro mathématique et zéro informatique. La Vie des Sciences, **4**(1), 1–13, (1987)
28. Université Pierre et Marie Curie, Paris, F.: CADNA: Control of Accuracy and Debugging for Numerical Applications. (http://www.lip6.fr/cadna)
29. Jézéquel, F., Chesneaux, J.M.: CADNA: a library for estimating round-off error propagation. Comput. Phys. Commun. **178**(12), 933–955 (2008)
30. Jézéquel, F., Chesneaux, J.M., Lamotte, J.L.: A new version of the CADNA library for estimating round-off error propagation in Fortran programs. Comput. Phys. Commun. **181**(11), 1927–1928 (2010)
31. Lamotte, J.L., Chesneaux, J.M., Jézéquel, F.: CADNA_C: A version of CADNA for use with C or C++ programs. Comput. Phys. Commun. **181**(11), 1925–1926 (2010)
32. Chesneaux, J.M., Troff, B. : Computational stability study using the CADNA software applied to the Navier-Stokes solver PEGASE. In: Alefeld, G., Frommer, A. (eds.) Scientific Computing and Validated Numerics, pp. 84–90. Akademie, Berlin (1996)
33. Alberstein, N., Chesneaux, J.M., Christiansen, S., Wirgin, A.: Comparison of four software packages applied to a scattering problem. Math. Comput. Simul. **48**, 307–318 (1999)
34. Jézéquel, F., Rico, F., Chesneaux, J.M., Charikhi, M.: Reliable computation of a multiple integral involved in the neutron star theory. Math. Comput. Simul. **71**(1), 44–61 (2006)
35. Scott, N., Jézéquel, F., Denis, C., Chesneaux, J.M.: Numerical 'health check' for scientific codes: the CADNA approach. Comput. Phys. Commun. **176**(8), 507–521 (2007)
36. Scott, N., Faro-Maza, V., Scott, M., Harmer, T., Chesneaux, J.M., Denis, C., Jézéquel, F.: E-collisions using e-science. Phys. Part. Nuclei Lett. **5**(3), 150–156 (2008)
37. Rump, S.: Reliability in Computing. The Role of Interval Methods in Scientific Computing. Academic Press, Oakville (1988)
38. Muller, J.M.: Arithmétique des Ordinateurs. , Masson (1989)
39. Chesneaux, J.M., Jézéquel, F.: Dynamical control of computations using the trapezoidal and Simpson's rules. J. Univers. Comput. Sci. **4**(1), 2–10 (1998)
40. Jézéquel, F.: Dynamical control of converging sequences computation. Appl. Numer. Math. **50**(2), 147–164 (2004)
41. Jézéquel, F., Chesneaux, J.M.: Computation of an infinite integral using Romberg's method. Num. Algo. **36**(3), 265–283 (2004)
42. Jézéquel, F.: A dynamical strategy for approximation methods. C. R. Acad. Sci. Paris Mécanique **334**, 362–367 (2006)
43. Ahmed, Z.: Definitely an integral. Am Math. Month. **109**(7), 670–671 (2002)
44. Bailey, D., Li, X.: A comparison of three high-precision quadrature schemes. In: Proceedings of 5th Real Numbers and Computers conference, Lyon, France, pp. 81–95 (2003)
45. Devaney, R.L.: An introduction to chaotic dynamical systems. Second edn. Addison-Wesley Studies in Nonlinearity, Addison-Wesley Publishing Company Advanced Book Program, Redwood City (1989)

46. Argyris, J., Faust, G., Haase, M.: An exploration of chaos. Texts on Computational Mechanics, vol. VII. North-Holland Publishing Co., Amsterdam (1994)
47. Tucker, W.: The Lorenz attractor exists. C. R. Acad. Sci. Paris Sér. I Math. **328**(12), 1197–1202 (1999)
48. Galias, Z., Tucker, W.: Rigorous study of short periodic orbits for the Lorenz system. In: Proceedings of IEEE International Symposium on Circuits and Systems, pp. 764–767. ISCAS'08, Seattle (2008)
49. Tucker, W.: Fundamentals of chaos. In: Kocarev, L., et al. (eds.) Intelligent computing based on chaos. Studies in Computational Intelligence, vol. 184, pp. 1–23. Springer, Berlin (2009)
50. Pichat, M., Vignes, J.: The numerical study of chaotic systems—future and past. In: 16th IMACS World Congress on Scientific Computation, Applied Mathematics and Simulation. Lausanne, Switzerland (2000)
51. Yao, L.S.: Computed chaos or numerical errors. Nonlinear Anal. Model. Contr. **15**(1), 109–126 (2010)