Numerical 'health check' for scientific codes: the CADNA approach

N.S. Scott [a,1], F. Jézéquel [b], C. Denis [b] and J.-M. Chesneaux [b]

[a] *School of Electronics, Electrical Engineering & Computer Science, The Queeen's University of Belfast, Belfast BT7 1NN, UK*

[b] *Laboratoire d'Informatique de Paris 6, Université Pierre et Marie Curie - Paris 6, 4 place Jussieu, 75252 Paris Cedex 05, France*

Abstract

Scientific computation has unavoidable approximations built into its very fabric. One important source of error that is difficult to detect and control is round-off error propagation which originates from the use of finite precision arithmetic. We propose that there is a need to perform regular numerical 'health checks' on scientific codes in order to detect the cancerous effect of round-off error propagation. This is particularly important in scientific codes that are built on legacy software. We advocate the use of the CADNA library as a suitable numerical screening tool. We present a case study to illustrate the practical use of CADNA in scientific codes that are of interest to the Computer Physics Communications readership. In doing so we hope to stimulate a greater awareness of round-off error propagation and present a practical means by which it can be analyzed and managed.

PACS: 02.60.-x; 02.60.Jh; 02.70.-c; 02.50.Ey.

Key words: atomic collision processes; CADNA; CESTAC method; Clebsch-Gordan coefficient; discrete stochastic arithmetic; dynamical control; floating-point arithmetic; quadrature; round-off error propagation; Racah coefficient; R-matrix; Slater integrals.

1 Introduction

Computer Physics Communications has, since its inception over thirty-five years ago, had a particular focus on the publication of descriptions of computer

[1] Corresponding author: ns.scott@qub.ac.uk

programs. Indeed, in the journal's first paper [1], published in 1969 and entitled "The publication of scientific Fortran programs", Keith Roberts [2] proposed a collection of general principles that he regarded as necessary if an international library of published scientific programs was to be successfully established. It was largely on these principles that the CPC Program Library was founded and more than three decades later, with a library of over 2,000 published programs, it continues to develop and expand.

The principles elucidated were practical and, for the most part, remain valid today. They require, for example, that published programs should be intelligible, portable, efficient, adaptable, modular and properly documented. However, one principle, *verification*, remains elusive. Robert's suggestion, that "*The scientific community should always be able to verify that a published program will produce correct results, or that a published calculation is correct, in the same way that it can check the truth of scientific theorems or experimental measurements.*", is a tall order for scientific software. Indeed the principle, as stated, is ambiguous since it needs to be clarified whether the results are acceptable in relation to the real world, the mathematical model or the computational model.

Many CPC programs are large software systems developed to enable virtual experiments to be conducted on some physical system. The development process, extending from the physical world to the mathematical model, then to the computational model and finally to the computer implementation, involves a number of approximations: physical effects may be discarded, continuous functions replaced by discretized ones and real numbers replaced by finite precision representations. In consequence, approximation is woven into the very fabric of scientific software and cannot be eliminated. Unfortunately, despite developments in software engineering, there is every reason to believe that the comment made by Leslie Fox in 1971 [2] is still valid today, "*I have little doubt that about 80 per cent of all the results printed from the computer are in error to a much greater extent than the user would believe ...*". It is incumbent, therefore, on the computational scientist to understand the source and propagation of these errors and to manage them judiciously. The theme of accuracy and reliability in scientific computation has recently been explored and is amplified in [3].

One important source of error that is both esoteric and difficult to manage originates from the use of finite precision arithmetic. It is well known that the floating-point arithmetic commonly used in scientific computing only approximates exact arithmetic. Arithmetic expressions are no longer associative, commutative and distributive. More important, the evaluation of most arith-

[2] who was a founding Specialist Editor of CPC and its Principal Editor from 1980 until his death in 1985.

metic expressions generates a round-off error. It is not uncommon to find that the same code, using the same data, produces different results when executed on different platforms [3]. Indeed, the same code, using the same data, on the same computer can produce different results if the rounding mode is changed. Potentially these errors can occur with each variable assignment and for each arithmetic operation. This problem is exacerbated in a supercomputing environment where trillions of floating-point operations may be performed every second and in a Grid environment where the overall computation may involve contributions from many heterogeneous platforms each with potentially different round-off characteristics.

The propagation of these errors must be addressed to avoid the production of computed results with few or no significant digits. In particular, numerical verification is required to give confidence that the computed results are acceptable. However, in the absence of suitable tool support this is a practical impossibility for large codes and refuge is normally taken in the unstable shelter of extended precision, in the hope that accuracy will be maintained. The following problem proposed by S. M. Rump [4] demonstrates the precariousness of this position.

Consider the following innocuous looking function,

$$f(x, y) = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y), \qquad (1)$$

to be evaluated at $x = 77617$ and $y = 33096$.

A Fortran implementation of Eq. (1), executed on an Itanium processor using the Intel ifort compiler(v9.0.021), in single, double and quadruple precision produces the results shown in Table 1.

The first three entries might lead the unwary to conclude that the single precision result is incorrect and that the double precision result is accurate to 14 decimal digits. In fact all three results are incorrect, *not only in the first digit but also in the sign*. The correct result, which lies within the small interval displayed in the fourth entry of Table 1, was obtained using variable precision interval arithmetic (VPIA) [5] with about 40 decimal digits of accuracy.

This example illustrates that round-off error can seriously compromise the reliability of a fixed precision floating-point computation. Further, an indication of the seriousness of the error cannot always be obtained simply by observing floating-point results at increased precision. For large scientific programs, such

[3] even when each platform conforms to the IEEE 754 floating-point standard. Variations can occur because of compiler optimization, use of extended precision and behaviour of intrinsic functions.

Table 1
The computation of $f(x,y) = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$ using Fortran, variable precision interval arithmetic, and CADNA.

Method	$f(77617, 33096)$
$Fortran$: *single precision*	6.3382530×10^{29}
$Fortran$: *double precision*	1.17260394005318
$Fortran$: *quad precision*	1.172603940053178631858834904520 18
$VPIA$	$[-0.827396059946821368141165095479816292005,$
	$-0.827396059946821368141165095479816291986]$
$CADNA$: *single precision*	@.0
$CADNA$: *double precision*	@.0

as those published in Computer Physics Communications, *a numerical 'health check' is strongly recommended*. This is particularly important in codes that rely on legacy software that has outlived its original hardware and its creators. Often when available computing resources are increased the computational scientist increases the problem size perhaps, inadvertently, rendering legacy routines unfit for purpose because of the unseen, cancerous effect of round-off. Ideally, we seek a numerical screening tool that will:

- report gradual and catastrophic loss of precision;
- report the accuracy of intermediate and final results;
- be of acceptable efficiency; and
- be non invasive to the source code.

The terms *accuracy* and *precision* are frequently used inconsistently. It is appropriate therefore to clarify what we mean by them. In general terms, the *accuracy* of a computation determines how close the computation comes to the true mathematical value. It indicates, therefore, the correctness of the result. The *precision* of a computation reflects the exactness of the computed result without reference to the meaning of the computation. It is, therefore, the number of significant digits not affected by round-off error. For example, the number, 3.1428571 has eight decimal digit precision, irrespective of what it represents. If this number represents 22/7 then it is also accurate to eight decimal digits but if it represents the irrational number π then it is accurate only to three decimal digits.

Several methods and tools have been developed over the years to analyze round-off error propagation. These include direct analysis [6], inverse analysis [7], methods based on algorithmic differentiation [8], interval arithmetic [9], Monte Carlo arithmetic [10] and tools such as PRECISE (PRecision Estimation and Control In Scientific and Engineering computing) [11].

Each of these methods and tools has strengths and weaknesses and each has made an important contribution to the field. However, it is not the purpose of this paper to compare and contrast them. Rather we focus on the CESTAC method (Contrôle et Estimation Stochastique des Arrondis de Calculs) [12] and its associated tool, CADNA (Control of Accuracy and Debugging for Numerical Applications) [13]: a library designed to assist in numerical verification and, in particular, to estimate precisely the computing error in computer generated results i.e. to estimate of the number of common significant figures between the computed result and the exact result [4]. The use of CADNA is illustrated in the last two lines of Table 1. When the CADNA version of the Fortran program is executed it outputs a computational zero, denoted by @.0, in both single and double precision. This indicates clearly, that in both cases, the answer contains no digits that are not affected by round-off, *the result is just numerical noise*.

An objective of this paper is to stimulate a greater awareness of round-off error propagation in scientific computation and to illustrate a practical means by which it can be analyzed and managed. The main thrust of the paper is presented in §3 where we illustrate how CADNA can be used to perform a numerical 'health check' on scientific codes of the type published in Computer Physics Communications. We go beyond the numerical 'health check' in §4 to demonstrate how CADNA can be used in a proactive way to compute benchmark results by controlling both the truncation error of the approximate numerical method and the round-off error. Finally, in §5 we assess CADNA against the screening tool criteria mentioned earlier in this section.

To use any tool effectively the user must have some appreciation of the ideas that underpin it. For this reason we begin by presenting background material on the philosophy behind CADNA.

2 CADNA

The CADNA library is an implementation of discrete stochastic arithmetic (DSA), which is based on the CESTAC method. Before describing the CADNA library it is necessary to provide some background by giving a brief outline of the CESTAC method and DSA from the perspective of a CADNA user.

[4] A new open source version of the CADNA library is being prepared for publication in Computer Physics Communications.

Where no overflow occurs, the exact result, r, of any non exact floating-point arithmetic operation is bounded by two consecutive floating-point values R^- and R^+.

The basic idea of the method is to perform each arithmetic operation N times, randomly rounding each time, with a probability of 0.5, to R^- or R^+. The computer's deterministic arithmetic, therefore, is replaced by a stochastic arithmetic where each arithmetic operation is performed N times before the next arithmetic operation is executed, thereby propagating the round-off error differently each time. This is the essence of the CESTAC method [14,15]. The method furnishes us with N samples, R_i, of the computed result R. The value of the computed result, $\overline{R}$, is the mean value of $\{R_i\}$ and the number of exact significant digits in $\overline{R}$, $C_{\overline{R}}$, is estimated using the mean value and the standard deviation of $\{R_i\}$.

It has been proved that the validity of $C_{\overline{R}}$ is compromised if the two operands in a multiplication or the divisor in a division are not significant. *It is essential, therefore, that any computed result with no significance is detected and reported, since its subsequent use may invalidate the method.* The need for this dynamical control of multiplications and divisions has led to the concept of the computational zero [16].

A computed result, R, is a computational zero, denoted by @.0, if and only if one of the following two conditions holds:

(1) $\forall i, R_i = 0 \quad i = 1, \ldots, N$
(2) $C_{\overline{R}} \leq 0$

This means that a computational zero is either a mathematical zero or a number without any significance *i.e.* numerical noise.

To establish consistency between the arithmetic operators and the relational operators discrete stochastic relations are defined as follows. Let X and Y be two computed results. Then discrete stochastic relations are defined as:

$$X = Y \quad if X - Y = @.0, \tag{2}$$
$$X > Y \quad if \overline{X} > \overline{Y} \quad and\ X - Y \neq @.0, \tag{3}$$
$$X \geq Y \quad if \overline{X} \geq \overline{Y} \quad or\ X - Y = @.0. \tag{4}$$

These definitions take into account the numerical noise and allow the recovery of some coherence between relational and arithmetic operations [17,18].

Discrete Stochastic Arithmetic (DSA) is simply the combination of the CESTAC method, the concept of the computational zero and the discrete stochastic relationships.

2.2 The CADNA library

The CADNA library is an implementation of DSA devoted to programs written in ADA, C, C++ and Fortran. In this paper we focus on the Fortran implementation which is a set of data types, functions and subroutines that may be easily incorporated into any Fortran program. In essence, Fortran types are replaced by the corresponding stochastic types. A stochastic number is an N-dimensional set containing the perturbed floating-point values. In practice, the floating-point values R^- and R^+ are obtained using the rounding modes towards $-\infty$ and $+\infty$ defined in the IEEE 754 standard [19]. N is set to 3 with the first two rounding modes chosen at random and the third set to the opposite of the second. Interested readers are advised to consult [20] where the choices concerning the value of N and the random rounding mode are justified. Arithmetic operators, logical operators, all the standard intrinsic functions [5] and some vector operations have been overloaded so that when an operator is used the operands are triplets and the returned result is a triplet. The print statement has been modified, through a special function `str`, to output the computed result, $\overline{R}$, only with its exact number of significant digits, $C_{\overline{R}}$. A related special function, `cestac`, takes a stochastic argument and returns an integer giving the current number of exact significant digits in the stochastic argument. Input data and variables that are set to literal values will often not be exact floating-point values e.g. $x = 0.1$. Such values can, *and should*, be perturbed using the CADNA function `data_st(x)`.

During execution, when a numerical anomaly is detected, dedicated CADNA counters are incremented. At the end of the run, the value of these counters together with appropriate warning messages are printed on standard output. These warnings are of two types.

(1) Warnings concerning the validity of the CADNA results. These include: unstable multiplication where the two operands are computational zeroes; unstable power, where the operand of the power function is a computational zero; and unstable division were the divisor is a computational zero.
(2) Warnings concerning numerical instabilities. These instabilities can occur in overloaded functions such as SIGN, MOD, DIM, LOG, SIN or in branching statements involving a computational zero. A numerical insta-

[5] as defined by the F77 standard.

bility is also reported in the case of a cancellation, *i.e.* the subtraction of two very close values which generates a sudden loss of precision.

At the end of the run, each type of anomaly together with their occurrences are printed. If no anomaly has been detected the computed results are reliable and the precision of each has been correctly estimated up to a certain probability. If anomalies have been detected two cases need to be considered. Warnings of type (1) indicate that the validity of $C_{\overline{R}}$ has been compromised and the CADNA results cannot be relied on. Warnings of type (2) means that numerical instabilities have been detected. In both cases the messages need to be analyzed, the source of the anomaly identified and, if necessary, the code changed.

Numerical debugging can be performed using a symbolic debugger such as `gdb`. A breakpoint needs to be set to detect each call of the CADNA instability function. The name of this function depends on the system and the compiler. Under Unix this name can be detected using the following statement.

```
nm 'name_of_binary' | grep unstability
```

Using `gdb` under Linux the following statement will output a trace of all instabilities to the file `gdb.out`.

```
gdb 'name_of_binary' < gdb.in > gdb.out &
```

where `gdb.in` contains

```
break unstability_
run
while 1
where
cont
end
```

In these instructions `where` prints a complete trace of the instability that stopped the run and `cont` causes execution to resume.

Normally the use of the CADNA library in a Fortran program involves the following six steps.

(1) Declaration of the CADNA library to the compiler via `use cadna`. This use statement is required after each `PROGRAM`, `MODULE`, `SUBROUTINE` and `FUNCTION` statement.
(2) Initialization of random arithmetic and other internal parameters of CADNA via `call cadna_init`. This is called once after the main program declaration statements. The first argument of the `cadna_init` func-

tion is the maximum number of instabilities which will be detected. The other arguments are optional.

(3) Substitution of the type `REAL` and `DOUBLE PRECISION` by `type(single_st)` and `type(double_st)`, respectively.

(4) Use of the function `data_st` to perturb input data if required. Care must also be taken not to read floating-point file data directly into stochastic variables which accommodate three file values. Rather the file data must be read into an appropriate Fortran type such as `REAL` and then assigned to the stochastic variable.

(5) Change of output statements to print stochastic results to exact precision using `str`.

(6) Print the results of the anomaly detection via `call cadna_end()`.

These steps are illustrated below for a double precision Fortran implementation of a program to evaluate of the function given in Eq. (1). The CADNA statements are shown as comments.

```
PROGRAM f
  !use cadna
  double precision  :: y,x,res   !type(double_st)  :: y,x,res
  !call cadna_init(100)
  x=77617d0
  y=33096d0
  res=333.75*y*y*y*y*y*y*y+x*x*(11*x*x*y*y-y*y*y*y*y*y*y- &
      121*y*y*y*y-2.0)+5.5*y*y*y*y*y*y*y*y*y+x/(2*y)
  print *, res    ! print *, str(res)
  ! call cadna_end()
END PROGRAM f
```

In this case, the `data_st` function is not used to perturb x and y because the chosen values are exactly represented.

3 Numerical 'health check' case study

We now present our experience of using the CADNA library to give a numerical 'health check' to 2DRMP. This suite of two-dimensional R-matrix propagation programs [21,22] is aimed at creating virtual experiments on HPC [23,24] and Grid architectures [25,26] in order to study electron scattering from H-like atoms and ions at intermediate energies.

The essence of the technique is as follows. The two-electron configuration space (x_1, x_2) is divided into square sectors as illustrated in Fig.1. The two-electron wavefunction describing the motion of the target electron and the

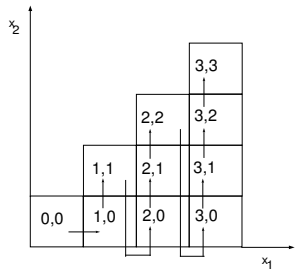

Fig. 1. Subdivision of the configuration space (x_1, x_2) into a set of connected sectors.

colliding electron is expanded within each sector in terms of one-electron basis functions, $\{P_{n,l}\}$, that are eigenfunctions of the Schrödinger equation, solved subject to certain fixed boundary conditions. The expansion coefficients are determined by diagonalizing the corresponding Hamiltonian matrix. The R-matrix may then be propagated across the sectors at each scattering energy and the scattering properties of interest determined.

A critical component in the 2DRMP suite involves the construction of the Hamiltonian matrices in each sector. In the case of diagonal sectors, computation of the two-electron part of the Hamiltonian matrix represents a significant bottleneck [27]. This part of the Hamiltonian involves sums of products of two dimensional radial integrals, termed a Slater integrals, and angular integrals.

Like many scientific codes 2DRMP is built on legacy routines developed over many years. In this case use is made of radial and angular integral routines developed for and used in the general R-matrix packages over the past three decades [28–33].

Thus, given the importance of the radial and angular legacy routines and their use in a new context we have subjected them to a CADNA 'health check'. In addition, since the radial integrals can only be computed using an approximate quadrature method and the angular integrals are in closed form this allows us to investigate the application of CADNA to approximate and finite methods respectively. We begin by examining the radial integrals.

3.1 CADNA analysis of the radial integrals

Given solutions to the radial Schrödinger equation,

$$\frac{dP_{n,l}(x)}{dx^2} = \left(\frac{l(l+1)}{x^2} - \frac{2Z}{x} - k_{n,l}^2 \right) P_{n,l}(x), \quad x \in [a, b], \tag{5}$$

subject to R-matrix boundary conditions [27], where n, l and Z represent the principal quantum number, the orbital angular momentum (a nonnegative

integer) and the electronic charge, respectively, the 2DRMP Slater integrals take the form,

$$I_\lambda = J_{1,\lambda} + J_{2,\lambda}, \tag{6}$$

with

$$J_{1,\lambda} = \int_a^b \int_a^y f_\lambda(x,y)dxdy, \tag{7}$$

$$f_\lambda(x,y) = \frac{P_{n_1,l_1}(y)P_{n_3,l_3}(y)}{y^{\lambda+1}} x^\lambda P_{n_2,l_2}(x)P_{n_4,l_4}(x), \quad x \in [a,y], \tag{8}$$

$$J_{2,\lambda} = \int_a^b \int_y^b \phi_\lambda(x,y)dxdy, \tag{9}$$

$$\phi_\lambda(x,y) = P_{n_1,l_1}(y)P_{n_3,l_3}(y) y^\lambda \frac{P_{n_2,l_2}(x)P_{n_4,l_4}(x)}{x^{\lambda+1}}, \quad x \in [y,b], \tag{10}$$

and

$$max(|l_1 - l_3|, |l_2 - l_4|) \leq \lambda \leq min(l_1 + l_3, l_2 + l_4). \tag{11}$$

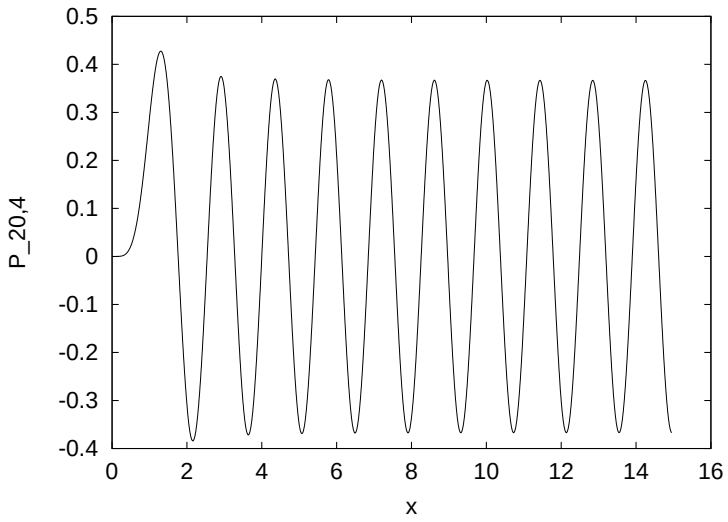

Fig. 2. The function $P_{20,4}(x)$.

To compute the Slater integrals, 2DRMP uses a subroutine based on `subroutine rs` taken from [29] [6] . To perform an initial numerical 'health check' on `subroutine rs` involves instrumenting it with CADNA, following steps 1-6 as described in §2.2, and observing the output.

[6] This subroutine uses Simpson's Rule to compute the outer integral of Eqs. (7) and (9) and a combination of the Trapezoidal Rule and Simpson's Rule to compute the inner integral of Eqs. (7) and (9).

Table 2
The computation of I_λ with $\lambda \in \{0, 2, 4, 6, 8\}$ in double precision for the case $a = 10^{-5}$, $b = 15.0$, $n_1 = 20$, $l_1 = 4$, $n_2 = 20$, $l_2 = 4$, $n_3 = 20$, $l_3 = 4$, $n_4 = 20$, $l_4 = 4$ with CADNA and using 1025 equally spaced integration points. The method to compute I_λ is described in §3.1.1 while the improved method to compute I_λ is described in §3.1.2.

λ	I_λ	I_λ improved
0	$0.1247937243912E+000$	$0.\mathbf{124793}6614595E+000$
2	$0.471551365578E-001$	$0.\mathbf{471551}1531988E-001$
4	$0.288813766E-001$	$0.\mathbf{288813}6383162E-001$
6	$0.952431E-002$	$0.\mathbf{209343}0472201E-001$
8	$0.3995087E+002$	$0.\mathbf{164875}4288096E-001$

I_λ with $\lambda \in \{0, 2, 4, 6, 8\}$ was computed in double precision for the case $a = 10^{-5}$, $b = 15.0$, $n_1 = 20$, $l_1 = 4$, $n_2 = 20$, $l_2 = 4$, $n_3 = 20$, $l_3 = 4$, $n_4 = 20$, $l_4 = 4$, using 1025 equally spaced integration points[7]. This is a typical but modest computation described in [27]. The function $P_{20,4}$ is a smoothly varying function as shown in Fig. 2 and thus can be assumed to be computed accurately to required precision.

No anomaly was found, indicating that the stochastic values throughout the code are reliable and the precision of each has been correctly estimated. The computed values of I_λ are displayed in the second column of Table 2. Recalling that CADNA only prints out those significant digits not affected by round-off, this initial screening shows a loss of precision in I_λ for larger values of λ, suggesting a round-off problem somewhere within `subroutine rs`.

To investigate further we used the CADNA `cestac` function to detect whether there was any loss of precision in the intermediate results. This indicated that, for the larger values of λ, there is a significant loss of precision in the inner integral of $J_{2,\lambda}$ as y increases.

On examining the inner integral for the final case, $J_{2,8}$,

$$g(y) = \int_y^{15.0} \frac{P_{n_2,l_2}(x) P_{n_4,l_4}(x)}{x^9} dx, \tag{12}$$

we observe the strange behaviour depicted in Fig. 3. The upper graph of Fig. 3 displays the number of significant digits in $g(y)$ while the lower graph displays $g(y)$. In both figures, beyond approximately $y = 3$, the even points follow the

[7] The number of integration points is not chosen on any rigorous criterion but on experience. It represents a compromise between accuracy and efficiency: 1025 points is expected to give a result accurate to about six figures. Doubling the number of points gives one extra digit.

Table 3
The computed value of $g(y) = \int_y^{15.0} \frac{P_{n_2,l_2}(x)P_{n_4,l_4}(x)}{x^9}dx$ at selected values of y.

i	$y_i = a + (i-1)h$	$g(y_i)$, using Eqs. (14-16)	$g(y_i)$, using Eqs. (17-19)
380	5.55176411132801	$0.2179550E-006$	$0.104153303371283E-007$
381	5.56641253906239	$0.103235E-007$	$0.103234799220457E-007$
382	5.58106096679676	$0.2177518E-006$	$0.102120637691204E-007$
383	5.59570939453114	$0.100807E-007$	$0.100807385691787E-007$
...	...	...	...
746	10.9130886621089	$0.2075749E-006$	$0.35254005852359E-010$
747	10.9277370898432	$0.3474E-010$	$0.34742269862922E-010$
748	10.9423855175776	$0.2075739E-006$	$0.34293770930925E-010$
749	10.9570339453120	$0.3391E-010$	$0.33907324325529E-010$
...	...	...	...
1018	14.8974610058584	$0.2075400E-006$	$0.345211114012527E-012$
1019	14.9121094335928	$0.30E-012$	$0.300127939842871E-012$
1020	14.9267578613272	$0.2075399E-006$	$0.252902341589428E-012$
1021	14.9414062890616	$0.2E-012$	$0.204059815759057E-012$

upper dashed line while the odd points follow the lower solid one. Furthermore, as y increases, the even points are computed to higher precision than the odd ones, as shown by column three of Table 3. In fact, as the value of y approaches 15.0 *the odd points have only a single digit that is not affected by round-off.*

This might lead one to conclude that the even points are more accurate than the odd ones. This, however, is the converse of the case and is a telling demonstration of the difference between *accuracy* and *precision* that was alluded to in §1. Numerical quadrature is an approximate method that is affected both by a truncation error in the method and by round-off error. These two sources of error interfere differently in the computation of the even and the odd points. As we shall see in the following section the truncation error is worse in even points than that in odd points, but the round-off error is worse in odd points than in even points. The odd points are therefore computed accurately to low precision while the even points are computed inaccurately to higher precision.

While the values in Table 3 are small in magnitude we shall find in §3.1.2 that their behaviour is indicative of a catastrophic error in the value of I_8.

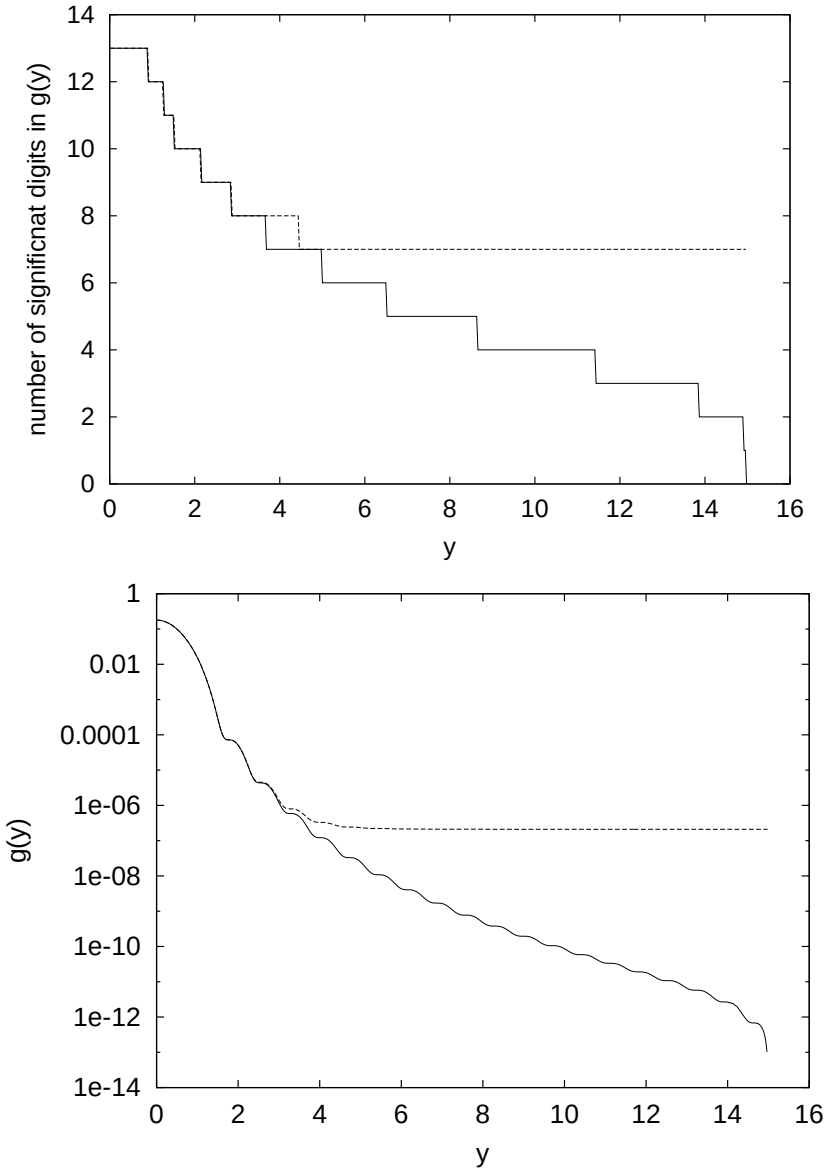

Fig. 3. The inner integral of $J_{2,8}$, $g(y) = \int_y^{15.0} \frac{P_{n_2,l_2}(x)P_{n_4,l_4}(x)}{x^9}dx$. The upper graph displays the number of significant digits in $g(y)$ and the lower graph displays $g(y)$ using a log scale on the y axis. In both graphs the even points are indicated by the dashed line and odd points by the solid line.

3.1.1 Legacy algorithm to compute $g(y) = \int_y^b \frac{P_{n_2,l_2}(x)P_{n_4,l_4}(x)}{x^{\lambda+1}}dx$

The CADNA 'health check' has led us to conclude that there is a round-off problem associated with the computation of $g(y)$ defined by Eq. (12). To see why, we need to analyse the legacy algorithm used by `subroutine rs`.

In the 1970s, for reasons of storage economy and computational efficiency, inner integrals such as Eq. (12) were replaced by their mathematically equivalent form,

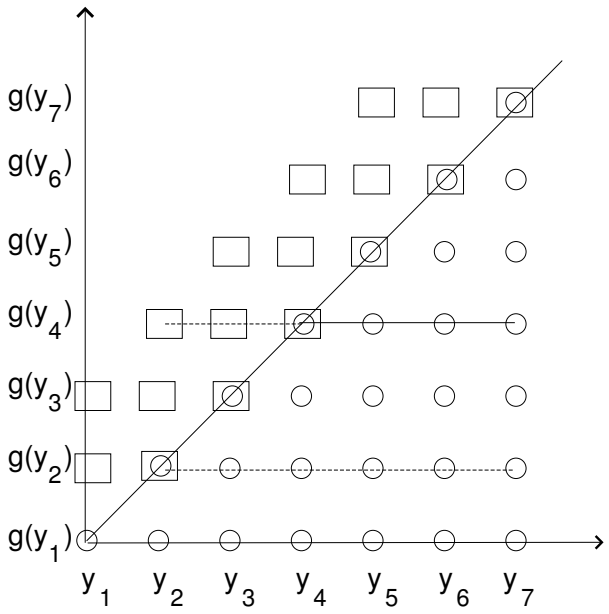

Fig. 4. $g(y_4) = g(y_2) - \int_{y_2}^{y_4}$. Here $g(y_4)$ is represented by the solid horizontal line through the circles, $g(y_2)$ is represented by the dashed line through the circles, and $\int_{y_2}^{y_4}$ is represented by the dashed line through the rectangles.

$$g(y) = \int_a^b \frac{P_{n_2,l_2}(x)P_{n_4,l_4}(x)}{x^{\lambda+1}}dx - \int_a^y \frac{P_{n_2,l_2}(x)P_{n_4,l_4}(x)}{x^{\lambda+1}}dx, \tag{13}$$

and computed as follows,

$$g(y_1) = \int_{y_1}^b \frac{P_{n_2,l_2}(x)P_{n_4,l_4}(x)}{x^{\lambda+1}}dx, \tag{14}$$

$$g(y_2) = g(y_1) - \int_{y_1}^{y_2} \frac{P_{n_2,l_2}(x)P_{n_4,l_4}(x)}{x^{\lambda+1}}dx, \tag{15}$$

$$g(y_i) = g(y_{i-2}) - \int_{y_{i-2}}^{y_i} \frac{P_{n_2,l_2}(x)P_{n_4,l_4}(x)}{x^{\lambda+1}}dx \quad 3 \leq i \leq k, \tag{16}$$

with $y_k = b$ and y_1 set to a. The integral in Eq. (14) is approximated using a composite Simpson's Rule, the integral in Eq. (15) is approximated using a two-point Trapezoidal Rule and the integral in Eq. (16) is approximated using a three-point Simpson's Rule. This scheme is illustrated in Fig. 4 for $g(y_4)$.

By examining the integrand of Eq. (12), $\frac{P_{n_2,l_2}(x)P_{n_4,l_4}(x)}{x^9}$, which is plotted in Fig. 5, the incorrect values of the even points in Fig. 3 can be seen to be influenced by the use of the two-point Trapezoidal Rule to compute $g(y_2)$. This, as can been seen from Fig. 5, is where the integrand climbs very steeply. Furthermore, by examining the components of Eq. (16) using the `cestac` function we can

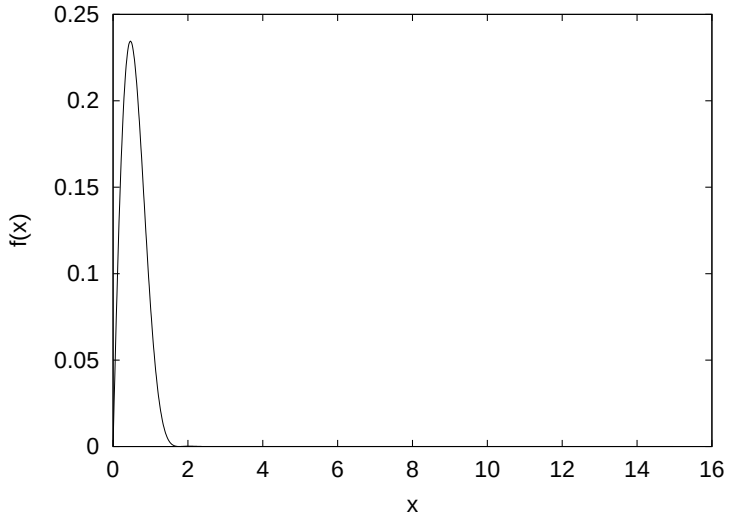

Fig. 5. The integrand of the inner integral of $J_{2,8}$: $f(x) = \frac{P_{20,4}P_{20,4}}{x^9}$.

see that, fortuitously, as y increases, the even points suffer less cancellation during the subtraction than the odd points.

In summary, the even points of $g(y)$ result from a poor algorithm that fortuitously suffers little loss of precision, while the odd points result from a better algorithm that suffers a significant loss of precision. This behaviour becomes more apparent with increasing λ. The round-off problem is exacerbated further when $g(y)$ is multiplied by y^λ in Eq.(10). The errors towards the origin are damped but, as y increases, the errors are amplified resulting in spurious values for I_6 and I_8.

3.1.2 Improved algorithm to compute $g(y) = \int_y^b \frac{P_{n_2,l_2}(x)P_{n_4,l_4}(x)}{x^{\lambda+1}}dx$

An alternative algorithm is to compute $g(y) = \int_y^b \frac{P_{n_2,l_2}(x)P_{n_4,l_4}(x)}{x^{\lambda+1}}dx$ directly, but to construct it in the direction of decreasing y, not only for efficiency, but more importantly because of the nature of the integrand shown in Fig. 5.

In this scheme,

$$g(y_k) = 0.0, \tag{17}$$

$$g(y_{k-1}) = \int_{y_{k-1}}^{y_k} \frac{P_{n_2,l_2}(x)P_{n_4,l_4}(x)}{x^{\lambda+1}}dx, \tag{18}$$

$$g(y_i) = g(y_{i+2}) + \int_{y_i}^{y_{i+2}} \frac{P_{n_2,l_2}(x)P_{n_4,l_4}(x)}{x^{\lambda+1}}dx, \tag{19}$$

with $y_k = b$ and y_1 set to a. This time the integral in Eq. (18) is approximated using a two-point Trapezoidal Rule and the integral in Eq. (19) is approxi-

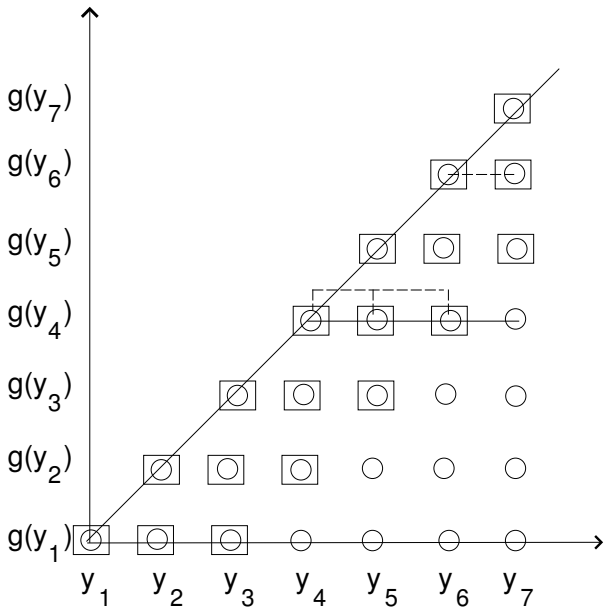

Fig. 6. $g(y_4) = g(y_6) + \int_{y_4}^{y_6}$. Here $g(y_4)$ is represented by the solid horizontal line through the circles, $g(y_6)$ is represented by the dashed line through the circles, and $\int_{y_4}^{y_6}$ is represented by the dashed line connecting the rectangles.

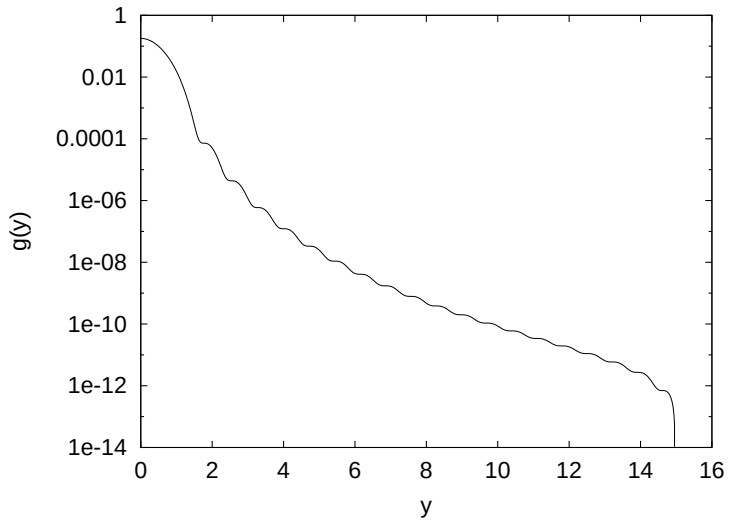

Fig. 7. The inner integral of $J_{2,8}$ using the improved algorithm: $g(y) = \int_y^{15.0} \frac{P_{n_2,l_2}(x)P_{n_4,l_4}(x)}{x^9} dx$, displayed on a log scale on the y axis.

mated using a three-point Simpson's Rule. The two-point rule is now applied to the smallest part of the integrand. This scheme is depicted in Fig. 6 for $g(y_4)$, with $k = 7$ for illustration.

A CADNA 'health check' was performed on the implementation of the new

algorithm. This time the values of the computed integral,

$$g(y) = \int_y^{15.0} \frac{P_{n_2,l_2}(x)P_{n_4,l_4}(x)}{x^9} dx, \tag{20}$$

display no erratic behaviour. This can be seen from Fig. 7. Furthermore, the fourth column of Table 3, confirms that the computed results suffer no loss of precision. Finally, comparison of the computed values of I_λ and I_λ improved, in Table 2, shows a significant and disconcerting difference for the two cases $\lambda = 6$ and $\lambda = 8$.

However, while the improved algorithm is computed to high precision how can we be confident that the results are accurate? We are still using an approximate method and CADNA is silent about the method's truncation error. In §4 we will use CADNA to minimize the global error and prove that these improved values of I_λ are actually accurate to 6 significant figures. But first we turn our attention to a CADNA 'health check' of the angular integrals.

3.2 CADNA analysis of the angular integrals

The two-electron angular integrals, $f_\lambda(l_1, l_2, l_3, l_4; L)$, are defined as follows,

$$\begin{aligned} f_\lambda(l_1, l_2, l_3, l_4; L) = (-1)^{l_1+l_3+L} & \frac{\sqrt{(2l_1+1)(2l_2+1)(2l_3+1)(2l_4+1)}}{2\lambda+1} \\ & \times W(l_1, l_2, l_3, l_4; L, \lambda) \\ & \times C(l_1, l_3, \lambda; 0, 0, 0) C(l_2, l_4, \lambda; 0, 0, 0), \end{aligned} \tag{21}$$

with

$$max(|l_1 - l_3|, |l_2 - l_4|) \leq \lambda \leq min(l_1 + l_3, l_2 + l_4). \tag{22}$$

The orbital angular momenta, l_1, l_2, l_3, l_4, are integer values that satisfy the following triangular relation with the system's total angular momentum L,

$$|l_1 - l_3| \leq L \leq l_1 + l_3, \tag{23}$$
$$|l_2 - l_4| \leq L \leq l_2 + l_4. \tag{24}$$

The $W(a, b, c, d; e, f)$ and $C(j_1, j_2, j_3; j_4, j_5, j_6)$ coefficients are the well known Racah and Clebsch-Gordan coefficients respectively. These coefficients are in essence sums of products of factorials and are defined as closed form expressions in Appendix A.

Table 4
A selection of computed values of $f_\lambda(11,11,11,11;4)$, $C(11,11,\lambda;0,0,0)$ and $W(11,11,11,11;4,\lambda)$.

λ	$f_\lambda(11,11,11,11;4)$	$C(11,11,\lambda;0,0,0)$	$W(11,11,11,11;4,\lambda)$
4	$0.50813444080822E-001$	$-0.236847553604108E+000$	$0.15410887950933E-001$
8	$-0.317795375791E-001$	$-0.2438287246232E+000$	$-0.17177940290068E-001$
12	$-0.71409106455E-002$	$-0.25662232592E+000$	$-0.5124471257194E-002$
16	$0.20770765E-001$	$-0.281612934E+000$	$0.1633825112085E-001$
22	$0.5036E-001$	$0.486E+000$	$0.18122135513439E-001$

Computation of $f_\lambda(l_1,l_2,l_3,l_4;L)$ is an example of a finite computation that, unlike an approximate method, is only affected by round-off. *In this case a CADNA 'health check' will reveal not only the precision, but more important, the accuracy of the computed result.*

The legacy routines used to compute the Clebsch-Gordan and Racah coefficients are `subroutine cg`, from `program STG2` [29], and `subroutine dracah`, from `program WEIGHTS` [34], respectively. As before, these routines were instrumented with CADNA and the output observed. No anomaly was found, again indicating that the stochastic values throughout the code are reliable and the precision of each has been correctly estimated.

In Table 4 we present some results for Case 3 as defined in [27]. This is a typical case that 2DRMP is expected to handle. This initial numerical health screening shows immediately a significant loss of precision and accuracy in $f_\lambda(11,11,11,11;4)$ as λ increases. Further investigation using CADNA reveals that the cause of the problem is the computation of the Clebsch-Gordan coefficients.

The closed expressions in Appendix A can be implemented in different ways, each resulting in different round-off behaviour. The legacy Clebsch-Gordan routine implements Eq. (27) directly, making no attempt to control the round-off error propagation associated with the multiplication and division of factorial values. This is perfectly satisfactory for small values of angular momentum, but, as can been seen from column three in Table 4, is wholly inadequate for larger values of angular momentum. This can be seen more starkly if we use `subroutine cg` to compute $C(10,30,40;2,2,4)$, an example taken from [39]. The result is a computational zero, @.0, which means that there is no significant digit.

The Racah legacy routine on the other hand does not implement Eq. (28) directly, rather it employs the approach discussed by Wills [39] which recasts Eq. (28) as Eq. (29). This approach removes factorials from the sum and those factorials outside the sum are manipulated as $\ln(n!)$ rather than $n!$. This approach is reliable for both large and small angular momenta. For example, using `subroutine dracah` to compute $W(35,35,40,40:26,45)$, an example

taken from [40], we obtain the following result that is accurate to 11 significant figures, $-0.17868488177E-002$.

4 Beyond the 'health check': dynamical determination of the optimal step size

The purpose of this section is to illustrate how CADNA can be used in a proactive way to establish benchmark results for the Slater integrals by dynamically determining the optimal step size to minimize the truncation error.

When an approximate numerical method is used, a *truncation error*, $e(h)_m$, which depends on the chosen step size h, is generated in addition to the *computing error*, $e(h)_c$, which is due to round-off error propagation. From the perspective of a computational scientist, the practical error of interest is the global error $e(h)_g$ which combines $e(h)_m$ and $e(h)_c$. As h increases $e(h)_m$ increases and $e(h)_c$ decreases, but when h decreases $e(h)_m$ decreases and $e(h)_c$ increases. These two sources of error counterbalance each other. Ideally we wish to minimize the global error which occurs when $e_c(h) \approx e_m(h)$. With normal floating-point arithmetic this is not possible as we have no estimate of $e_c(h)$: however, with CADNA it is possible. In effect, we generate a sequence of approximations, I_n, halving the step size each time. When the difference between two successive iterations is a computational zero no further meaningful computation can occur and we have achieved the optimal computed result for the method using the finite precision available. We establish this procedure on a solid mathematical foundation by recalling the following theorem that was proved in [35].

Theorem 1 Let $I(h)$ be an approximation of order p of an exact value I, *i.e.* $I(h) - I = Kh^p + \mathcal{O}(h^q)$ with $1 \leq p < q,\ K \in \mathrm{R}$.

If I_n is the approximation computed with the step $\frac{h_0}{2^n}$, then $C_{I_n,I_{n+1}} = C_{I_n,I} + \log_{10}\left(\frac{2^p}{2^p-1}\right) + \mathcal{O}\left(2^{n(p-q)}\right)$, where $C_{a,b}$ is the number of significant digits common to a and b.

For the Trapezoidal Rule and Simpson's Rule since $p = 2$ and $q = 4$ and $p = 4$ and $q = 6$, respectively [36], we obtain [37] for the Trapezoidal Rule,

$$C_{I_n,I_{n+1}} = C_{I_n,I} + \log_{10}\left(\frac{4}{3}\right) + \mathcal{O}\left(\frac{1}{4^n}\right), \tag{25}$$

and for Simpson's Rule,

Table 5
Benchmark computation of I_λ with $\lambda \in \{0, 2, 4, 6, 8\}$ in double precision for the case $a = 10^{-5}$, $b = 15.0$, $n_1 = 20$, $l_1 = 4$, $n_2 = 20$, $l_2 = 4$, $n_3 = 20$, $l_3 = 4$, $n_4 = 20$, $l_4 = 4$. While the results are displayed to 11 exact significant digits, by virtue of Theorem 1, the computed result is correct to 10 significant figures.

λ	I_λ
0	$0.12479372449E + 000$
2	$0.47155137140E - 001$
4	$0.28881377469E - 001$
6	$0.20934314687E - 001$
8	$0.16487550218E - 001$

$$C_{I_n,I_{n+1}} = C_{I_n,I} + \log_{10}\left(\frac{16}{15}\right) + \mathcal{O}\left(\frac{1}{4^n}\right). \tag{26}$$

This means that in a series of successive iterations of I_n, when the convergence zone is reached [8], the significant digits common to two successive approximations are also in common with the exact result I, up to one bit. This is because the term $\log_{10}\left(\frac{2^p}{2^p-1}\right)$ decreases as p increases and corresponds to one bit for the worst case $p = 1$. As we are using a mixture of the Trapezoidal Rule and Simpson's Rule we expect p to lie between 2 and 4.

In practical terms, if computations are performed in the convergence zone, until the difference between two successive approximations is a computational zero then the significant bits of the last approximation not affected by round-off errors are in common with the exact value of the integral, I, up to one bit.

Approximations $I_n(\lambda)$, for $\lambda \in \{0, 2, 4, 6, 8\}$, were computed in double precision for the case $a = 10^{-5}$, $b = 15.0$, $n_1 = 20$, $l_1 = 4$, $n_2 = 20$, $l_2 = 4$, $n_3 = 20$, $l_3 = 4$, $n_4 = 20$, $l_4 = 4$, until, for each λ, the difference $\mid I_n(\lambda) - I_{n+1}(\lambda) \mid$ had no exact significant digit. These values, which required 2^{17} integration steps, are presented in Table 5. While the CADNA function `str` prints 11 exact significant digits, by virtue of Theorem 1, only 10 are in common with the exact result $I(\lambda)$. Comparing Table 5 with Table 2 it is clear that the improved algorithm using 1025 integration points produces results accurate to six significant figures. This also confirms the catastrophic error in the original algorithm for I_6 and I_8.

[8] *i.e.* $\mathcal{O}\left(2^{n(p-q)}\right)$ becomes negligible.

5 Conclusions

In this paper we have focused on the danger of round-off error propagation in scientific codes, particularly in codes that employ legacy routines. We have advocated and demonstrated the use of the CADNA library as a numerical 'health check' screening tool to help detect and control these errors. We now evaluate CADNA again the numerical screening tool criteria identified in the introduction.

- **Report gradual and catastrophic loss of precision?** We have demonstrated that CADNA's `cestac` and `str` functions and its concept of a computational zero easily enables the detection and reporting of both gradual and catastrophic loss of precision. We have observed that round-off error can seriously diminish the precision of the results of a computation and that the degree of damage cannot always be determined by repeating the computation with a longer word length. However, by using the CADNA library we can determine for each arithmetic and logical operation whether a numerical anomaly has occurred and if not CADNA will provide definitively the number of significant figures in the result that are not affected by round-off. This is arguably CADNA's greatest strength, rendering it a very effective numerical screening tool for scientific programs.
- **Report the accuracy of intermediate and final results?** This criterion presents CADNA with some difficulty. The problem is that approximate numerical methods are subject to both round-off error and a truncation error due to the computational method. We have illustrated a poor method that is well computed and gives rise to an inaccurate result with a large number of exact significant digits. Likewise, we have illustrated a better method that is poorly computed and gives rise to a result that is more accurate with very few significant digits. This emphasizes the important distinction between accuracy and precision and shows that the CADNA library is silent about the accuracy of approximate computational methods. Most scientific codes will be constructed from a mix of finite, iterative and approximate methods. The complex interaction and interference amongst their results means that CADNA cannot automatically indicate the accuracy of intermediate and final results. However, as we will see below CADNA can still be an effective and useful tool when used proactively in a careful and controlled manner.
- **Be of acceptable efficiency?** It has been shown empirically [12] that the use of stochastic triples is sufficient to provide estimations of round-off errors that are perfectly realistic and not pessimistic. When used in 'health check' mode the increase in execution time by a factor of at least three is acceptable. However, it is unlikely to be acceptable in production runs, except perhaps in safety-critical application areas, such as transportation and nuclear-power generation, where performance may be outweighed by safety.

- **Be non invasive to the source code?** The use of overloaded operators and the replacement of intrinsic types with derived stochastic types makes the CADNA library straightforward to implement. However, in a large code even this is tedious and error prone. Problems can arise when third party software or libraries are used. For example, if LAPACK routines are used these must also be instrumented with CADNA. Nevertheless, the benefits gained significantly outweigh the inconvenience of instrumenting a source code with CADNA.

In addition to using CADNA as a numerical 'health check' screening tool we have also explored the use of CADNA in proactive way. Some further comments are pertinent here.

In normal floating-point arithmetic iterative methods are typically terminated when the difference between two successive iterations is less than or equal to ϵ where ϵ is some arbitrary positive value. This criterion is unsatisfactory. First, if ϵ is too large the sequence is terminated prematurely before the optimal solution is reached. Second, if ϵ is too small useless iterations will be performed which cannot, because of round-off error propagation, improve the accuracy of the result. Furthermore, when the termination criterion is reached no information is available on the precision of the computed result. We have shown that by using the CADNA library the iterative process can be dynamically controlled and terminated at the optimal point. This is when, in the convergence zone, the difference between two successive iterations is a computational zero. No further meaningful computation can occur and we have achieved the optimal computed result for the method using the floating-point number representation available. In addition, we know the exact number of significant digits in the computed result.

For certain iterative problems, use of the CADNA library enables definitive benchmark computations to be performed. We have shown that when Theorem 1 is applicable then, when the optimal termination point is reached, we get the added benefit of knowing not only the number of exact significant digits, but that these digits are identical with those in the exact result. A significant benefit is that this represents a benchmark result against which the computed results of other, perhaps, more complicated methods can be directly compared. An example is the development of ad hoc extended frequency dependent quadrature rules recently designed to improve the accuracy and significantly reduce the computation time of the Slater integrals [27]. This method's computed results agree with the benchmark CADNA computations and can therefore be concluded to be accurate to 10 significant figures, with no formal analysis of the method being required.

Finally, we should note that CADNA is not a numerical panacea. First, CADNA tells us nothing about the accuracy of the underlying computational

method. Second, when used in 'health check' mode it suffers from the plight of all testing methods: it only gives us information about the tests carried out and tells us nothing definitive about the range of values not tested. However, having been used successfully in industry and research centres for many years we believe it to be a reliable and useful tool for helping to validate the results of numerical software.

Acknowledgments

NSS is indebted to Laboratoire d'Informatique de Paris 6, l'Université Pierre et Marie Curie, Paris for a Visiting Professorship during which much of this work was performed. The authors are grateful to Maurice Clint and Charlotte Froese Fischer for a critical reading of the manuscript and for constructive comments.

References

[1] K. V. Roberts, *The publication of scientific Fortran programs*, Comput. Phys. Commun. **1** (1969) 1.

[2] L. Fox , *How to get meaningless answers in scientific computation (and what to do about it)*, IMA Bulletin **7** (1971) 296.

[3] Bo Einarsson(Ed), *Accuracy and Reliability in Scientific Computing*, (SIAM Philadelphia) (2005).

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

[5] J. S. Ely, *Prospects for Using Variable Precision Interval Software in C++ for Solving Some Contemporary Scientific Problems*, PhD thesis, The Ohio State University (1990).

[6] J. H. Wilkinson, *Error analysis revisited*, IMA Bulletin **22** (1986) 192.

[7] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, (SIAM Philadelphia) (1996).

[8] S. Linnainmaa, *Error linearization as an effective tool for estimation of rounding errors in floating-point computations*, BIT **15** (1983) 165.

[9] U. W. Kulisch, *Advanced Arithmetic for the Digital Computer*, (Springer-Verlag, Wien) (2002).

[10] D. S. Parker, B. Price and P. Eggert, *How to gamble with floating point and win*, Comput. Sci. Engrg. July-August (2000) 59.

[11] F. Chaitin-Chatelin and V. Fraysse, *Lectures on Finite Precision Computations*, (SIAM Philadelphia) (1996).

[12] J. Vignes, *A Stochastic arithmetic for reliable scientific computation*, Math. Comput. Simulation **35** (1993) 233.

[13] CADNA website - http://www.lip6.fr/cadna

[14] J. Vignes, *Error analysis in computing*, in: *International Federation for Information Processing Congress*, Stockholm, August (1974) 610.

[15] J. Vignes, *New methods for evaluating the validity of the results of mathematical computations*, Math. Comput. Simulation **20** (1978) 227.

[16] J. Vignes, *Zéro mathématique et zéro informatique*, in: *La Vie des Sciences*, Comptes Rendus de l'Académie de Sciences **4** (1987) 1.

[17] J.-M. Chesneaux, *Stochastic arithmetic properties*, Computational and Applied Mathematics, I Algorithms and theory, C. Brezinski (editor), North Holland (1992) 81.

[18] J.-M. Chesneaux, *The equality relations in scientific computing*, Numerical Algorithms **7** (1994) 129.

[19] IEEE 1987, *IEEE Standard 754-1895 for binary Floating-point Arithmetic*, IEEE, 1985, Reprinted in SIGPLAN **22** (2) 9.

[20] J. Vignes, *Discrete stochastic arithmetic for validating results of numerical software*, Numer. Algorithms **37** (2004) 377.

[21] J.W. Heggarty, P. B. Burke, M. P. Scott and N. S. Scott, *Computational Aspects of the Two-dimensional Propagation of R-matrices on MPPs*, Comput. Phys. Commun. **114** (1998) 195.

[22] J. W. Heggarty, *Parallel R-matrix Computation*, Ph.D. Thesis, Queen's University Belfast (1999).

[23] T. Stitt, *Propagation of R-matrices on a 2D plane*, Ph.D. Thesis, Queen's University Belfast (2005).

[24] Timothy Stitt , N. Stan Scott, M. Penny Scott and Phil G. Burke, *2-D R-matrix propagation: a large scale electron scattering simulation dominated by the multiplication of dynamically changing matrices*, Lecture Notes in Computer Science, J. M. L. M. Palma et al eds, **2565** (2003) 354.

[25] A. Carson, T. H. Harmer, N. S. Scott, V. Faro-Mazo, M. P. Scott and P. G. Burke, *2-D R-matrix Propogation: the Calculation of Atomic Collision Data using Grid Resources*, Lecture Notes in Computer Science, M. Daydé et al eds, **3402** (2005) 233.

[26] Virginia Faro-Mazo, P. Preston, T. Harmer and N.S. Scott, *VisRes-G: A Grid Computational Steering and Visualisation Tool for R-matrix Computations*, UK e-Science All Hands Meeting, (2004) 1007
(http://www.allhands.org.uk/2004/proceedings/papers/134.pdf).

[27] L. Gr. Ixaru, N. S. Scott and M. P. Scott, *Fast computation of the Slater integrals*, SIAM J. Sci. Comput. **28** (2006) 1252.

[28] K.A. Berrington, P.G. Burke, J.J. Chang, A.T. Chivers, W.D. Robb and K.T. Taylor, *A general program to calculate atomic continuum processes using the R-matrix method*, Comput. Phys. Commun. **8** (1974) 149.

[29] K.A. Berrington, P.G. Burke, M. Le Dourneuf, W.D. Robb, K.T. Taylor and V.K. Lan, *A new version of the general program to calculate atomic continuum processes using the r-matrix method*, Comput. Phys. Commun. **14** (1978) 367.

[30] N.S. Scott and K.T. Taylor, *A general program to calculate atomic continuum processes incorporating model potentials and the Breit-Pauli Hamiltonian within the R-matrix method*, Comput. Phys. Commun. **25** (1982) 347.

[31] P.G. Burke, V.M. Burke and N.S. Scott, *A new no-exchange R-matrix program*, Comput. Phys. Commun. **69** (1992) 76.

[32] K.A. Berrington, W.B. Eissner, P.H. Norrington, *RMATRX1: Belfast atomic R-matrix codes*, Comput. Phys. Commun. **92** (1995) 290.

[33] O. Zatsarinny, *BSR: B-spline atomic R-matrix codes*, Comput. Phys. Commun. **174** (2006) 273.

[34] N. S. Scott and A. Hibbert, *A more efficient version of the WEIGHTS and NJSYM packages*, Comput. Phys. Commun. **25** (1982) 347.

[35] F. Jézéquel, *A dynamical strategy for approximation methods*, C. R. Acad. Sci. Paris - Mécanique **334** (2006) 362.

[36] J. Stoer and R. Bulirsch, *Introduction to numerical analysis*, (Springer) (2002).

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

[38] M. E. Rose, *Elementary Theory of Angular Momentum*, (Wiley, New York) (1957).

[39] J. G. Wills, *On the evaluation of angular momentum coupling coefficients*, Comput. Phys. Commun. **2** (1971) 381.

[40] K. Srinivasa Rao and K. Venkatesh, *New Fortran programs for angular momentum coefficients*, Comput. Phys. Commun. **15** (1978) 227.

Appendix A

Wigner's closed expression for the Clebsch-Gordan coefficient (see Eq. 3.18 in [38]),

$$C\left(j_1,j_2,j_3;m_1,m_2,m_3\right)=\delta_{m_3,m_1+m_2}$$
$$\times\left[(2j_3+1)\frac{(j_3+j_1-j_2)!(j_3-j_1+j_2)!(j_1+j_2-j_3)!(j_3+m_3)!(j_3-m_3)!}{(j_1+j_2+j_3+1)!(j_1-m_1)!(j_1+m_1)!(j_2-m_2)!(j_2+m_2)!}\right]^{\frac{1}{2}}$$
$$\times\sum_{\nu}\frac{(-)^{\nu+j_2+m_2}(j_2+j_3+m_1-\nu)!(j_1-m_1+\nu)!}{\nu!(j_3-j_1+j_2-\nu)!(j_3+m_3-\nu)!(\nu+j_1-j_2-m_3)!} \tag{27}$$

Racah's closed expression for the W-coefficient (see Eq. 6.7 in [38]),

$$W\left(a,b,c,d;ef\right)=\Delta_R(abe)\Delta_R(cde)\Delta_R(acf)\Delta_R(bdf)$$
$$\times\sum_{\chi}\frac{(-)^{\chi+a+b+c+d}(\chi+1)!}{(\chi-a-b-e)!(\chi-c-d-e)!(\chi-a-c-f)!(\chi-b-d-f)!}$$
$$\times\frac{1}{(a+b+c+d-\chi)!(a+d+e+f-\chi)!(b+c+e+f-\chi)!}, \tag{28}$$

with

$$\Delta_R(abc)=\left[\frac{(a+b-c)!(a-b+c)!(-a+b+c)!}{(a+b+c+1)!}\right]^{\frac{1}{2}}.$$

Scott and Hibbert formulation of the W-coefficient (see Eqs. (4-7) in [34]) using approach of Wills [39],

$$W\left(a,b,c,d;ef\right)=(-)^{q_{min}+a+b+c+d}\frac{a_0!}{b_0!}$$
$$\times\,\Delta_R(abc)\Delta_R(cde)\Delta_R(acf)\Delta_R(bdf)F(abcdef), \tag{29}$$

with

$$a_{\ i}=(q_{min}+1+i)\Big[(q_{min}+i-a-b-e)(q_{min}+i-c-d-e)$$
$$\times(q_{min}+i-b-d-f)(q_{min}+i-a-c-f)\Big]^{-1}, \tag{30}$$

$$b_{\ i}=(a+b+c+d-q_{min}-i)(a+e+d+f-q_{min}-i)$$
$$\times(b+c+e+f-q_{min}-i), \tag{31}$$

$$F\left(abcdef\right)=[1-a_1b_0(1-a_2b_1(1-a_3b_2(1\ldots]. \tag{32}$$