CADNA: a library for estimating round-off error propagation

Fabienne Jézéquel [1], Jean-Marie Chesneaux

UPMC Univ. Paris 06
UMR 7606
Laboratoire d'Informatique de Paris 6
4 place Jussieu, F-75005 Paris, France
{*Jean-Marie.Chesneaux, Fabienne.Jezequel*}*@lip6.fr*

Abstract

The CADNA library enables one to estimate round-off error propagation using a probabilistic approach. With CADNA the numerical quality of any simulation program can be controlled. Furthermore by detecting all the instabilities which may occur at run time, a numerical debugging of the user code can be performed. CADNA provides new numerical types on which round-off errors can be estimated. Slight modifications are required to control a code with CADNA, mainly changes in variable declarations, input and output. This paper describes the features of the CADNA library and shows how to interpret the information it provides concerning round-off error propagation in a code.

PACS: 02.70.-c

Key words: CADNA; CESTAC method; Discrete Stochastic Arithmetic; floating-point arithmetic; numerical validation; round-off errors.

PROGRAM SUMMARY

Manuscript Title: CADNA: a library for estimating round-off error propagation
Authors: Fabienne Jézéquel, Jean-Marie Chesneaux
Program Title: CADNA
Journal Reference:
Catalogue identifier:
Licensing provisions: none
Programming language: Fortran
Computer: PC running LINUX with an i686 or an ia64 processor, UNIX workstations including SUN, IBM.

[1] Corresponding author

Operating system: LINUX, UNIX
Keywords: CADNA; CESTAC method; Discrete Stochastic Arithmetic; floating-point arithmetic; numerical validation; round-off errors.
PACS: 02.70.-c
Classification: 6.5 Software including Parallel Algorithms
Nature of problem:
A simulation program which uses floating-point arithmetic generates round-off errors, due to the rounding performed at each assignment and at each arithmetic operation. Round-off error propagation may invalidate the result of a program. The CADNA library enables one to estimate round-off error propagation in any simulation program and to detect all numerical instabilities that may occur at run time.
Solution method:
The CADNA library [1] implements Discrete Stochastic Arithmetic [2-4] which is based on a probabilistic model of round-off errors. The program is run several times with a random rounding mode generating different results each time. From this set of results, CADNA estimates the number of exact significant digits in the result that would have been computed with standard floating-point arithmetic.
Restrictions:
CADNA requires a Fortran 90 (or newer) compiler. In the program to be linked with the CADNA library, round-off errors on complex variables cannot be estimated. Furthermore array functions such as product or sum must not be used. Only the arithmetic operators and the abs, min, max and sqrt functions can be used for arrays.
Running time:
The version of a code which uses CADNA runs at least three times slower than its floating-point version. This cost depends on the computer architecture and can be higher if the detection of numerical instabilities is enabled. In this case, the cost may be related to the number of instabilities detected.

References:

[1] The CADNA library, URL address: http://www.lip6.fr/cadna

[2] J.-M. Chesneaux, L'arithmétique Stochastique et le Logiciel CADNA, Habilitation à diriger des recherches, Université Pierre et Marie Curie, Paris, 1995.

[3] J. Vignes, A stochastic arithmetic for reliable scientific computation, *Math. and Comp. in Sim.* 35, 1993, pp. 233-261.

[4] J. Vignes, Discrete Stochastic Arithmetic for Validating Results of Numerical Software, *Num. Algo.* 37, 2004, pp. 377-390.

LONG WRITE-UP

1 Introduction

Floating-point arithmetic only approximates exact arithmetic. So, when a scientific code is run on a computer its results are not exact; the approximation introduces a round-off error for each arithmetic operation, as does the assignment statement (because registers have more digits than memory words), when the value cannot be coded exactly.

As the terms *precision* and *accuracy* are widely used throughout, they need to be clearly defined. *Precision* refers to the number of bits used for the representation of floating-point numbers. For instance, in the IEEE 754 standard [1], single precision and double precision variables are encoded respectively on 32 and 64 bits. *Accuracy* denotes the closeness of a computed result to the exact result. The accuracy of a computed result may be affected by a truncation error inherent to the approximation method used and by a round-off error due to the finite precision of the arithmetic used.

CADNA (Control of Accuracy and Debugging for Numerical Applications) [2,3] is a library which allows, during the execution of a code:

- the estimation of the error due to round-off error propagation,
- the detection of numerical instabilities,
- the checking of the sequencing of the program (tests and branchings),
- the estimation of the accuracy of all intermediate computations.

CADNA is based on the CESTAC method [4–6] which studies round-off error propagation from a stochastic point of view. The basic idea is to use a random rounding to obtain several samples of each result of any arithmetic operation. The number of common digits in these samples estimates the number of exacty significant digits in the floating-point result. So the deterministic arithmetic of the computer is replaced by the so-called Discrete Stochastic Arithmetic (DSA) [3].

Validation of numerical results, which is a real problem for scientific computing, can therefore be carried out using the CADNA library. For instance, round-off error propagation in 2DRMP, a suite of two-dimensional R-matrix propagation programs, has been studied using CADNA [7]. Although programs written in ADA, C or Fortran can be controlled using the CADNA library, this paper focuses on the Fortran version of CADNA. The following section is a reference guide that describes types, subroutines and functions that compose the CADNA library. Section 3 is a user's guide that describes

step by step how to (slightly) modify a source code to use the Discrete Stochastic Arithmetic implemented in the library. Section 4 gives instructions for the installation of CADNA and describes how to test the library. Section 5 comments on the results of the test programs. Finally, the software structure of the library is presented in Section 6.

2 The CADNA library

2.1 The Discrete Stochastic Arithmetic and its implementation

The CADNA library implements Discrete Stochastic Arithmetic (DSA) which is based on the CESTAC method. The principles of the CESTAC method are first briefly described.

The CESTAC method [4–6] consists in running the same code N times with a different round-off error propagation for each execution. The round-off error on the final floating-point result is estimated from the different computed results R_i $(i = 1, ..., N)$. The different round-off error propagations are obtained by using the random rounding mode, which consists in choosing with an equal probability, for each operation, the result rounded up or down. In practice, the rounding mode is randomly set to rounding towards $-\infty$ or $+\infty$.

It has been proved [4] that a computed result R is modelled to the first order in 2^{-p} as:

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

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 equation (1), we deduce that:

(1) the mean value of the random variable Z is the exact result r,
(2) under some assumptions, 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 determine the accuracy of R. Thus from N samples R_i $(i = 1, ..., N)$ the number of significant decimal digits common to

$\overline{R}$ and r can be estimated with the following equation.

$$C_{\overline{R}} = \log_{10}\left(\frac{\sqrt{N}\left|\overline{R}\right|}{\sigma\tau_\beta}\right), \tag{2}$$

where

$$\overline{R} = \frac{1}{N}\sum_{i=1}^{N} R_i \quad \text{and} \quad \sigma^2 = \frac{1}{N-1}\sum_{i=1}^{N}\left(R_i - \overline{R}\right)^2. \tag{3}$$

τ_β is the value of Student's distribution for $N-1$ degrees of freedom and a probability level $1-\beta$. In practice $N = 3$, $\beta = 0.05$ and then $\tau_\beta = 4.4303$. Thus with the CESTAC method, the mean value $\overline{R}$ is chosen as the computed result and its number of exact significant decimal digits is estimated with equation (2).

The CESTAC method is based on a first order model: the terms in 2^{-2p} (p being the number of bits of the mantissa) which appear in the expression of the round-off error due to multiplications and divisions have been neglected. Only the terms in 2^{-p} are considered. It has been shown [4,2] that, if a computed result becomes non-significant, *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 dynamic 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 samples R_i, and also to the concept of computational zero, also named informatical zero [8].

Definition 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 non-significant. A computational zero is a value that cannot be differentiated from the mathematical zero because of its round-off error. From this concept, discrete stochastic relations have been defined.

Definition 2 *Let X and Y be N-samples provided by CESTAC method.*

- *Discrete stochastic equality denoted by $s=$ is defined as:*
  $X \, s= \, Y \quad if \quad X - Y = @.0$

- *Discrete stochastic inequalities denoted by $s>$ and $s\geq$ are defined as:*
  $Xs>Y \quad if \quad \overline{X}>\overline{Y}$ *and* $X-Y\neq @.0$
  $Xs\geq Y \quad if \quad \overline{X}\geq\overline{Y}$ *or* $X-Y=@.0$.

Discrete Stochastic Arithmetic (DSA) [2,3] is the association of the synchronous implementation of the CESTAC method, the concept of computational zero and the discrete stochastic relations. Elements of the DSA, which are named *stochastic numbers*, are N-sets provided by the CESTAC method. Their number of exact significant digits can be estimated from equation (2). Therefore, with the CADNA library, which implements the DSA, one can estimate the impact of round-off errors on any result of a scientific code and also check that no anomaly occurred during the run, especially in branching statements.

No notable change is required in a code to be run with CADNA. Indeed CADNA has been written in Fortran 90 language, which enables one to create new numerical types and to overload the arithmetic operators for these types. All the arithmetic operators have been overloaded for stochastic numbers, also for stochastic arrays of rank 1, in such a manner that when an operator is used, the operands are N-sets and the returned result is a randomly perturbed N-set. The relational operators have been overloaded in accordance with the discrete stochastic relations. The standard functions defined in Fortran (sin, cos, exp, ...) have also been overloaded. Likewise, input/output statements have been modified, mainly the printing statement which gives as a result the mean value of the N-set written with only its exact significant digits. Furthermore, in order to enable the evaluation of the weight of uncertainties on initial data on the results, a function called data_st may be used to perturb data as illustrated in 2.5.7.

During the run of a program, as soon as a numerical anomaly (for example the product of non-significant numbers, or a relational test involving a non-significant result) is produced, some special counters are updated. At the end of the run, all information about numerical anomalies are printed on the standard output. If no anomaly has been detected, it means that the program runs without any numerical problem. Results are then given with their accuracy (number of exact significant digits). If some numerical anomalies have been detected, they must be analysed. Helped by the debugger associated with the compiler, the user may retrieve the statements that produced the anomalies and determine if changes in the code are required.

The stochastic types and the overloaded or newly defined functions of the library are presented in the next subsections.

2.2 Stochastic types

In this version, CADNA provides two new numerical types, the *stochastic types*:

type (single_st)	for stochastic variables in single precision
	stochastic type associated with real
type (double_st)	for stochastic variables in double precision
	stochastic type associated with double precision

2.3 Intrinsic functions

We present here how the intrinsic functions defined in Fortran have been extended to stochastic types.

2.3.1 Conversion functions

The int and nint functions:
They take a parameter of stochastic type and return an integer. The knowledge of the accuracy is lost. If X is a stochastic variable consisting in N samples X_i,

- int(X) is computed as $\mathrm{int}(\frac{\sum_{i=1}^{N} X_i}{N})$
- nint(X) is computed as $\mathrm{nint}(\frac{\sum_{i=1}^{N} X_i}{N})$.

The aint and anint functions:
They take a parameter of stochastic type. The return value has the same type as the input parameter. The control of the accuracy is preserved. If X (respectively Y) is a stochastic variable consisting in N samples X_i (respectively Y_i),

- the statement Y=aint(X) is equivalent to $Y_i = \mathrm{aint}(X_i),\ i = 1, ..., N$
- the statement Y=anint(X) is equivalent to $Y_i = \mathrm{anint}(X_i),\ i = 1, ..., N$.

The real function:
It takes a parameter of stochastic type and returns a value of type single_st.

The dble function:
It takes a parameter of stochastic type and returns a value of type double_st.

2.3.2 Numerical functions

The aimag and conjg functions:
These functions are not overloaded, since this version of CADNA has no stochastic type corresponding to the standard complex type.

The abs function:
Given a single_st argument, this function returns a positive single_st value. Given a double_st argument, it returns a positive double_st value. It accepts an array of stochastic numbers of rank 1 as an argument.

The min and max functions:
The min and max functions have been extended in a more restricted way than the previous functions: if they contain a stochastic argument, they *must only have* two *arguments* and these two arguments must have the same precision (single or double). So if there is a stochastic argument, the only (unordered) possible type couples are (real, single_st), (single_st, single_st), (double precision, double_st), (double_st, double_st).

The following table gives some examples of correct and wrong calls.

min(S,D)	S of single_st type D of double_st type	incorrect
min(S,XD)	S of single_st type XD of double precision or double_st type	incorrect
min(S1,S2)	S1 and S2 of single_st type	correct
min(S,XS)	S of single_st type XS of real type	correct
min(D1,D2)	D1 and D2 of double_st type	correct
min(D,XD)	D of double_st type XD of double precision type	correct

The min and max functions accept arrays of stochastic numbers of rank 1 as arguments with the same rules as for scalar arguments.

The sign, mod and dim functions:
These functions accept stochastic arguments. The rules are the same as for the min and max functions. No stochastic array is accepted.

2.3.3 Mathematical functions

These are the following functions:
**, sqrt, exp, log, log10, sin, cos, tan, asin, acos, atan, atan2, sinh, cosh, tanh.
They accept parameters of single_st or double_st stochastic type. The return value has the same type as the input parameter.

For the ** operator, the rules for arguments are the same as for the min and max functions. No stochastic array is accepted.

The sqrt function accepts a stochastic array of rank 1 as an argument.

Mathematical functions are defined, for stochastic types, by their generic name only (for instance there is no stochastic version of dsin).

2.4 Relational operators

Comparison operators are overloaded and accept stochastic types and a mixture of standard real or integer types with different precisions. They take into account the accuracy of the operands.

Thus when the expression a .EQ. 0. is true, it means that a is a computational zero, *i.e.* a is a mathematical zero or a has no exact significant digit.

Similarly, when the expression a .GT. b is true, it means that a-b is not a computational zero (*i.e.* has at least one exact significant digit) and $\sum_{i=1}^{N} a_i > \sum_{i=1}^{N} b_i$.

2.5 CADNA specific functions

The previous part described how some standard Fortran statements are slightly affected when using the CADNA tool. Now we present functions that are specific to the library. Note that the subroutines cadna_init, cadna_end and str have to appear, respectively to initialize the library, to close the library and to print the results with their accuracy. The other functions nb_significant_digit, computed_zero, old_type, cadna_enable, cadna_disable and data_st will appear in some applications.

2.5.1 Initializing and closing the library

The cadna_init subroutine has to be called once, early in the main program. This subroutine has four integer arguments:

cadna_init(numb_instability, cadna_instability, cancel_level, init_random).

With the first argument which must always be present, the user chooses the maximum number of numerical instabilities that will be detected.

- if numb_instability $= -1$, all the instabilities will be detected
- if numb_instability $= 0$, no instability will be detected
- if numb_instability $=$ M (strictly positive M), the M first instabilities will be detected.

The other arguments are optional.

The second argument allows the user to determine what kind of instabilities will be enabled or disabled. There are 8 integer parameters in the library:
cadna_branching,
cadna_mathematic,
cadna_intrinsic,
cadna_cancellation,
cadna_division,
cadna_power,
cadna_multiplication,
cadna_all.

By default, the detection of all types of instability is enabled. The user has only to specify what kind of instability is to be disabled by passing, as the second argument, the addition of the chosen parameters. cadna_all disables all the detections of instabilities.

For instance, with the statement
call cadna_init(-1, cadna_branching)
the detection of unstable tests is disabled.
With the statement
call cadna_init(-1, cadna_mathematic + cadna_intrinsic)
the detection of instabilities due to mathematical functions and Fortran intrinsic functions is disabled.

The third argument corresponds to the following. An unstable cancellation is pointed out when the difference between the number of exact significant digits (*i.e.* digits which are not affected by round-off errors) of the result of an addition or a subtraction and the minimimum of the number of exact significant digits of the two operands is greater than the cancel_level argument. The default value of this argument is 4. In other words, when one loses more than cancel_level significant digits in one addition or subtraction, CADNA considers that a catastrophic cancellation has been detected (if the detection of this kind of instability is enabled).

The last argument is an integer which is used to initialize some internal variables for random arithmetic. The default value for this argument is 51.

The cadna_end subroutine "closes" the library and prints to the standard output the result of the detection of numerical instabilities.

2.5.2 Obtaining a string from a result with its evaluated accuracy

The str function has a stochastic argument and returns a string containing the scientific notation of this argument; only the exact significant digits appear in the string. Thus accuracy is easy to read. *Note that there is no guarantee of the last digit provided by the* str *function.* When the argument has no exact significant digit, the string that is returned is @.0. The field width of the output string is 14 for a single_st variable and 23 for a double_st variable. The following instructions:

```
type (single_st) :: X,Y,Z
...
write(*,*) 'X = ',str(X)
write(*,*) 'Y = ',str(Y)
write(*,*) 'Z = ',str(Z)
```

may yield for instance:

X = 0.123456E+00	(6 exact significant digits)
Y = 0.123E+00	(3 exact significant digits)
Z = @.0	(no exact significant digit, computational zero)

2.5.3 Obtaining the number of exact significant digits of a stochastic variable

The nb_significant_digit function has a stochastic argument and returns an integer giving the number of exact significant decimal digits of this argument when the function is called.

At some point nb_significant_digit(x) may return 7; later during the run it may return 5. If x becomes non-significant then nb_significant_digit(x) returns 0.

2.5.4 Testing if a variable is a computational zero

The computed_zero function has one stochastic argument and returns a logical value. The computed_zero function returns TRUE if its argument is a computational zero, *i.e.* its argument is a mathematical zero or has no exact significant digit.

2.5.5 Obtaining a standard value from a stochastic variable

The old_type function has a stochastic (single_st or double_st) argument and returns a value of the associated standard type (real or double precision). The output value is the mean value of the N samples. Obviously, for the statement
y = old_type(x)
where y is real and x is single_st any information on the accuracy of x is lost when using y.

2.5.6 Enabling and disabling the detection of instabilities

The cadna_enable and cadna_disable subroutines are used respectively to enable and disable the detection of one kind of instability. Each of these subroutines has one integer argument, which may be one of the seven following integer parameters defined in the CADNA library:
cadna_branching,
cadna_mathematic,
cadna_intrinsic,
cadna_cancellation,
cadna_division,
cadna_power,
cadna_multiplication.

2.5.7 Reducing accuracy of initial data

Initial data are read by the generic function read as for standard types. By the implicit conversion formerly described, they are stored into stochastic variables having the maximal accuracy represented by N equal samples. These data are often known with less significant digits than provided by their internal representation. The data_st function allows the user to introduce some effective uncertainties on these data, reducing their initial accuracy. So the accuracy of results depends in some way on the accuracy of initial data.

The data_st subroutine has three arguments: call data_st(X,ERX, IER).
The first argument is stochastic and must be present.
The second one is an optional real argument that contains the relative or

absolute uncertainty of the first one. The last argument determines the kind of the uncertainty: relative or absolute.

If X is a stochastic variable and ERX is a real value strictly less than 1, the call data_st(X,ERX, IER) statement modifies the values of the N samples in X according to the following formula:

$$X_i = X_i * (1 + ERX * ALEA) \text{ for } i = 1 \text{ to } N \text{ if IER} = 0$$

$$X_i = X_i + ERX * ALEA \text{ for } i = 1 \text{ to } N \text{ if IER} = 1$$

$ALEA$ is a random variable uniformly distributed between -1 and 1.
If ERX is 0, no perturbation takes place as if the statement was suppressed.
If ERX is absent, perturbation will concern only the last bit of the mantissa.
If IER is absent, it is like $IER = 0$. The data_st subroutine without ERX must be used when data are considered as exact but cannot be exactly coded in the memory.

3 CADNA user guide

The use of the CADNA library involves seven steps:

- declaration of the CADNA library for the compiler,
- initialization of the CADNA library,
- substitution of the type REAL or DOUBLE PRECISION by stochastic types in variable declarations,
- possible changes in the input data if perturbation is desired, to take into account uncertainty in initial values,
- change of output statements to print stochastic results with their accuracy,
- possible use of CADNA functions to evaluate the number of exact significant digits or access the current ordinary value (losing knowledge of current accuracy)
- termination of the CADNA library.

Subsections 3.1 to 3.10 describe the necessary changes for a Fortran source code to be compiled with the CADNA library. Subsection 3.11 is devoted to the dynamical numerical debugging that CADNA allows.

3.1 Declaration of the CADNA library

The use CADNA pseudo-statement must be placed before any declaration of stochastic variables, in order for stochastic types and overloaded or new func-

tions or subroutines to be found by the compiler. As usual in a Fortran 90 source code, this statement must be added:

- after one among the following lines
  - PROGRAM that begins an application
  - MODULE that begins a module
  - SUBROUTINE if it begins an "isolated subroutine", *i.e.* a subroutine that is not declared into the scope of a program or a module declaration
  - FUNCTION if it begins an "isolated function", *i.e.* a function that is not declared into the scope of a program or a module declaration
- before any declaration.

3.2 Initialization and termination of the CADNA library

The call to the cadna_init subroutine must be inserted immediately after the main program declaration statements to initialize the random arithmetic. For more information about the arguments of the cadna_init subroutine, see 2.5.1.

The call to the cadna_end subroutine must be the last executed program statement. The cadna_end subroutine writes on the standard output a report on the numerical stability of the run.

3.3 Declaration of variables

3.3.1 Changes in the type of variables

To control the numerical quality of a variable, its standard type must be replaced by the associated stochastic type.

Example:

standard declaration	CADNA declaration
real :: a,b	**type (single_st)** :: a,b
double :: c	**type (double_st)** :: c
real, dimension(6) :: d,e,f	**type (single_st)**, dimension(6) :: d,e,f

When the real declaration is implicit, this general declaration should be added: implicit type (single_st) (A-H,O-Z)

3.3.2 Changes in the name of some variables

In a Fortran program, the name of a variable can be the name of an intrinsic function. For instance in a Fortran source code, such a declaration is valid:
integer :: nint, dim, max

Intrinsic functions are overloaded in the CADNA library. Therefore it is not allowed anymore and, if the name of a variable is also the name of an intrinsic function, it must be changed in the entire source code.

3.4 DATA for initializing stochastic variables

As previously described, each stochastic variable is represented by N different variables of the standard associated type. So initializing stochastic variables in DATA sections is possible by writing N occurrences of the standard initial value.

Example:

standard declaration	CADNA declaration
real :: d data d/3.245/	**type (single_st)** :: d data **d%x, d%y, d%z /3*3.245/**

3.5 Changes in assignments or arithmetic operations

3.5.1 Conversions between usual types and stochastic types

In assignment statements, conversions are implicit from Fortran real, integer or double precision types *to* and *from* stochastic types (because the = operator has been overloaded), but for conversions from stochastic types to standard types, the knowledge of accuracy is lost.

An immediate conversion from a stochastic type to the corresponding standard type may also be performed using the old_type function, which also loses any knowledge of accuracy.

When a variable is set to a value which can not be exactly coded on computer, the data_st function must be used.

Example:

Initial Fortran statements	Modified statements for CADNA
	use cadna
real :: x	**type (single_st)** :: x
	call cadna_init(-1)
x=1.234	x=1.234
	call data_st(x)

3.5.2 Standard arithmetic operators

As previously described, all arithmetic operators on floating-point variables are overloaded and arithmetic expressions without functions do not have to be modified. Expressions may contain a mixture of stochastic types, standard types and integer types.

With the following declarations:
type (single_st) :: a,b
type (double_st) :: c
the statement c = a * a + b * 3 needs no change.

The result of expressions containing stochastic terms will be of stochastic type. As for standard types, double_st prevails over single_st.
So with the previous declarations, c = a * c + b * 3 needs no change.

3.5.3 Vector operators and functions

In Fortran 90, some arithmetic operators and intrinsic functions are generalized to act on arrays. In this version of CADNA, only arithmetic operators and the abs, min, max, sqrt functions are overloaded for stochastic arrays. Standard or stochastic scalar types may be mixed with stochastic arrays if they have the same precision. The assignment statement has also been overloaded for stochastic arrays with the same rules.

3.6 Changes in reading statements

The generic function read is adapted to standard floating-point variables, which must be transformed into stochastic variables.

Example:

Initial Fortran statements	Modified statements for CADNA
	use cadna
real :: x	**real :: xaux**
	type (single_st) :: x
	call cadna_init(-1)
.....	.....
read (5,*) x	read (5,*) **xaux**
	x=xaux

3.7 Changes in printing statements

Before printing each stochastic variable, it must be transformed in a string by the str function. The required length is 14 for a single_st variable and 23 for a double_st variable. Therefore formats should be modified.

For example, if a real variable x becomes a single_st variable, the printing instruction can be modified as follows:

Initial Fortran statements	Modified statements for CADNA
	use cadna
real :: x	**type (single_st)** :: x
	call cadna_init(-1)
...	...
write(6,100) x	write(6,100) **str(x)**
100 format(1x,'x = ',f8.3)	100 format(1x,'x = ',**a14**)

If standard formats (write(*,*)...) are used, the only change is the use of the str function.

3.8 Changes in intrinsic functions

3.8.1 Changes of non-generic names

For each intrinsic function, only its generic version has been overloaded in the CADNA library. So each intrinsic function that is not generic must be replaced by its generic name. For instance, any call to the alog function must be replaced by a call to the log function.

3.8.2 Changes in the call of min and max functions

With CADNA, the min and max functions must have two arguments (for more details, see 2.3). Consequently a call to the min or max function with more than two arguments must be changed.

Example:

Initial Fortran statements	Modified statements for CADNA
	use cadna
real :: a,b,c,d	**type (single_st)** :: a,b,c,d
	call cadna_init(-1)
...	...
d=max(a,b,c)	d=max(**max(a,b)**,c)

3.8.3 Suppression of intrinsic functions declarations

Intrinsic functions are sometimes declared as in the following example:
intrinsic max, abs
As intrinsic functions are overloaded in the CADNA library, they are no longer intrinsic. Therefore such declarations must be removed from the original source code.

3.9 Changes in statement functions

Statement functions are defined with the "=" operator. With CADNA such functions must be written in a standard way, because the "=" operator has not been overloaded for the definition of statement functions.

Example:

Initial Fortran statements	Modified statements for CADNA
	use cadna
real :: f,x,y,a	**type (single_st)** :: f
f(x,y) = x + a*y	**call cadna_init(-1)**
a=2./3.	
.....	.....
	function f(x,y)
	use cadna
	type (single_st) :: f,x,y,a
	a=2./3.
	call data_st(a)
	f = x + a*y
	return
	end function f

3.10 Constants passed as function arguments

Function definitions and function calls must sometimes be adapted because stochastic parameters of functions must not be passed by value.

Example:

Initial Fortran statements	Modified statements for CADNA
	use cadna
real :: f, a	**type (single_st) :: aux,** f, a
	call cadna_init(-1)
	aux=2.
a=3.14*f(2.)	a=3.14*f(**aux**)
...	...
function f(x)	function f(x)
	use cadna
real :: f, x	**type (single_st)** :: f, x
...	...
end function f	end function f

3.11 Numerical debugging with CADNA

One can enable the detection of the following instabilities:
UNSTABLE DIVISION(S),
UNSTABLE POWER FUNCTION(S),
UNSTABLE MULTIPLICATION(S),
UNSTABLE BRANCHING(S),
UNSTABLE MATHEMATICAL FUNCTION(S),
UNSTABLE INTRINSIC FUNCTION(S),
UNSTABLE CANCELLATION(S).

The library counts the number of detections for each instability. The global information for these detections is printed out by the cadna_end subroutine, see 2.5.1.

The accuracy estimated by CADNA is valid if there is no deep numerical anomaly during the computation, *i.e.* no UNSTABLE DIVISION, UNSTABLE POWER FUNCTION or UNSTABLE MULTIPLICATION, see [2,4].

The meaning of the message is:

- **unstable division:** the divisor is non-significant
- **unstable power function:** one operand of the ** operator is non-significant
- **unstable multiplication:** both operands are non-significant
- **unstable branching:** the difference between the two operands is non-significant (a computational zero). The chosen branching statement is associated with the equality
- **unstable mathematical function:** in the LOG, SQRT, EXP or LOG10 function, the argument is non-significant.
- **unstable intrinsic function**:

(1) **in the INT or NINT function**: the function INT (or NINT) returns different values for each component of the stochastic argument.
(2) **in the ABS function**: the argument is non-significant.
(3) **in the SIGN or MOD function**: the second argument is non-significant.

- **unstable cancellation**: as explained in 2.5.1, an unstable cancellation is pointed out when the difference between the number of exact significant digits (*i.e.* digits which are not affected by round-off errors) of the result of an addition or a subtraction and the minimimum of the number of exact significant digits of the two operands is greater than the cancel_level argument. The default value of this argument is 4. In other words, when one loses more than cancel_level significant digits in one addition or subtraction, CADNA considers that a catastrophic cancellation has been detected (if the detection of this kind of instability is enabled).

To perform actual numerical debugging, it is necessary, for each instability, to identify the statement in the code that generates this instability. This can be performed directly using a symbolic debugger like **gdb** with Linux or as a background task using special input and output files. In both cases, one has to put a breakpoint at the entry of the instability internal function of the CADNA library. This function is called each time a numerical instability is detected. To get the right label for this system and compiler dependent function, one can use the following statement:

nm *name_of_the_binary_code* | grep instability

For instance, using **gdb** with Linux, the general statement which enables the detection of all the instabilities in a single run is

nohup gdb *name_of_the_binary_code* < gdb.in > gdb.out &

nohup allows to keep the process alive even when logging off.

The *gdb.in* file may contain

```
break instability_
```

```
run
while 1
where
cont
end
```

where prints out the complete trace of the instability which has stopped the run and **cont** makes the execution going on. The *gdb.out* file will contain all the traces of instabilities.

Although numerical debugging is intended to be performed on rather long source codes, it is illustrated in 5.6 with an experiment carried out using a simple Fortran source code.

4 Installation instructions

All installation instructions can be found in the INSTALL file of the package.

You first need to configure the installation by typing in the directory which contains the CADNA package

```
./configure
```

You may use options, such as FC to specify the name of the FORTRAN compiler or prefix to specify (with an absolute path) which directory will contain the compiled library. For instance, you may choose the following options:

```
./configure FC=g95 --prefix=/home/jmc/cadna_bin
```

To compile the library, type

```
make
```

Then to install the CADNA library in the directory specified with the prefix option, type

```
make install
```

Finally, to compile and execute the seven examples presented in the following section, just type

```
cd examples
make clean
make
```

5 Test runs

We present, with the seven examples included in the distribution, an illustration of the use of the CADNA library and the benefits of the DSA. For each example, we describe the results obtained using the standard floating-point arithmetic and then the results provided by the CADNA library.

The results reported in this section have been obtained using the g95 (version 0.9) Fortran compiler on a Pentium M processor running Linux. Different results may be obtained with another processor or another compiler, especially when the digits printed out using the standard floating-point arithmetic are affected by round-off errors. With CADNA, as results are printed out using the str function, only their exact significant digits appear. We recall that there is no guarantee of the last digit provided by the str function.

5.1 Example 1: a rational fraction function of two variables

In the following example [9], the rational fraction

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

is computed with $x = 77617$, $y = 33096$. The 15 first digits of the exact result are -0.827396059946821.

Using IEEE double precision arithmetic with rounding to the nearest, one obtains: res = 5.764607523034235E+17 and using CADNA in double precision, one obtains:

```
-----------------------------------------------------
CADNA 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
-----------------------------------------------------
CADNA software --- University P. et M. Curie --- LIP6
There is  1 numerical instability
0  UNSTABLE DIVISION(S)
0  UNSTABLE POWER FUNCTION(S)
0  UNSTABLE MULTIPLICATION(S)
```

```
 0  UNSTABLE BRANCHING(S)
 0  UNSTABLE MATHEMATICAL FUNCTION(S)
 0  UNSTABLE INTRINSIC FUNCTION(S)
 1  UNSTABLE CANCELLATION(S)
```

CADNA points out the complete loss of accuracy of the result.

5.2 Example 2: solving a second order equation

The roots of the following second order equation are computed:

$$0.3x^2 - 2.1x + 3.675 = 0.$$

The exact values are: Discriminant d=0, x1=x2=3.5.

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

```
 d =  -0.0000028610227
There are two complex solutions.
z1 = 0.3500000E+01  +  i * 0.8457279E-03
z2 = 0.3500000E+01  +  i * -.8457279E-03
```

and using CADNA in single precision, one obtains:

```
-----------------------------------------------------
CADNA 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
-----------------------------------------------------
 d = @.0
Discriminant is zero.
The double solution is   0.349999E+01
-----------------------------------------------------
CADNA software --- University P. et M. Curie --- LIP6
There are 1 numerical instabilities
0 UNSTABLE DIVISION(S)
0 UNSTABLE POWER FUNCTION(S)
0 UNSTABLE MULTIPLICATION(S)
0 UNSTABLE BRANCHING(S)
0 UNSTABLE MATHEMATICAL FUNCTION(S)
```

```
0 UNSTABLE INTRINSIC FUNCTION(S)
1 UNSTABLE CANCELLATION(S)
```

The standard floating-point arithmetic cannot detect that d=0. The wrong branching is performed and the result is false.

The CADNA software takes the accuracy of operands into account in the order relations or in the equality relation and, therefore, the correct branching is performed and the exact result is obtained.

5.3 Example 3: computing a determinant

The determinant of Hilbert's matrix of size 11 is computed using Gaussian elimination without pivoting strategy. The determinant is the product of the different pivots. Hilbert's matrix is defined by: $a(i, j) = 1/(i + j - 1)$. All the pivots and the determinant are printed out.

The exact value of the determinant is $3.0190953344493\,10^{-65}$.

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

```
Pivot number  1  =  0.1000000000000000D+01
Pivot number  2  =  0.8333333333333331D-01
Pivot number  3  =  0.5555555555555526D-02
Pivot number  4  =  0.3571428571428830D-03
Pivot number  5  =  0.2267573696145566D-04
Pivot number  6  =  0.1431549050529594D-05
Pivot number  7  =  0.9009749264103679D-07
Pivot number  8  =  0.5659971084095516D-08
Pivot number  9  =  0.3551369635569034D-09
Pivot number 10  =  0.2226762517485834D-10
Pivot number 11  =  0.1399228241996033D-11
Determinant      =  0.3028594438809703D-64
```

and using CADNA in double precision, one obtains:

```
-----------------------------------------------------
CADNA 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
```

```
-----------------------------------------------------
Pivot number  1  =  0.100000000000000E+001
Pivot number  2  =  0.833333333333333E-001
Pivot number  3  =  0.55555555555555E-002
Pivot number  4  =  0.3571428571428E-003
Pivot number  5  =  0.22675736961E-004
Pivot number  6  =  0.1431549051E-005
Pivot number  7  =  0.90097493E-007
Pivot number  8  =  0.5659970E-008
Pivot number  9  =  0.35513E-009
Pivot number 10  =  0.2226E-010
Pivot number 11  =  0.14E-011
Determinant      =  0.30E-064
-----------------------------------------------------
CADNA software --- University P. et M. Curie --- LIP6
No instability detected
```

The gradual loss of accuracy is pointed out by CADNA. One can see that the value of the determinant is significant even if it is very "small". This shows how difficult it is to judge the numerical quality of a computed result by its magnitude.

5.4 Example 4: computing a second order recurrent sequence

This example was proposed by J.-M. Muller [10]. The 25 first 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 value of the limit is 6.

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

```
U( 3) = 0.5590163934426237D+01
U( 4) = 0.5633431085044127D+01
U( 5) = 0.5674648620512615D+01
U( 6) = 0.5713329052423919D+01
U( 7) = 0.5749120920462043D+01
U( 8) = 0.5781810933690098D+01
U( 9) = 0.5811314466602178D+01
U(10) = 0.5837660476543959D+01
U(11) = 0.5861018785996283D+01
```

```
U(12) = 0.5882524608269310D+01
U(13) = 0.5918655323805488D+01
U(14) = 0.6243961815306110D+01
U(15) = 0.1120308737284091D+02
U(16) = 0.5302171264499677D+02
U(17) = 0.9473842279276452D+02
U(18) = 0.9966965087355071D+02
U(19) = 0.9998025776093678D+02
U(20) = 0.9999882245337588D+02
U(21) = 0.9999992970745579D+02
U(22) = 0.9999999580049865D+02
U(23) = 0.9999999974893262D+02
U(24) = 0.9999999998498109D+02
U(25) = 0.9999999999910112D+02
 The exact limit is 6.
```

and using CADNA in double precision, one obtains:

```
-----------------------------------------------------
CADNA 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) = 0.55901639344262E+001
U( 4) = 0.5633431085044E+001
U( 5) = 0.56746486205E+001
U( 6) = 0.5713329052E+001
U( 7) = 0.574912092E+001
U( 8) = 0.57818109E+001
U( 9) = 0.581131E+001
U(10) = 0.58377E+001
U(11) = 0.5861E+001
U(12) = 0.588E+001
U(13) = 0.6E+001
U(14) =@.0
U(15) =@.0
U(16) =@.0
U(17) = 0.9E+002
U(18) = 0.999E+002
U(19) = 0.9999E+002
U(20) = 0.99999E+002
U(21) = 0.999999E+002
```

```
U(22) = 0.9999999E+002
U(23) = 0.999999999E+002
U(24) = 0.9999999999E+002
U(25) = 0.99999999999E+002
 The exact limit is 6.
-----------------------------------------------------
CADNA software --- University P. et M. Curie --- LIP6
CRITICAL WARNING: the self-validation detects major problem(s).
The results are NOT guaranteed
There are 9 numerical instabilities
7 UNSTABLE DIVISION(S)
0 UNSTABLE POWER FUNCTION(S)
2 UNSTABLE MULTIPLICATION(S)
0 UNSTABLE BRANCHING(S)
0 UNSTABLE MATHEMATICAL FUNCTION(S)
0 UNSTABLE INTRINSIC FUNCTION(S)
0 UNSTABLE CANCELLATION(S)
```

The traces UNSTABLE DIVISION(S) are generated by divisions where the denominator is a computational zero. Such operations make the computed trajectory turn off the exact trajectory and then, the estimation of accuracy is not possible any more. Even using the double precision, the computer cannot give any significant result after the iteration number 15.

5.5 Example 5: computing a root of a polynomial

This example deals with the improvement and optimization of an iterative algorithm by using new tools which are contained in CADNA. This program computes a root of the polynomial

$$f(x) = 1.47x^3 + 1.19x^2 - 1.83x + 0.45$$

by Newton's method. The sequence is initialized with $x = 0.5$.

The iterative algorithm $x_{n+1} = x_n - \dfrac{f(x_n)}{f'(x_n)}$ is stopped with the criterion

$$|x_n - x_{n-1}| < 10^{-12}.$$

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

```
 x( 29 ) =  0.4285714317551499
 x( 30 ) =  0.4285714317551499
```

and using CADNA in double precision, one obtains:

```
-----------------------------------------------------
CADNA 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
-----------------------------------------------------
 x( 100 ) =  0.4285714E+000
 x( 101 ) =  0.4285714E+000
 -----------------------------------------------------
 CADNA software --- University P. et M. Curie --- LIP6
 CRITICAL WARNING: the self-validation detects major problem(s).
 The results are NOT guaranteed
 There are  501  numerical instabilities
 76  UNSTABLE DIVISION(S)
 0   UNSTABLE POWER FUNCTION(S)
 0   UNSTABLE MULTIPLICATION(S)
 72  UNSTABLE BRANCHING(S)
 0   UNSTABLE MATHEMATICAL FUNCTION(S)
 77  UNSTABLE INTRINSIC FUNCTION(S)
 276 UNSTABLE CANCELLATION(S)
```

With CADNA, one can see that 8 significant digits were lost (despite the apparent stability). By using a symbolic debugger, one can see that, at the last iteration, the denominator is a non-significant value (a computational zero) and that the last answer to the stopping criterion is not reliable. CADNA allows to stop the algorithm when the subtraction $x_n - x_{n-1}$ is non-significant (there is no more information to compute at the next iteration). In Newton's method, a division by a computational zero may suggest a double root. One can simplify the fraction. When these two transformations are done, the code is stabilized and the results are obtained with the best accuracy of the computer. The exact value of the root is $x_{sol} = 3/7 = 0.428571428571428571...$ Now, we obtain:

```
-----------------------------------------------------
CADNA 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
-----------------------------------------------------
```

```
x( 48 ) = 0.428571428571429E+000
x( 49 ) = 0.428571428571429E+000
-----------------------------------------------------
CADNA software --- University P. et M. Curie --- LIP6
No instability detected
```

5.6 Example 6: solving a linear system

In this example, CADNA is able to provide correct results which were impossible to be obtained with the standard floating-point arithmetic. The following linear system is solved using Gaussian elimination with partial pivoting. The system is

$$\begin{pmatrix} 21 & 130 & 0 & 2.1 \\ 13 & 80 & 4.74\ 10^{8} & 752 \\ 0 & -0.4 & 3.9816\ 10^{8} & 4.2 \\ 0 & 0 & 1.7 & 9\ 10^{-9} \end{pmatrix} . X = \begin{pmatrix} 153.1 \\ 849.74 \\ 7.7816 \\ 2.6\ 10^{-8} \end{pmatrix}$$

The exact solution is $x_{sol}^{t} = (1, 1, 10^{-8}, 1)$. Using IEEE single precision arithmetic with rounding to the nearest, one obtains:

```
x_sol(1) =     0.6261988E+02     (exact solution:     0.1000000E+01)
x_sol(2) =    -0.8953979E+01     (exact solution:     0.1000000E+01)
x_sol(3) =     0.0000000E+00     (exact solution:     0.1000000E-07)
x_sol(4) =     0.1000000E+01     (exact solution:     0.1000000E+01)
```

and using CADNA in single precision, one obtains:

```
-----------------------------------------------------
CADNA 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
-----------------------------------------------------
x_sol(1) =     0.999E+00         (exact solution:     0.1000000E+01)
x_sol(2) =     0.1000E+01        (exact solution:     0.1000000E+01)
x_sol(3) =     0.999999E-08      (exact solution:     0.9999999E-08)
x_sol(4) =     0.1000000E+01     (exact solution:     0.1000000E+01)
-----------------------------------------------------
```

```
CADNA software --- University P. et M. Curie --- LIP6
There are 3 numerical instabilities
0 UNSTABLE DIVISION(S)
0 UNSTABLE POWER FUNCTION(S)
0 UNSTABLE MULTIPLICATION(S)
1 UNSTABLE BRANCHING(S)
0 UNSTABLE MATHEMATICAL FUNCTION(S)
1 UNSTABLE INTRINSIC FUNCTION(S)
1 UNSTABLE CANCELLATION(S)
```

During the reduction of the third column, the matrix element a(3,3) is equal to 4864. But the exact value of a(3,3) is zero. The standard floating-point arithmetic cannot detect that a(3,3) is non-significant. This value is chosen as pivot. That leads to erroneous results. CADNA detects the non-significant value of a(3,3). This value is eliminated as pivot. That leads to satisfactory results.

With this simple example, we show how numerical debugging (introduced in 3.11) can be performed in order to identify which instructions are responsible for instabilities. As described in 3.11, using a symbolic debugger, a file containing all the traces of instabilities can be created. With the present example, the relevant part of this file, obtained using **gdb** with Linux and named *gdb.out*, is shown below.

```
Breakpoint 1, 0x080599e6 in instability_ ()
(gdb)  > > >#0  0x080599e6 in instability_ ()
#1  0x0804e18f in cadna_sub_MP_sub_st_st__ ()
#2  0x08049b8b in MAIN_ () at ex6_cad.f90:59
#3  0x08063c56 in main (argc=1, argv=0xbfc0de34)

Breakpoint 1, 0x080599e6 in instability_ ()
#0  0x080599e6 in instability_ ()
#1  0x08056e3e in cadna_intr_MP_abs_st__ ()
#2  0x080496ba in MAIN_ () at ex6_cad.f90:37
#3  0x08063c56 in main (argc=1, argv=0xbfc0de34)

Breakpoint 1, 0x080599e6 in instability_ ()
#0  0x080599e6 in instability_ ()
#1  0x08059993 in cadna_gt_MP_gt_st_st__ ()
#2  0x080496d2 in MAIN_ () at ex6_cad.f90:37
#3  0x08063c56 in main (argc=1, argv=0xbfc0de34)
```

From the *gdb.out* file, the first instability is caused by the instruction located at line 59 in the Fortran source file *ex6_cad.f90*. This instruction is:

```
a(k,j)=a(k,j) - aux*a(i,j)
```

The instability is due to the subtraction of two single precision stochastic variables, *i.e.* of type single_st. When the cadna_end function is called, this instability generates the message

```
1 UNSTABLE CANCELLATION(S)
```

The second instability is caused by the instruction located at line 37 in the Fortran source file:

```
if (abs(a(j,i)).gt.pmax) then
```

This instability is due to the abs intrinsic function used with a single precision stochastic argument and generates the output message

```
1 UNSTABLE INTRINSIC FUNCTION(S)
```

Finally, the third instability is caused by the same instruction. It is due to the gt relational operator used with two single precision stochastic arguments. This instruction generates the output message

```
1 UNSTABLE BRANCHING(S)
```

An additional tool which, for each type of instability, lists all the instructions responsible for it is currently under development.

5.7 Example 7: when CADNA fails

CADNA is based on a probabilistic model. It should never be forgotten that all the estimations computed by CADNA are probabilistic, even if the probability is close to 1. Moreover, the theoretical model shows that CADNA is able to estimate the round-off errors of the first order. If they represent the global round-off errors, CADNA works well but, if they are dominated by terms of greater order, CADNA may fail. That is what happened in example 4. However because of an unstable division, the problem has been detected.

In the present example, we have the same behaviour but only with additions and subtractions, so without any warning of numerical instability. Let us perform the following computation:

```
x=6.83561d+05
y=6.83560d+05
z=1.00000000007d0
r = z - x
```

```
r1 = z - y
r = r + y
r1 = r1 + x
r1 = r1 - 2
r = r + r1
!      r = ((z-x)+y) + ((z-y)+x-2)
```

The exact result is $1.4\,10^{-10}$. The result obtained using IEEE double precision arithmetic with rounding to the nearest is 2.3283064365386963E-10

With CADNA, because we essentially performed the same computation, $((z-x)+y)$ and $((z-y)+x-2)$, we find that if the same rounding mode is chosen for both parts, the final result appears as exact but it is wrong. It happens in one case in four and the result provided by CADNA is then 0.116415321826935E-009 with 15 exact significant digits. If computations are performed 100,000 times using CADNA, one may obtain:

```
-----------------------------------------------------
CADNA 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
-----------------------------------------------------
 Enter the number of iterations
100000
r = @.0                    ;  ierr =  27289
-----------------------------------------------------
CADNA software --- University P. et M. Curie --- LIP6
There are  300000  numerical instabilities
0  UNSTABLE DIVISION(S)
0  UNSTABLE POWER FUNCTION(S)
0  UNSTABLE MULTIPLICATION(S)
0  UNSTABLE BRANCHING(S)
0  UNSTABLE MATHEMATICAL FUNCTION(S)
0  UNSTABLE INTRINSIC FUNCTION(S)
300000  UNSTABLE CANCELLATION(S)
```

The last value of r is printed out, and also ierr the number of times when the result was wrong. The corresponding source code is:

```
program ex7
  use cadna
  implicit none
```

```
  type(double_st) :: r,r1,x,y,z
  integer :: i, nloop, ierr
  call cadna_init(-1)
  print *,' Enter the number of iterations'
  read *,nloop
  ierr = 0
  do i=1,nloop
     x=6.83561d+05
     y=6.83560d+05
     z=1.00000000007d0
     r = z - x
     r1 = z - y
     r = r + y
     r1 = r1 + x
     r1 = r1 - 2
     r = r + r1
  !      r = ((z-x)+y) + ((z-y)+x-2)
     if(r.ne.1.4d-10) ierr = ierr + 1
  enddo
  print *, 'r = ', str(r), ';  ierr = ',ierr
  call cadna_end()
end program ex7
```

6 CADNA library structure

The CADNA source code consists of one assembly language file described in subsection 6.1 and eleven Fortran 90 language files described in subsections 6.2 to 6.12.

6.1 cadna_rounding.s

cadna_rounding.s is a symbolic link to the assembly file corresponding to the processor and the Fortran compiler used. This assembly file contains five routines which change the rounding mode and set it to one of the rounding modes defined by the IEEE 754 standard [1]. These routines are called in the CADNA Fortran source codes. Indeed frequent changes of the rounding mode must be performed for random arithmetic. They can also be called in any Fortran code compiled with the CADNA library. The effects of these routines are listed below.

- rnd_arr: sets the rounding mode to rounding to the nearest.

- rnd_moinf: sets the rounding mode to rounding towards $-\infty$ (or downward rounding).
- rnd_plinf: sets the rounding mode to rounding towards $+\infty$ (or upward rounding).
- rnd_zero: sets the rounding mode to rounding towards 0.
- rnd_switch: switches the rounding mode from $+\infty$ to $-\infty$, or from $-\infty$ to $+\infty$.

The rounding mode can also be set by using the IEEE_set_rounding_mode subroutine from the IEEE_ARITHMETIC intrinsic module provided by the Fortran 2003 standard. Despite its portability, this subroutine is not used in the CADNA library because of its cost in terms of performance. The routines in assembly language previously described, which are frequently called in the library, are optimized for the processor and the Fortran compiler chosen.

6.2 cadna_type.f90

cadna_type.f90 contains the cadna_type module which defines several types, parameters, subroutines, ...

The cadna_type module contains the definition of the stochastic types single_st and double_st described in 2.2.

cadna_type contains the declaration and the initialization of integer parameters which can be used to specify which kind of instability to detect or not, as described in 2.5.1.

cadna_type contains the declaration and the initialization to TRUE of the following logical variables:
cadna_branching_tag,
cadna_mathematic_tag,
cadna_intrinsic_tag,
cadna_cancellation_tag,
cadna_division_tag,
cadna_power_tag,
cadna_multiplication_tag.
If one of these variables is set to TRUE or FALSE the corresponding instability is respectively enabled or disabled. These are internal variables; they are only used in the CADNA Fortran source code.

cadna_type contains the declaration and the initialization to FALSE of the cadna_deactivate logical variable, which is internal to CADNA. If this variable is set to TRUE or FALSE CADNA self-validation is respectively deactivated or activated. Indeed the validation of the CESTAC method requires a dynamical

control of divisions, multiplications (and therefore power operations) [4,2].

cadna_type contains the declaration and the initialization to 0 of integer variables used to count the instabilities of each kind which may occur at run time.

cadna_type contains the cadna_enable and cadna_disable subroutines which can be used to enable or disable one kind of instability as described in 2.5.6. If the detection of instabilities generated by divisions, multiplications, or power operations is disabled, then the cadna_deactivate variable is set to TRUE, *i.e.* CADNA self-validation is deactivated.

cadna_type contains the cadna_init subroutine which has to be called to initialize the library, as described in 2.5.1.

With CADNA, a random rounding mode is used. This implies frequent changes of the rounding mode, which are performed or not, depending on the value returned by the myrandom function defined in cadna_type. Indeed the myrandom function returns a logical value which is randomly assigned.

Finally, cadna_type contains the cadna_end subroutine which lists all the numerical instabilities which occurred at run time, as described in 2.5.1.

6.3 cadna_cestac.f90

cadna_cestac.f90 contains the cadna_cestac module which defines:

- the nb_significant_digit function described in 2.5.3
- the computed_zero function described in 2.5.4.

These functions can be used in any Fortran code to be compiled with the CADNA library and accept a single precision or a double precision stochastic argument. Both functions are actually interfaces. Two versions exist for each function:

- one for single precision stochastic variables,
- one for double precision stochastic variables.

6.4 cadna_convert.f90

cadna_convert.f90 contains the cadna_convert module which defines the conversion functions (int, nint, aint, anint, real, dble) described in 2.3.1 and the old_type function described in 2.5.5.

The cadna_convert module also contains the data_st subroutine which perturbs data, as described in 2.5.7.

6.5 cadna_to.f90

cadna_to.f90 contains the cadna_to module which overloads the assignment operator (=). If a stochastic variable is involved in an assignment, one of the subroutines of the cadna_to module is actually called. Indeed the cadna_to module contains as many subroutines as ordered pairs of types which can be involved in the assignment.

6.6 Overloading of arithmetic operators

Arithmetic operators are overloaded in the following files:
cadna_add.f90,
cadna_sub.f90,
cadna_mul.f90,
cadna_div.f90.
These files contain the cadna_add, cadna_sub, cadna_mul and cadna_div modules. If a stochastic variable is an operand of an arithmetic operator, one of the functions of the corresponding module is actually called. Indeed each module defines as many functions as ordered pairs of types which can be involved in the operation.

6.7 Overloading of relational operators

Relational operators are overloaded in the following files:
cadna_equ.f90,
cadna_ge.f90,
cadna_gt.f90,
cadna_le.f90,
cadna_lt.f90.
These files contain the cadna_equ, cadna_ge, cadna_gt, cadna_le and cadna_lt modules. The name of each module indicates which operator it overloads. It is worth noting that the cadna_equ module contains the functions which overload the EQ (EQual) and the NE (Not Equal) operators. One function exists per ordered pair of types which can be involved in a kind of test. The overloading of relational operators takes into account the numerical quality of the operands, as described in 2.4.

6.8 cadna_intr.f90

cadna_intr.f90 contains the cadna_intr module which overloads the following functions: abs, dim, max, min, mod, sign. Detailed information on the definition of these functions for stochastic arguments is given in 2.3.2.

6.9 cadna_math.f90

cadna_math.f90 contains the cadna_math module which overloads the mathematical functions listed in 2.3.3. Detailed information on the definition of these functions for stochastic arguments is given in 2.3.3.

6.10 cadna_str.f90

cadna_str.f90 contains the cadna_str module which defines the str function. This function prints the exact significant digits of a stochastic argument, as described in 2.5.2.

6.11 cadna.f90

cadna.f90 contains the cadna module which just enables to use all the modules previously presented.

6.12 cadna_unstab.f90

cadna_unstab.f90 contains the instability subroutine. This subroutine has an integer argument which characterizes a type of instability. This subroutine, which increments the counter inherent to a type of instability, is called each time a numerical instability occurs.

7 Conclusion

In this paper, we have presented the CADNA library which enables one to control the numerical quality of any scientific code written in Fortran. CADNA can estimate in any (final or intermediate) result which digits are affected by

round-off error propagation. Furthermore as CADNA can detect any numerical instability which may occur at run time, a numerical debugging of the code can be performed.

Implementing CADNA in a code mainly consists in replacing the standard numerical types by stochastic types, on which round-off error propagation can be estimated. CADNA provides the stochastic types and the definition of arithmetic operators, relational operators and functions which have been overloaded to be used with stochastic variables. Thanks to this overloading, CADNA can be used without having to rewrite or notably change the initial code.

The benefits of CADNA have been pointed out in Section 5 with several test programs which implement direct or iterative algorithms. However as the CADNA library is based on a probabilistic model of round-off errors, it is not infallible. CADNA may consider as reliable a result which is actually not. This drawback is illustrated in the last example presented in Section 5. Nevertheless the prediction of such over-optimistic results is rare. Indeed the last program in Section 5 has been written on purpose. The opposite trend, which is due to the statistic parameters chosen for CADNA is usually observed: the numerical quality estimated by CADNA is rather slightly pessimistic.

Although attention has been paid to performance optimization in the CADNA library, its cost remains a factor of about 3.5 for memory and at least 3 for execution time. This run time performance degradation depends on the architecture used and on the numerical instabilities to be detected. The changes of the rounding mode required by the Discrete Stochastic Arithmetic are performed using routines in assembly language which are frequently called in the CADNA source code. These routines have been optimized according to the target processor and depend also on the Fortran compiler used. In order to enable round-off error estimation at a moderate computing cost, future improvements in the CADNA library will depend on the definite evolution of computer architectures and compilers.

Acknowledgements The authors sincerely wish to thank N.S. Scott and the reviewers for their careful reading and their constructive comments.

References

[1] IEEE Computer Society, New York. *IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985*, 1985. Reprinted in SIGPLAN Notices, 22(2):9-25, 1987.

[2] J.-M. Chesneaux. *L'arithmétique stochastique et le logiciel CADNA.*

Habilitation à diriger des recherches, Université Pierre et Marie Curie, Paris, November 1995.

[3] J. Vignes. Discrete stochastic arithmetic for validating results of numerical software. *Num. Algo.*, 37(1–4):377–390, December 2004.

[4] J.-M. Chesneaux. Study of the computing accuracy by using probabilistic approach. In C. Ullrich, editor, *Contribution to Computer Arithmetic and Self-Validating Numerical Methods*, pages 19–30, IMACS, New Brunswick, New Jersey, USA, 1990.

[5] J. Vignes. A stochastic arithmetic for reliable scientific computation. *Math. Comput. Simulation*, 35:233–261, 1993.

[6] J. Vignes and M. La Porte. Error analysis in computing. In *Information Processing 1974*, pages 610–614. North-Holland, 1974.

[7] N.S. Scott, F. Jézéquel, C. Denis, and J.-M. Chesneaux. Numerical 'health check' for scientific codes: the CADNA approach. *Computer Physics Communications*, 176(8):507–521, April 2007.

[8] J. Vignes. Zéro mathématique et zéro informatique. *C. R. Acad. Sci. Paris Sér. I Math.*, 303:997–1000, 1986. also: La Vie des Sciences, 4 (1) 1-13, 1987.

[9] S.M. Rump. *Reliability in Computing. The Role of Interval Methods in Scientific Computing.* Academic Press, 1988.

[10] J.-M. Muller. *Arithmétique des Ordinateurs.* Masson, 1989.