CADNA for C/C++ 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

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 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 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 of the program and hence of 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 C or 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 analyzed 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 C++. This language enables to create new numerical **types** with their operators; furthermore the designating symbol of an operator can be chosen 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 C/C++ programs.
Thus a new numerical type has been created, the **stochastic number**; it is nothing else than an N-set ($N = 3$) containing perturbed floating-point values. All the arithmetic operators ($+$, $-$, $*$, $/$) have been overloaded 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. All standard functions defined in "math.h" (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 exposed in 2.5.6.

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*:

float_st	for stochastic variables in single precision stochastic type associated to float
double_st	for stochastic variables in double precision stochastic type associated to double

2.3 Intrinsic functions

We present here how the intrinsic functions defined in C have been extended for stochastic types.

2.3.1 Conversion functions

The float, double, long, unsigned, int and short cast operators:
They act on variables of stochastic type and work like for numerical predefined types. Thus the result is of classical type and the knowledge of the accuracy is lost.

If X is a stochastic variable consisting in N samples X_i, for instance

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

2.3.2 Numerical functions

The fabs function:
Given a float_st argument, this function returns a positive float_st value; given a double_st argument, it returns a positive double_st value.

The floor, ceil and rint functions:
These functions accept stochastic arguments and work like on the classical types.

The pow function:
This function accepts both classical or stochastic types. If at least one argument is a stochastic variable, the ouput value is of stochastic type. In this case, if one argument is a double or a double_st variable, the pow function returns a value of type double_st.

2.3.3 Mathematical functions

These are the following functions: sqrt, exp, log, log10, sin, cos, tan, asin, acos, atan, atan2, sinh, cosh, tanh, hypot. They accept arguments of float_st or double_st stochastic type. The output value has the same type as the argument. If the function has two arguments, they must be of the same stochastic type.

2.4 Relational operators

Comparison operators are overloaded and accept stochastic types and a mixture of classical types and stochastic types. They take into account the accuracy of the operands.
Thus when the expression a == 0.0 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 >= b is true, it means that

- a-b is a computational zero or

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

and, when the expression a > 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 classical C statements are slightly affected when using the CADNA tool. Now we present functions that are specific to the library. Note that the functions cadna_init and cadna_end have to appear, respectively to initialize and to close the library. The other functions cadna_enable, cadna_disable, computedzero, data_st, nb_significant_digit, str and strp will appear in some applications.

2.5.1 Initializing and closing the library

The cadna_init function has to be called once, early in the main program. This function 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 first M 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 7 integer parameters in the library:
CADNA_BRANCHING,
CADNA_CANCEL,
CADNA_DIV,

CADNA_INTRINSIC,
CADNA_MATH,
CADNA_MUL,
CADNA_POWER.

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.

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

The last 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 cadna_end function "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 string argument and a stochastic argument. It returns a pointer to the first argument. This output string contains the scientific notation of the stochastic 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 exact significant digit, the string that is returned is @.0.
The number of characters in the output string is:

$14 + 1$ for a float_st variable ;
$23 + 1$ for a double_st variable ;

+1 is needed by the "\0x0" character at the end of the string.
To avoid the use of the string parameter, a special implementation of the str function has been written. It must be used only with the family of printf

functions. The name of this new function is strp. Using this function, the allocation of the string is dynamically managed by the function itself. The only restriction is that it is not possible to have more than 256 calls to the strp function in one call to the printf function.
For C++ programmers, the classical cout and << notations have been overloaded for the stochastic types. No modification is needed.

Example:
Let us consider the following instructions:

```
int i;
double a;
...
printf("iteration %d a=%lf\n",i,a);
...
```

Using CADNA, the corresponding instructions in C can be:

```
int i;
double_st a;
char s[25];
...
printf("iteration %d a=%s\n",i,str(s,a));
...
```

Or using the strp function:

```
int i;
double_st a;
...
printf("iteration %d a=%s \n",i,strp(a));
...
```

In C++, it is simpler:

```
int i;
double_st a;
...
 cout << "iteration " << i << "a=" << a << endl;
...
```

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

The nb_significant_digit method returns an integer giving the number of exact significant decimal digits of a stochastic variable when the method is called. At some point x.nb_significant_digit() may return 7; later during the run

it may return 5. If x becomes non-significant then x.nb_significant_digit() returns 0.

2.5.4 Obtaining the triplet associated with a stochastic variable

The display method prints the triplet associated with a stochastic variable. For instance, let d be a double precision stochastic variable. The following instructions

```
printf("%s\n",strp(d));
d.display();
```

may provide

```
0.30E-064
+3.0217133019536030e-65 -- +3.0133146666062181e-65 -- +3.0248565827034563e-65
```

The three double precision values associated to d have 2 common significant digits.

The getx, gety and getz methods return the three values associated with a stochastic variable.
For instance, using the same stochastic variable d as above, the following instructions

```
double x,y,z;
x=det.getx();
y=det.gety();
z=det.getz();
cout << "x = " << x << endl;
cout << "y = " << y<< endl;
cout << "z = " << z << endl;
```

provide

```
x = 3.02171e-65
y = 3.01331e-65
z = 3.02486e-65
```

2.5.5 Testing if a variable is a computational zero

The computedzero method acts on a stochastic variable and returns 0 or 1. The computedzero method returns 1 if this stochastic variable is a computational zero, *i.e.* it is a mathematical zero or it has no exact significant digit.

2.5.6 Reducing accuracy of initial data

Initial data are often known with less significant digits than provided by their internal representation.The data_st method 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 method acts on a stochastic variable X and has two optional arguments: X.data_st(ERX,IER);
The first argument is an optional double argument that contains the relative or absolute uncertainty of the stochastic variable X. The second argument determines the kind of the uncertainty: relative or absolute. If X is a stochastic variable and ERX is a double value strictly less than 1, the X.data_st(ERX,IER); instruction 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 method without ERX must be used when data are considered as exact but cannot be exactly coded in the memory.

Chapter 3

User's 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 float or double 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,
- termination of the CADNA library.

The reader may refer to the sample program given in 3.8 with two versions, *i.e.* the initial C code and the code modified to be compiled with the CADNA library.

3.1 Declaration of the CADNA library

The following pseudo-statement

```
#include <cadna.h>
```

must take place in any file which contains declarations of stochastic variables or CADNA functions to be found by the compiler.

3.2 Initialization and termination of the CADNA library

The call to the cadna_init function must be added just after the **main program declaration statements** to initialize random arithmetic. For more information about the arguments of the cadna_init function, see 2.5.1.
The call to the cadna_end function 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 declarations	CADNA declarations
float a, b;	**float_st a, b;**
double c;	**double_st c;**
float d[6], e, f;	**float_st d[6], e, f;**

3.4 Changes in assignments or arithmetic operations

3.4.1 Conversions between usual types and stochastic types

In assignment statements, conversions are implicit from C float, int or double 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.**

With the following declarations:
float_st a, b;
float r;
the assignments a = r;, b = 2; and r = a; are correct but there is, of course, no information on the accuracy of r.

When a variable is set to a value which cannot be exactly coded, the data_st method should be used.

Example:

Initial C statements	Modified C statements for CADNA

float x, y;
x=1.234;
 	
y=-3.0;
	#include <cadna.h>
	float_st x, y;
	x=1.234;
	x.data_st();
	y=-3.0;

3.4.2 Classical 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, classical types and integer types.

With the following declarations:
float_st a, b;
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 classical types, double_st prevails over float_st.
So with the previous declarations, c = a * c + b * 3 needs no change.

3.5 Changes in reading statements

The family of scanf functions is adapted to classical floating-point variables, which must be transformed into stochastic variables.

Example:

Initial C statements	Modified C statements for CADNA
float x; ..... scanf("x = %14.7e \n", &x);	**#include <cadna.h>** **float xaux;** **float_st** x; ..... scanf("x = %14.7e \n", &**xaux**); **x = xaux;**

3.6 Changes in printing statements

Before printing each stochastic variable, it must be transformed into a string by the str or strp function. The required length is 15 for a float_st variable and 25 for a double_st variable. Therefore formats should be modified.

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

Initial C statements	Modified C statements for CADNA
float x; ... printf("x = %14.7e n", x);	**#include <cadna.h>** **float_st** x; ... printf("x = %s n", **strp(x)**);

3.7 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 C statements	Modified C statements for CADNA

float a;
 	
a=3.14*f(2.0);
...
 	
float f(x)
{
float x;
...
}
	#include <cadna.h>
	float_st aux, a;
	aux=2.0;
	a=3.14*f(aux);
	...
	
	float_st f(x)
	{
	float_st x;
	...
	}

3.8 A example of numerical code and its modified version

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

3.8.1 Standard C source code

```
#include <stdio.h>
#define N 4

// Initialization:
void InitMat(float M[N][N])
{int i,j;
 for(i=0;i<N;i++)
   for(j=0;j<N;j++) scanf("%e",&M[i][j]);
}

// Inversion using the Gauss-Jordan method:
void InvertMat(float M[N][N])
{int i,j,k;
 float temp;
 for(k=0;k<N;k++)
   {temp = M[k][k];
    M[k][k] = 1.0;
    for(j=0;j<N;j++) M[k][j]/=temp;
    for(i=0;i<N;i++)
      if(i!=k)
        {temp=M[i][k];
         M[i][k] = 0.0;
         for(j=0;j<N;j++) M[i][j] -= temp*M[k][j];
        }
   }
}

// Display of a matrix:
void DisplayMat(float M[N][N])
{int i,j;
 for(i=0;i<N;i++)
   {for(j=0;j<N;j++)
      printf("%14.7e  ",M[i][j]);
    printf("\n");
   }
}

void main()
```

```
{float M[N][N];
 printf("Initial matrix:\n");
 InitMat(M);
 DisplayMat(M);
 InvertMat(M);
 printf("Inverted matrix:\n");
 DisplayMat(M);
}
```

3.8.2 Source code using the CADNA library

```
#include <cadna.h>
#include <stdio.h>
#define N 4

// Initialization:
void InitMat(float_st M[N][N])
{float aux;
 int i,j;
 for(i=0;i<N;i++)
     for(j=0;j<N;j++) {scanf("%e",&aux); M[i][j] = aux;}
}

// Inversion using the Gauss-Jordan method:
void InvertMat(float_st M[N][N])
{int i,j,k;
 float_st temp;
 for(k=0;k<N;k++)
   {temp = M[k][k];
    M[k][k] = 1.0;
    for(j=0;j<N;j++) M[k][j]/=temp;
    for(i=0;i<N;i++)
      if(i!=k)
        {temp=M[i][k];
         M[i][k]=0.0;
         for(j=0;j<N;j++) M[i][j] = M[i][j] - temp*M[k][j];
        }
   }
}
```

```
// Display of a matrix:
void DisplayMat(float_st M[N][N])
{int i,j;
 for(i=0;i<N;i++)
   {for(j=0;j<N;j++)
      printf("%s  ",strp(M[i][j]));
    printf("\n");
   }
}

void main()
{cadna_init(-1);
 float_st M[N][N];
 printf("Initial matrix:\n");
 InitMat(M);
 DisplayMat(M);
 InvertMat(M);
 printf("Inverted matrix:\n");
 DisplayMat(M);
 cadna_end();
}
```

3.8.3 Example of execution without CADNA

```
 Initial matrix:
 1.0000000e+00   2.0000000e+03   5.0000000e-01   4.0000000e+00
 2.9999999e-05   1.0000000e+00   2.0000000e+00   8.0000000e+00
 4.0000000e+00   5.0000000e-01   2.9999999e-08   2.0000000e+00
 2.0000000e+00   3.0000000e+00   5.0000000e-01   5.0000000e+09
Inverted matrix:
-6.2576764e-05  -8.1558341e-05   2.5001565e-01  -9.9964599e-11
 5.0009380e-04  -1.2504448e-04  -1.2502252e-04  -1.4995290e-13
-2.5004597e-04   5.0006253e-01   5.8761423e-05  -7.9992352e-10
-2.4999515e-13  -4.9991469e-11  -9.9937121e-11   2.0000000e-10
```

3.8.4 Example of execution with CADNA

```
 Initial matrix:
 0.1000000E+01   0.1999999E+04   0.5000000E+00   0.4000000E+01
 0.2999999E-04   0.1000000E+01   0.2000000E+01   0.8000000E+01
```

```
 0.4000000E+01   0.5000000E+00   0.2999999E-07   0.2000000E+01
 0.2000000E+01   0.3000000E+01   0.5000000E+00   0.5000000E+10
Inverted matrix:
-0.62E-04   @.0             0.250015E+00  -0.10E-09
 0.5000938E-03  -0.124E-03  -0.1250225E-03  -0.14E-12
-0.250045E-03   0.500062E+00   0.5876140E-04  -0.799923E-09
-0.250E-12  -0.49E-10  -0.999371E-10   0.2000000E-09
```

3.9 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),
LOSS OF ACCURACY DUE TO 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 function, 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 power function 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**:
  - when using integer cast functions, the integral part of the argument can not be exactly determined due to the round-off error propagation;
  - in the fabs function: the argument is non-significant;
  - the floor, ceil or trunc function returns different values for each component of the stochastic argument.
- **loss of accuracy due to 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 prefix to specify (with an absolute path) which directory will contain the compiled library. For instance, you may choose the following option:

```
./configure --prefix=/home/personal/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 g++ (version 4.3) compiler on an Intel Centrino 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, only the exact significant digits appear in results. We recall that there is no guarantee of the last digit provided by CADNA.

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_C  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
------------------------------------------------------------------
----------------------------------------
|  Polynomial function of two variables  |
|  with CADNA                            |
----------------------------------------
res= @.0
------------------------------------------------------------------
CADNA_C  software --- University P. et M. Curie --- LIP6
```

```
There is  1 numerical instability
1 LOSS OF ACCURACY DUE TO 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:

```
---------------------------------
|  Second order equation         |
|  without CADNA                 |
---------------------------------
d = -2.861023e-06
There are two complex solutions.
z1 = +3.500000e+00  +  i * +8.457279e-04
z2 = +3.500000e+00  +  i * -8.457279e-04
```

and using CADNA in single precision, one obtains:

```
-----------------------------------------------------------------
CADNA_C  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
-----------------------------------------------------------------
---------------------------------
|  Second order equation         |
|  with CADNA                    |
---------------------------------
d =  @.0
Discriminant is zero.
The double solution is  0.3499999E+001
```

```
------------------------------------------------------------
CADNA_C  software --- University P. et M. Curie --- LIP6
There is  1 numerical instability
1 LOSS OF ACCURACY DUE TO 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:

```
-------------------------------------------------------
|  Computation of the determinant of Hilbert's matrix  |
|  using Gaussian elimination without CADNA            |
-------------------------------------------------------
Pivot number 0 = 1.000000000000000e+00
Pivot number 1 = 8.333333333333331e-02
Pivot number 2 = 5.555555555555526e-03
Pivot number 3 = 3.571428571428830e-04
Pivot number 4 = 2.267573696145566e-05
Pivot number 5 = 1.431549050529594e-06
Pivot number 6 = 9.009749264103679e-08
Pivot number 7 = 5.659971084095516e-09
Pivot number 8 = 3.551369635569034e-10
Pivot number 9 = 2.226762517485834e-11
Pivot number 10 = 1.399228241996033e-12
Determinant     = 3.028594438809703e-65
```

and using CADNA in double precision, one obtains:

```
------------------------------------------------------------
------------------------------------------------------
|  Computation of the determinant of Hilbert's matrix  |
|  using Gaussian elimination with CADNA               |
------------------------------------------------------
Pivot number 0 =  0.100000000000000E+001
Pivot number 1 =  0.833333333333333E-001
Pivot number 2 =  0.55555555555555E-002
Pivot number 3 =  0.357142857142E-003
Pivot number 4 =  0.22675736961E-004
Pivot number 5 =  0.143154905E-005
Pivot number 6 =  0.90097492E-007
Pivot number 7 =  0.5659970E-008
Pivot number 8 =  0.355134E-009
Pivot number 9 =  0.2226E-010
Pivot number 10 =  0.13E-011
Determinant      =  0.30E-064
------------------------------------------------------------
CADNA_C  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:

```
-------------------------------------
| A second order recurrent sequence |
| without CADNA                     |
-------------------------------------
U(3) = +5.590163934426237e+00
U(4) = +5.633431085044127e+00
U(5) = +5.674648620512615e+00
U(6) = +5.713329052423919e+00
U(7) = +5.749120920462043e+00
U(8) = +5.781810933690098e+00
U(9) = +5.811314466602178e+00
U(10) = +5.837660476543959e+00
U(11) = +5.861018785996283e+00
U(12) = +5.882524608269310e+00
U(13) = +5.918655323805488e+00
U(14) = +6.243961815306110e+00
U(15) = +1.120308737284091e+01
U(16) = +5.302171264499677e+01
U(17) = +9.473842279276452e+01
U(18) = +9.966965087355071e+01
U(19) = +9.998025776093678e+01
U(20) = +9.999882245337588e+01
U(21) = +9.999992970745579e+01
U(22) = +9.999999580049865e+01
U(23) = +9.999999974893262e+01
U(24) = +9.999999998498109e+01
U(25) = +9.999999999910112e+01
U(26) = +9.999999999994618e+01
U(27) = +9.999999999999677e+01
U(28) = +9.999999999999980e+01
U(29) = +9.999999999999999e+01
U(30) = +1.000000000000000e+02
The exact limit is 6.
```

and using CADNA in double precision, one obtains:

```
------------------------------------------------------------------
CADNA_C  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
-----------------------------------------------------------------
-----------------------------------
| A second order recurrent sequence |
| without CADNA                     |
-----------------------------------
U(3) =  0.55901639344262E+001
U(4) =  0.5633431085044E+001
U(5) =  0.567464862051E+001
U(6) =  0.57133290524E+001
U(7) =  0.574912092E+001
U(8) =  0.57818109E+001
U(9) =  0.5811314E+001
U(10) =  0.583765E+001
U(11) =  0.5860E+001
U(12) =  0.588E+001
U(13) =  0.59E+001
U(14) =  0.6E+001
U(15) =  @.0
U(16) =  @.0
U(17) =  @.0
U(18) =  0.9E+002
U(19) =  0.999E+002
U(20) =  0.9999E+002
U(21) =  0.99999E+002
U(22) =  0.999999E+002
U(23) =  0.99999999E+002
U(24) =  0.999999999E+002
U(25) =  0.9999999999E+002
U(26) =  0.99999999999E+002
U(27) =  0.999999999999E+002
U(28) =  0.99999999999999E+002
U(29) =  0.100000000000000E+003
U(30) =  0.100000000000000E+003
The exact limit is 6.
-----------------------------------------------------------------
CADNA_C  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)
2 UNSTABLE MULTIPLICATION(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:

```
-------------------------------------------------------------
|  Computation of a root of a polynomial by Newton's method |
|  without CADNA                                            |
-------------------------------------------------------------
x( 66) = +4.285714325273430e-01
x( 67) = +4.285714325273430e-01
```

and using CADNA in double precision, one obtains:

```
-----------------------------------------------------------------
CADNA_C  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
---------------------------------------------------------------
---------------------------------------------------------
|  Computation of a root of a polynomial by Newton's method |
|  with CADNA                                               |
---------------------------------------------------------
x(100) =  0.4285714E+000
x(101) =  0.4285714E+000
---------------------------------------------------------------
CADNA_C  software --- University P. et M. Curie --- LIP6

CRITICAL WARNING: the self-validation detects major problem(s).
The results are NOT guaranteed.

There are 480 numerical instabilities
75 UNSTABLE DIVISION(S)
74 UNSTABLE BRANCHING(S)
57 UNSTABLE INTRINSIC FUNCTION(S)
274 LOSS OF ACCURACY DUE TO 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_C  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
-------------------------------------------------------------
-------------------------------------------------------------
|  Computation of a root of a polynomial by Newton's method |
|  with optimisation and CADNA                               |
-------------------------------------------------------------
x( 47) =  0.428571428571429E+000
x( 48) =  0.428571428571428E+000
-------------------------------------------------------------
CADNA_C  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:

```
-----------------------------------------------------
| Solving a linear system using Gaussian elimination |
| with partial pivoting                              |
-----------------------------------------------------
x_sol(0) = +6.261988e+01 (exact solution: xsol(0)= +1.000000e+00)
x_sol(1) = -8.953979e+00 (exact solution: xsol(1)= +1.000000e+00)
x_sol(2) = +0.000000e+00 (exact solution: xsol(2)= +1.000000e-08)
x_sol(3) = +1.000000e+00 (exact solution: xsol(3)= +1.000000e+00)
```

and using CADNA in single precision, one obtains:

```
------------------------------------------------------------
CADNA_C  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
------------------------------------------------------------
------------------------------------------------
| Solving a linear system using Gaussian elimination |
| with partial pivoting                            |
------------------------------------------------
x_sol(0) =  0.99E+000 (exact solution: xsol(0)=  0.1000000E+001)
x_sol(1) =  0.100E+001 (exact solution: xsol(1)=  0.1000000E+001)
x_sol(2) =  0.999999E-008 (exact solution: xsol(2)=  0.1000000E-007)
x_sol(3) =  0.100000E+001 (exact solution: xsol(3)=  0.1000000E+001)
------------------------------------------------------------
CADNA_C  software --- University P. et M. Curie --- LIP6
There are 2 numerical instabilities
1 UNSTABLE BRANCHING(S)
1 LOSS OF ACCURACY DUE TO 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 3.9) can be performed in order to identify which instructions are responsible for instabilities. As described in 3.9, 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, instability (unstab=0x80ec298)
at cadna_unstab.cc:40
40   (*unstab)++;
```

```
(gdb)  > > >#0  instability (unstab=0x80ec298)
at cadna_unstab.cc:40
#1  0x080498f3 in operator- (a=@0xbf854a58, b=@0xbf854b58)
at cadna_sub.cc:287
#2  0x0804943c in main () at ex6_cad.cc:51

Breakpoint 1, instability (unstab=0x80ec294)
at cadna_unstab.cc:40
40   (*unstab)++;
#0  instability (unstab=0x80ec294) at cadna_unstab.cc:40
#1  0x0805b417 in operator> (a=@0xbf854b98, b=@0xbf854bb8)
at cadna_gt.cc:123
#2  0x0804918b in main () at ex6_cad.cc:32
```

From the *gdb.out* file, the first instability is caused by the instruction located at line 51 in the source file *ex6_cad.cc*. 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 LOSS OF ACCURACY DUE TO CANCELLATION(S)
```

The second instability is caused by the instruction located at line 32 in the source file:

```
if(fabsf(a[j][i])>pmax){
```

This instability is due to the $>$ 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=6.83561e+5;
y=6.83560e+5;
z=1.00000000007;
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 2.32830643653870E-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_C 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 = 24666
------------------------------------------------------------
CADNA_C software --- University P. et M. Curie --- LIP6
```

```
There are 300000 numerical instabilities
300000 LOSS OF ACCURACY DUE TO 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:

```
#include "stdio.h"
#include <math.h>
#include "cadna.h"
main()
{
  cadna_init(-1);
  double_st r,r1,x,y,z;
  int i, nloop, ierr;
  printf("Enter the number of iterations: ");
  scanf("%d",&nloop);
  ierr = 0;
  for(i=0;i<nloop;i++){
    x=6.83561e+5;
    y=6.83560e+5;
    z=1.00000000007;
    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 != 1.4e-10) ierr = ierr + 1;
  }
  printf("r = %s   ierr = %d \n", strp(r) ,ierr);
  cadna_end();
}
```

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.