CADNA for FORTRAN source codes

Laboratoire d'Informatique de Paris 6
Université Pierre et Marie Curie - Paris 6
Paris, France
http://www.lip6.fr/cadna
cadna-team@lip6.fr

Contents

1 Introduction **5**

2 Reference guide **7**

2.1 Aim of the CADNA library 7
2.2 Stochastic types 9
2.3 Intrinsic functions 9
2.3.1 Conversion functions 9
2.3.2 Numerical functions 10
2.3.3 Mathematical functions 11
2.4 Relational operators 11
2.5 CADNA specific functions 12
2.5.1 Initializing and closing the library 12
2.5.2 Obtaining a string from a result with its evaluated accuracy 13
2.5.3 Obtaining the number of exact significant digits of a stochastic variable 14
2.5.4 Testing if a variable is a computational zero 14
2.5.5 Obtaining a standard value from a stochastic variable 14
2.5.6 Enabling and disabling the detection of instabilities . 15
2.5.7 Reducing accuracy of initial data 15

3 User guide **17**

3.1 Declaration of the CADNA library 18
3.2 Initialization and termination of the CADNA library 18
3.3 Declaration of variables 18
3.3.1 Changes in the type of variables 18
3.3.2 Changes in the name of some variables 19
3.4 DATA for initializing stochastic variables 19
3.5 Changes in assignments or arithmetic operations 19

3.5.1 Conversions between usual types and stochastic types 19
3.5.2 Standard arithmetic operators 20
3.5.3 Vector operators and functions 20
3.6 Changes in reading statements 21
3.7 Changes in printing statements 21
3.8 Changes in intrinsic functions 22
3.8.1 Changes of non generic names 22
3.8.2 Changes in the call of min and max functions 22
3.8.3 Suppression of intrinsic functions declarations 22
3.9 Changes in statement functions 22
3.10 Constants passed as function arguments 23
3.11 An example of numerical code and its modified version 24
3.11.1 Standard Fortran source code 24
3.11.2 Source code using the CADNA library 25
3.11.3 Example of execution without CADNA 27
3.11.4 Example of execution with CADNA 27
3.12 Numerical debugging with CADNA 28

4 Installation instructions and test runs 31
4.1 Installation instructions 31
4.2 Test runs 32
4.2.1 Example 1: a rational fraction function of two variables 32
4.2.2 Example 2: solving a second order equation 33
4.2.3 Example 3: computing a determinant 34
4.2.4 Example 4: computing a second order recurrent sequence 35
4.2.5 Example 5: computing a root of a polynomial 37
4.2.6 Example 6: solving a linear system 39
4.2.7 Example 7: when CADNA fails 42

Chapter 1

Introduction

The IEEE standard floating-point arithmetic [48] only approximates exact arithmetic. So, when a scientific code is run on a computer which respects the IEEE standard, its results are not exact; the approximation introduces a round-off error for each arithmetic statement, as always does the assignment statement (because registers have more digits than memory words), when the value cannot be exactly coded. Validation of numerical results is a real problem for scientific computing. Too long ignored by users, it is now recognized as an essential topic.
The CADNA environment [38, 24, 12] enables you to develop robust, high performance, numerical applications. CADNA can help investigate unusual behavior of numerical program written in ADA, C or FORTRAN.
CADNA is based on the CESTAC (Contrôle et Estimation Stochastique des Arrondis de Calcul) method [45, 35, 32]. This method 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 bits in these samples estimates the number of exact significant bits in the floating-point result. The deterministic arithmetic of the computer is replaced by a so-called "Discrete Stochastic Arithmetic" (DSA) [12].
This manual serves as a tool to enable the use of the options and flexibility provided by CADNA on numerical applications. CADNA (Control of Accuracy and Debugging Numerical Applications) is a library devoted to programs written in ADA, C or FORTRAN. CADNA allows, during the execution of the 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.

The next chapters are described below:

- Chapter 2 is a reference guide that describes types, subroutines and functions that compose the CADNA library.
- Chapter 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. It also gives a complete example of numerical code, with the original and modified versions.
- Chapter 4 gives instructions for the installation of CADNA, describes how to test the library and comments on the results of the test programs.

Chapter 2

Reference guide

2.1 Aim of the CADNA library

The arithmetic commonly used on computers for scientific programming is floating-point arithmetic. This arithmetic only approximates exact arithmetic. Consequently each arithmetic statement generates a round-off error.

So when a correct program with regard to syntax and logical organization is running on a computer, every produced result is unavoidably given with a so called "computing error". This error is due to all the round-off errors produced along the elementary statements required to obtain the result. Sometimes the error may be such that the final result is really wrong (and not only inaccurate).

The aim of the CADNA library presented here is to answer the following question:

What is the computing error due to floating-point arithmetic on the results produced by any program running on a computer?

So, we want to estimate the round-off error on each result with a technique which is independent on the program and hence on the algorithm used.

CADNA is a library, more precisely it is a set of data types, functions and subroutines that may be used in any program written in Fortran. It implements the CESTAC method in a synchronous way. With a few modifications in the source code, this library has for main purpose to estimate the effects of round-off error propagation on every numerical computed result. It also allows to study the effects of the initial data uncertainties upon computed results, as described in 2.5.

This implementation consists in replacing the computer deterministic arithmetic by a stochastic arithmetic and in performing N times ($N = 3$) each elementary operation before executing the next statement.

Thus, it is as N identical programs were simultaneously running on N synchronized computers each of them using random arithmetic. So for each result, we obtain N samples from which we compute the mean value and the standard deviation which characterize the corresponding stochastic number. The value of this number is defined as the mean value of the different samples. The accuracy of this number, *i.e.* its number of exact significant digits, is estimated using the mean value and the standard deviation. If all the samples are equal to zero or if the number of exact significant digits is less than one, then the number is defined as a computational zero. This means that a computational zero is either the mathematical zero or a number without any significance.

So round-off error propagation can be analysed step by step. Numerical instabilities and non-significant results are detected. The branchings based on order relations may also be controlled. Therefore, this synchronous implementation of the CESTAC method allows to validate any scientific code during its run.

With the CADNA library, one can run any scientific code using random arithmetic, without having to rewrite or notably change the initial code. This tool has been written in Fortran 90 which is an important language for scientific computation. This language enables to create new numerical **types** with their operators; furthermore the designating symbol of an operator can be choosen among the primitive symbols in the language ($+$, $*$,...). In other words, this language enables the so called "operator overloading". Thanks to these new properties, CADNA has been developed for Fortran programs.

Thus a new numerical type has been created, the **stochastic number**; it is nothing else that a N-set ($N = 3$) containing perturbed floating-point values. All the arithmetic operators ($+$, $-$, $*$, $/$) have been overloaded, also for 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 ($>$, $\geq$, $<$, $\leq$, $==$) are overloaded satisfying the properties described in [12, 24]. All standard functions defined in Fortran (SIN, COS, EXP, ...) have also been overloaded. Likewise, in/out 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 is 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 sections.

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 for 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 output 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 output 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

- $\frac{\sum_{i=1}^{N} a_i}{N} > \frac{\sum_{i=1}^{N} b_i}{N}$.

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.

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 on the last digit provided by the* str *function.* When the argument has no significant digit, the string that is returned is @.0.

The number of characters in the output string is:

14 for a single_st variable ;
23 for a double_st variable ;

Example:

```
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

The generic function read is adapted to standard floating-point variables which must be transformed into stochastic variables (cf 3.6). These stochastic variables have the maximal accuracy represented by N equal samples. However 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.

Chapter 3

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.

The reader may refer to the sample program given in 3.11 with two versions, *i.e.* the initial Fortran code and the code modified to be compiled with the CADNA library.
The last paragraph of this chapter deals with 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 functions 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.

3.3 Declaration of variables

3.3.1 Changes in the type of variables

To control the numerical quality of a variable, just replace its standard type 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, add this line as a general declaration:
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 :: don	**type (single_st)** :: don
data don/3.245/	data **don%x, don%y, don%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 all knowledge of accuracy.

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

Example:

Initial Fortran statements	Modified statements for CADNA
	use cadna
real :: x,y	**type (single_st)** :: x,y
	call cadna_init(-1)
x=1.234	x=1.234
	call data_st(x)
y=-3.	y=-3.

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 been overloaded for stochastic arrays with the same rules as above.

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 An example of numerical code and its modified version

The following source codes use the Gauss-Jordan method to invert a matrix.

3.11.1 Standard Fortran source code

```
program Inversion
implicit none
integer n
parameter (n=4)
real M(n,n)
write(*,*)'Initial matrix:'
call InitMat(M,n)
call InvertMat(M,n)
write(*,*)
write(*,*)'Inverted matrix:'
call DisplayMat(M,n)
end program Inversion

! Initialization:
subroutine InitMat(M,n)
implicit none
integer i,j,n
real M(n,n)
do i=1,n
   read(*,*)(M(i,j),j=1,n)
enddo
return
end subroutine InitMat

! Inversion using the Gauss-Jordan method:
 subroutine InvertMat(M,n)
 implicit none
 integer n,i,j,k
 real M(n,n),temp
 do k=1,n
    temp=M(k,k)
    M(k,k)=1.
    do j=1,n
```

```
        M(k,j)=M(k,j)/temp
      end do
      do i=1,n
        if (i.ne.k) then
          temp=M(i,k)
          M(k,k)=0.
          do j=1,n
            M(i,j)=M(i,j)-temp*M(k,j)
          end do
        endif
      end do
 enddo
 return
 end subroutine InvertMat

! Display of a matrix:
subroutine DisplayMat(M,n)
implicit none
integer n,i,j
real M(n,n)
10     format(10E15.7)
do i=1,n
     write(*,10)(M(i,j),j=1,n)
 enddo
 return
 end subroutine DisplayMat
```

3.11.2 Source code using the CADNA library

```
program Inversion
use cadna
implicit none
integer n
parameter (n=4)
type (single_st) M(n,n)
call cadna_init(-1)
write(*,*)'Initial matrix:'
call InitMat(M,n)
call InvertMat(M,n)
write(*,*)
```

```
write(*,*)'Inverted matrix:'
call DisplayMat(M,n)
call cadna_end()
end program Inversion

! Initialization:
subroutine InitMat(M,n)
use cadna
implicit none
integer i,j,n
type (single_st) M(n,n)
real aux(n)
do i=1,n
   read(*,*)(aux(j),j=1,n)
   do j=1,n
         M(i,j)=aux(j)
   enddo
enddo
return
end subroutine InitMat

! Inversion using the Gauss-Jordan method:
subroutine InvertMat(M,n)
use cadna
implicit none
integer n,i,j,k
type (single_st) M(n,n),temp
do k=1,n
    temp=M(k,k)
    M(k,k)=1.
    do j=1,n
       M(k,j)=M(k,j)/temp
    end do
    do i=1,n
       if (i.ne.k) then
          temp=M(i,k)
          M(k,k)=0.
          do j=1,n
             M(i,j)=M(i,j)-temp*M(k,j)
          end do
```

```
      endif
    end do
enddo
return
end subroutine InvertMat

! Display of a matrix:
subroutine DisplayMat(M,n)
use cadna
implicit none
integer n,i,j
type (single_st) M(n,n)
10 format(10a15)
do i=1,n
    write(*,10)(str(M(i,j)),j=1,n)
enddo
return
end subroutine DisplayMat
```

3.11.3 Example of execution without CADNA

```
 Initial matrix:
1.     2.E3 0.5   4.
3.E-5  1.   2.    8.
4.     0.5  3.E-8 2.
2.     3.   0.5   5.E9

 Inverted matrix:
  0.1000063E+01 -0.1177399E-03 -0.4254787E+04  0.4998230E+00
 -0.5000938E-03  0.1000118E+01  0.2127644E+01  0.7497645E-03
  0.2500460E-03 -0.4700588E+00 -0.6802303E-05  0.3999618E+01
  0.2499951E-12  0.4699354E-10  0.1700738E-05  0.0000000E+00
```

3.11.4 Example of execution with CADNA

```
 Initial matrix:
1.     2.E3 0.5   4.
3.E-5  1.   2.    8.
4.     0.5  3.E-8 2.
2.     3.   0.5   5.E9
```

```
Inverted matrix:
 0.1000062E+01 @.0            -0.4254788E+04  0.50E+00
-0.500093E-03   0.1000117E+01  0.2127643E+01  0.75E-03
 0.250045E-03  -0.4700587E+00 -0.6802302E-05  0.3999617E+01
 0.250E-12      0.468E-10      0.1700738E-05  0.0000000E+00
```

3.12 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 with 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 and UNSTABLE MULTIPLICATION, see [38, 24, 12].

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 &

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.
P.S.: **nohup** allows to keep the process alive even when logging off.
The *gdb.out* file will contain all the traces of instabilities.

Chapter 4

Installation instructions and test runs

4.1 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
```

4.2 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.

4.2.1 Example 1: a rational fraction function of two variables

In the following example [40], 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.

4.2.2 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.

4.2.3 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.

4.2.4 Example 4: computing a second order recurrent sequence

This example was proposed by J.-M. Muller [37]. 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.

4.2.5 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
```

4.2.6 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 $x_{sol}^t = \left(1, 1, 10^{-8}, 1\right)$. 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 3.12) can be performed in order to identify which instructions are responsible for instabilities. As described in 3.12, 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.

4.2.7 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
```

Bibliography

[1] J.-M. Chesneaux, F. Jézéquel, J.-L. Lamotte, Stochastic arithmetic and verification of mathematical models In Uncertainties in environmental modelling and consequences for policy making, P. Baveye, J. Mysiak, M. Laba Eds., NATO Science for Peace and Security Series - C: Environmental Security, Springer, pages 101-125, 2009.

[2] J.-M. Chesneaux, S. Graillat, F. Jézéquel, Numerical validation and assessment of numerical accuracy, Oxford e-Research Centre, overview article, 44 pages, march 2009, `http://cpc.cs.qub.ac.uk/oerc_numerical_accuracy.pdf`.

[3] J.-M. Chesneaux, S. Graillat, F. Jézéquel, Rounding errors, invited paper, In Wiley Encyclopedia of Computer Science and Engineering (Benjamin Wah, ed.) Hoboken: John Wiley & Sons, vol. 4, pages 2480-2494, 2009.

[4] F. Jézéquel, J.-M. Chesneaux, CADNA: a library for estimating round-off error propagation, *Computer Physics Communications*, 178(12), pages 933-955, 2008.

[5] N. S. Scott, V. Faro-Maza, M. P. Scott, T. Harmer, J.-M. Chesneaux, C. Denis, F. Jézéquel, E-Collisions using e-Science, *Physics of Particles and Nuclei Letters*, 5(3), pages 150-156, 2008.

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

[7] N.S. Scott, L. Gr. Ixaru, C. Denis, F. Jézéquel, J.-M. Chesneaux, M.P. Scott, High performance computation and numerical validation of e-collision software, *"Trends and Perspectives in Modern Computational Science"*, Invited lectures, ICCMSE 2006 conference, Interna-

tional Conference of Computational Methods in Sciences and Engineering, G. Maroulis and T. Simos (Eds), Lecture Series on Computer and Computational Sciences, vol. 6, pages 561-570, 2006.

[8] F. Jézéquel, A dynamical strategy for approximation methods, *C. R. Acad. Sci. Paris - Mecanique*, 334, pages 362-367, 2006.

[9] F. Jézéquel, F. Rico, J.-M. Chesneaux, M. Charikhi, Reliable computation of a multiple integral involved in the neutron star theory, *Mathematics and Computers in Simulation*, 71(1), pages 44-61, 2006.

[10] F. Jézéquel, Dynamical control of approximation methods, Habilitation à diriger des recherches, Université Pierre et Marie Curie, Paris, 2005.

[11] J.-L. Lamotte, Vers une chaîne de validation des logiciels numériques l'aide de méthodes probabilistes, Habilitation à diriger des recherches, Université Pierre et Marie Curie, Paris, 2004.

[12] J. Vignes, Discrete Stochastic Arithmetic for Validating Results of Numerical Software, *Special Issue of Numerical Algorithms*, 2004, 37, pp. 377-390

[13] F. Jézéquel, J.-M. Chesneaux, Computation of an infinite integral using Romberg's method, *Numerical Algorithms*, 36 (3): 265-283, July 2004.

[14] F. Jézéquel, Dynamical control of converging sequences computation, *Applied Numerical Mathematics*, 50(2): 147-164, 2004.

[15] F. Jézéquel, J.-M. Chesneaux, For reliable and powerful scientific computations, *Sc. Comp. Val. Num.*, Krämer and Wolff von Gudenberg ed., Kluwer Academic/Plenum publishers (2001) 367-378,

[16] M. Montagnac, J.-M. Chesneaux, Dynamic control of a BICGStab algorithm, *Applied Numerical Mathematics*, vol. 32 (2000), 103-117.

[17] N. C. Albertsen, J.-M. Chesneaux, S. Christiansen, A. Wirgin, Comparison of four software packages applied to a scattering problem. *Math. Comput. Simul.*, 48(3): 307-317 (1999).

[18] J.-M. Chesneaux, F. Jézéquel, Dynamical control of computations using the Trapezoidal and Simpson's rules *Journal of Universal Computer Science*, Vol. 4 (1), 2-10, 1998.

[19] R. Alt, J. Vignes, Validation of results of collocation methods for ODEs with the CADNA library, *Appl. Num. Maths*, 20, 1996, pp 1-21.

[20] J.-M. Chesneaux, B. Troff, Computational stability study using the CADNA software applied to the navier-stokes solver PEGASE *Scientific Computing and Validated Numerics*, 1996, pp 84-90.

[21] J.M. Chesneaux, A. Matos, Breakdown and near-breakdown control in the CGS algorithm using stochastic arithmetic *Numerical Algorithms*, 11, 1996, pp 99-116.

[22] M. Pichat, J. Vignes, Validité des résultats numériques dans les processus à comportement chaotique. Un outil d'évaluation : le logiciel CADNA. *CRAS*, Paris, Tome 322, Série 2b, 1996, pp. 681-688.

[23] N.C. Albertsen, J.-M. Chesneaux, S. Christensen, A. Wirgin, Evaluation of Round-off Error by Interval and Stochastic Arithmetic Methods in a Application of the Rayleigh Theory to the Study of Scattering from an Uneven Boundary. *Math. and Num. Aspect of Waves Propagation. Proc. 3rd Inter.Conf.* G. Cohen Ed. SIAM Proc Philadelphia, 1995, pp. 338-346.

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

[25] J.-M. Chesneaux, The equality relations in scientific computing, *Num. Algo 7*, 1994, pp. 129-143.

[26] J.-M. Chesneaux, A. Wirgin, Reflection from a corrugated surface revisited. *J. Acoust. Soc. Am 96.(1)*, 1994, pp. 1-16.

[27] S. Guilain, J. Vignes, Validation of numerical software results. Application to the computation of apparent heat release in direct-injection diesel engines. *Math and Comp. in Sim. 37*, 1994, pp. 73-92.

[28] M. Pichat, Chaotic evolution and stochastic arithmetic. *Proc. 14th, IMACS World Congress*, Atlanta, 1994.

[29] J.-M. Chesneaux, J. Vignes, L'algorithme de Gauss en Arithmétique Stochastique, *C.R Acad. Sci.*, Paris, Sér.II, 316, 1993, pp. 171-176.

[30] S. Guilain, J. Vignes, Qualification des logiciels numériques. Application à un logiciel d'analyse de la combustion dans les moteurs à allumage commandé. *Revue de l'Institut du Pétrole*. Vol. 48, 5, 1993, pp. 545-575.

[31] M. Pichat, J. Vignes, Ingéniérie du contrôle de la précision des calculs sur ordinateur. *Ed. Technip*, Paris 1993.

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

[33] J.-M. Chesneaux, J. Vignes, Les fondements de l'arithmétique stochastique, *C.R Acad. Sci.*, Paris, Sér.I, 315, 1992, pp. 1435-1440.

[34] J. Vignes, Optimization software validation. *Times XXXX Sobrapo XXIII. Joint Inter. Meeting* Rio de Janeiro, 1991.

[35] J.-M. Chesneaux, Study of the computing accuracy by using probabilistic approach, *Contribution to comp. arithmetic and Self-Validating Numerical Methods*, C. Ullrich ed., IMACS, New Brunswick, NJ, 1990, pp. 19-30.

[36] J. Vignes, Estimation de la précision des résultats de logiciels numériques. *La Vie des Sciences, Comptes Rendus, série générale*, 7, 1990, pp. 93-143.

[37] J.-M. Muller, Arithmétique des ordinateurs, Masson, 1989.

[38] J.-M. Chesneaux, Étude théorique et implémentation en ADA de la méthode CESTAC, *Thèse de l'université P. et M. Curie*, Paris, 1988.

[39] J.-M. Chesneaux, J. Vignes, Sur la robustesse de la méthode CESTAC, *C.R. Acad. Sc.* Paris, Sér. I Math. 307, 1988, pp. 855-860.

[40] S.M. Rump, Algorithms for Verified Inclusions - Theory and Practice. In R.E. Moore, editor, Reliability in Computing, volume 19 of Perspectives in Computing, pages 109-126. Academic Press, 1988.

[41] J. Vignes, Zéro mathématique et zéro informatique. *La Vie des Sciences, C.R. Acad. Sci.*, Paris, 4, 1, janvier 1987, pp. 1-13.

[42] J. Vignes, Implémentation des méthodes d'optimisation : test d'arrêt optimal, contrôle et précision de la solution (I) *R.A.I.R.O., 18, 1*, février 1984, pp. 1-18; (II), *R.A.I.R.O.18, 2*, mai 1984, pp. 103-129.

[43] R. Alt, Minimizing the error propagation in the numerical solution of ODEs Scientific Computing, *IMACS Transactions,* Vol.1, 1983, pp. 231-235.

[44] A. Feldstein, R. Goodman, Convergence estimates for the distribution of trailing digits, *Journal of A.C.M.,* vol. 23, 1976, pp.287-297.

[45] J. Vignes, M. La Porte, Error analysis in computing, *Information Processing 74*, North-Holland, 1974.

[46] R.W. Hamming, On the distribution of numbers, *The Bell System Technical Journal*, 1970, pp. 1609-1625.

[47] D.E. Knuth, The art of computer programming, 2. *Addison-Wesley series*, 1969.

[48] IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985, Institute of Electrical and Electronics Engineers, August, 1985, reprinted in SIGPLAN 22, 2, pp. 9-25.

Index

** function, 11
abs function, 10
acos function, 11
aimag function, 10
aint function, 10
anint function, 10
asin function, 11
atan2 function, 11
atan function, 11
cadna_end subroutine, 13, 18, 28
cadna_init subroutine, 12, 18
conjg function, 10
cosh function, 11
cos function, 11
data_st function, 9, 15, 20
data sections, 19
dble function, 10
dim function, 11
double_st, 9
exp function, 11
int function, 9
log10 function, 11
log function, 11
max function, 10, 22
min function, 10, 22
mod function, 11
nb_significant_digit function, 14
nint function, 9
old_type function, 14, 20
real function, 10
sign function, 11
single_st, 9
sinh function, 11
sin function, 11
sqrt function, 11
str function, 13, 21
tanh function, 11
tan function, 11
use CADNA, 18

CESTAC method, 5
computational zero, 8

discrete stochastic arithmetic, 5

numerical debugging, 28

printing statements, 21

random arithmetic, 8
reading statements, 21
relational operators, 11

statement functions, 22

vector operators, 20