Determining the Numerical Stability of Quantum Chemistry Algorithms

Gerald Knizia,*[†] Wenbin Li,[‡] Sven Simon,[‡] and Hans-Joachim Werner[†]

[†]Institut für Theoretische Chemie, Universität Stuttgart, Pfaffenwaldring 55, D-70569 Stuttgart, Germany
[‡]Institut für Parallele und Verteilte Systeme, Universität Stuttgart, Universitätsstraβe 38, D-70569 Stuttgart, Germany

ABSTRACT: We present a simple, broadly applicable method for determining the numerical properties of quantum chemistry algorithms. The method deliberately introduces random numerical noise into computations, which is of the same order of magnitude as the floating point precision. Accordingly, repeated runs of an algorithm give slightly different results, which can be analyzed statistically to obtain precise estimates of its numerical stability. This noise is produced by automatic code injection into regular compiler output, so that no substantial programming effort is required, only a recompilation of the affected program sections. The method is applied to investigate: (i) the numerical stability of the three-center Obara–Saika integral evaluation scheme for high angular momenta, (ii) if coupled cluster perturbative triples can be evaluated with single precision arithmetic, (iii) how to implement the density fitting approximation in Møller-Plesset perturbation theory (MP2) most accurately, and (iv) which parts of density fitted MP2 can be safely evaluated with single precision arithmetic. In the integral case, we find a numerical instability in an equation that is used in almost all integral programs. Due to the results of (ii) and (iv), we conjecture that single precision arithmetic can be applied whenever a calculation is done in an orthogonal basis set and excessively long linear sums are avoided.

1. INTRODUCTION

Numerical computations are done with number representations of limited accuracy. In special circumstances, an unfortunate amplification of rounding errors can occur, such that the final calculation result (for example, a total molecular energy) is much less accurate than expected or desired. Algorithms with this behavior are called numerically unstable. Unfortunately, in general it is very hard to determine if a concrete implementation of a quantum chemistry (QC) method suffers from numerical instabilities, and if it does, where the sources lie. In this article we present a numerical analysis method based on random rounding. This method is easy to apply and can answer both questions with minimal programming effort.

Various accuracy pitfalls arise from the properties of the ubiquitous floating point (FP) arithmetic, which violates common mathematical identities.[1] For example, the associativity law of addition does not hold for FP numbers a, b, c: In general, $(a + b) + c \neq a + (b + c)$, and which summation order is taken can have stark impact on the calculation result. For example, if we consider single precision FP numbers with about seven decimal digits of precision ($\varepsilon \approx 6 \times 10^{-8}$) and set

$$a = 1.4142136 \quad b = -1.4142136$$
$$c = 1.2345678 \times 10^{-5}$$

then we get for $a + b + c$, depending on the summation order:

$$(a + b) + c = 1.2345678 \times 10^{-5} \quad \text{or}$$
$$a + (b + c) = 1.2397766 \times 10^{-5} \qquad (1)$$

In the second variant the last five digits are completely bogus; the result's relative accuracy is only 5×10^{-4}, despite the fact that a single precision variable could store results accurate to 6×10^{-8}. Under unfortunate conditions, such rounding errors can accumulate and amplify in nontransparent ways and induce an inaccurate final calculation result.

QC programs usually do not state how many digits of a calculation result can be trusted—because this information is not readily available and obtaining it used to require lots of work. For the same reason a detailed numerical analysis of a program is usually not done unless obvious numerical problems show up, and even then typically the problems and/or solutions are only published as side remarks, if at all (e.g., refs 2–9). While one can expect that time-proven programs are "accurate enough" in practice, this does not necessarily hold for newly developed programs. In particular, algorithms which rely on deep recurrence relations are prone to numerical problems (e.g., molecular integral evaluation for Gaussian basis sets),[10] as are molecular dynamics algorithms which depend on time-reversibility (e.g., transition path sampling),[11] and algorithms employing any kind of large, semi-redundant basis set (e.g., density fitting basis sets in Hartree–Fock[12,13] and MP2 theories,[14,15] complementary auxiliary basis sets in F12 theory[16–18] or nonredundant orbital rotations,[19,20] or configuration spaces[20,21] in multireference theories).

As another aspect, new computational hardware systems, like general purpose graphics processing units (GPUs), field-programmable gate arrays (FPGAs), or Cell processors, are seeing a rapid increase in popularity. For example, several groups have implemented parts of QC programs, like molecular integrals,[22–25] Hartree–Fock or density functional theory,[23,24,26–29] or density-fitted MP2,[30–32] on GPUs or similar platforms.[33,34] Unlike on central processing units (CPUs), where until very recently single precision arithmetic offered only minor processing speed benefits, arithmetic in single precision is much faster than in double precision on such novel hardware. This is the case because in this context double precision support is typically reduced to a (economical) minimum, because low-precision arithmetic can lead to distinct advantages in the hardware costs associated with

Received: April 7, 2011
Published: July 14, 2011

chip area, clock period, latency, and power consumption. For example, the hardware resources required for an addition circuit scale linearly with word length, while for a multiplication circuit they even scale quadratically.[35] In light of these hardware developments, it is worthwhile to investigate whether (and how) expensive parts of calculations, for example, the perturbative (T) triples correction[36] in coupled cluster theory, can be evaluated with sufficient accuracy also with single precision arithmetic. Some of the aforementioned articles[24,26,32] also dealt with these issues; additionally ref 37 explicitly investigated whether parts of Cholesky-decomposition MP2 (CD-MP2) can be done in single precision arithmetic.

For these reasons we believe that better tools for numerical error analysis are required. In this article we describe one novel, easily applied approach, which can be executed without extensive source code modification.

So let us assume that we compute a FP number E_{dbl} with some program, and we want to find out how many digits of E_{dbl} can be trusted. Then traditionally one of the following techniques is used:

Comparison with a High Precision Reference. If the program can be changed to employ high-precision or multiprecision FP arithmetic instead of the standard double precision arithmetic, then we can in this way calculate a more accurate $E = E_{\mathrm{hp}}$ in high precision again and see if it agrees with the double precision result E_{dbl}.

The main problem here is that the source code of the calculation program usually has to be modified, and this can imply a huge workload. (It should be noted that some recent compilers (e.g., Intel Fortran) can redefine intrinsic floating point data types to quadruple precision, thus potentially greatly simplifying the calculation of E_{hp} reference values. However, since this implies a change of the memory layout of the program, addressing, I/O operations, data compression, interfaces within the program to to external programs and libraries, etc. may still need to be adjusted in nonstraightforward ways.) Additionally, high-precision arithmetic must be emulated and can become very slow. Furthermore, since only one E_{dbl} is available, and this result might by chance be very bad or very good, the precise size of the expected rounding errors is hard to estimate. Also, in some cases the convergence of the result with increasing precision can be misleading; an extreme example was constructed by Rump:[38,39]

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

where $a = 77617$ and $b = 33096$. When the computations are performed in single, double, and quadruple precision, the following results are obtained

$$\text{Single precision}: f_s = 1.172604$$

$$\text{Double precision}: f_d = 1.17260394005318$$

$$\text{Quad precision}: f_q = 1.17260394005317863185\cdots$$

$$\text{Exact result}: f = -0.827396059946821368141\cdots$$

By looking at the first three results, the single precision result seems to be accurate, yet the correct answer is not even found with quadruple precision (it requires 122 mantissa bits to get the leading term correct). Thus, in pathological cases the comparison with a high-precision reference can lead to erratic results, unless one can guarantee that the high-precision reference is really precise enough. The method we will present does not have this defect.

Interval Arithmetic (IA). IA is a well-known method of numerical stability analysis.[40] In IA, a single standard FP number is replaced by a pair of FP numbers representing its strict upper and lower bound; the basic arithmetic operations are defined on those intervals and incorporate rounding errors into the bounds. IA generates strict bounds on rounding errors, but it is less useful to obtain information on the practical numerical stability of algorithms. The strict bounds tend to exaggerate the actual accuracy loss due to rounding by orders of magnitude, since error cancellation cannot possibly occur.[41] Additionally, IA is difficult to implement into an existing program, requiring large-scale changes to the entire source code, and IA is much slower than straight FP arithmetic.

Since both high-precision reference and IA computations require large modifications of the affected programs, they are not commonly applied in QC. The only actually applied "stability analysis" method we are aware of is to try if the same calculation gives different results if other compilers or other numerical libraries (BLAS etc.) are used. Obviously, the reliability of this approach is limited.

Instead of the sketched methods, we propose to use a modification of a little-known probabilistic error analysis technique by Vignes.[42,43] The original method has been termed controle et estimation stochastique des arrondis de calculs (CESTAC) or in a slight modification discrete stochastic arithmetic (DSA).[44] These methods work by estimating rounding errors during the run time of a program in a statistical manner and thus attempt to model correct rounding error propagation. They have been used successfully for numerical analysis in a few previous cases.[45−47] The original implementations of the DSA method are based on operator overloading, and thus are restricted to programming languages supporting this feature (e.g., not Fortran 77 or C, two popular languages for numerical software), and furthermore, they incur the same (or even more) program adjustment overhead as IA or high-precision reference calculations.

In contrast to CESTAC/DSA, our method is noninvasive (does not require source-code modification) and focuses on obtaining rounding error estimates of final calculation results, not of every single intermediate number. With our approach, accurate numerical stability estimates can be obtained quickly and with little programming effort. Additionally, our method works for most popular programming languages, as long as they are compiled.

2. METHOD

The goal of a numerical stability analysis is: (i) to assess how the tiny intermediate rounding errors of FP arithmetic combine into the uncertainty of the final calculation result and (ii) to quantify how uncertainties are introduced or amplified by the various subtasks of a calculation. Both questions can be answered *experimentally* by introducing random numerical noise into the performed FP arithmetic. Concretely, if we want to judge the numerical impact of a subtask S (say, the calculation of molecular integrals) on the final result E (say, a total molecular energy), we proceed as follows:

- For each FP operation in S, we add numerical noise on the same order of magnitude as the precision of the individual

FP numbers occurring. For a double precision number $f * 2^m$, where f is the mantissa and m is the exponent, this magnitude is $\pm 2^{-53}$ ($\approx 10^{-15.95}$, in the 16th decimal digit).

- Because the noise is so small, under optimal conditions it does not affect the final calculation E noticeably. If, however, some amplification of the rounding errors takes place, then our introduced numerical noise is amplified in exactly the same way as the inherent finite precision rounding error (whose impact is what we actually want to quantify).
- If we now run the entire calculation of E multiple times, we will observe some numerical scattering in its result, which is caused by the noise introduced by S. By statistically analyzing the individual results E_i ($i = 1, ..., n$) obtained in n runs of the program, we can estimate how many digits of accuracy in E are lost due to S. A first indication is given by the order of magnitude of the root-mean-square deviation (rmsd):

$$\mathrm{rmsd}(E) = \sqrt{\frac{1}{n}\sum_i (E_i - \langle E_i \rangle)^2} \qquad (2)$$

which approximates the standard deviation of the scattering of the calculation results. Due to the central limit theorem,[49] in most cases one can expect the scattering in E to follow a normal distribution. In this case a statistically sound reliability analysis can be obtained even for small n (say, 3, ..., 5) by weighting with Student's t-distribution.[49] If the scattering exactly follows a normal distribution around the exact result, then $2 \cdot \mathrm{rmsd}(E)$ is a good estimate of the 95% confidence interval of the numerical error. But since the actual distributions often have wider tails, errors of $3 \cdot \mathrm{rmsd}(E)$ or even $4 \cdot \mathrm{rmsd}(E)$ can occasionally be seen.
- The original methods[42−44,50] now go at great length to estimate the number of trustworthy digits from this deviation. We omit this part because: (i) it provides no additional information to the rmsd and (ii) there are cases when the obtained results are wrong (see below).

Until here we closely followed the CESTAC/DSA prescriptions in refs 42−44. But the indiscriminate nature of the method—every FP operation acquires noise—makes it possible to implement it not by modifying the source code of a program but rather the compilation process transcribing it. This is very important when the program in question consists of several million lines of code, like Molpro,[51] which we will use as our test subject. As most compilers can produce an intermediate representation of a compiled program in terms of assembler code, and as the logical context of the FP operations is not required, this modification can even be accomplished without changing the compiler program itself. Rather than that, the intermediate assembly representation of the program being compiled is altered to include the noise production. (The assembly representation of a program contains a list of mnemonics of machine instructions with operands. It corresponds closely to the final binary program code. All instructions and accesses to registers and memory are explicit there.)

We implemented the concept for the Fortran 95/x86−64 and C++/x86−64 combination of platforms. Concretely, we developed a Python script which pretends to be either the g95 Fortran compiler or the GNU C/C++ compiler, respectively. The script accepts the same command line arguments as the compiler and can thus be seamlessly substituted for it when compiling the program to be analyzed. The script first calls the real g95 or g++ compiler in order to compile input source code to assembler files. It then scans generated assembly code, and whenever it encounters a FP instruction from a predefined list, it injects code for changing the x87 rounding mode in a pseudorandom manner (see below). It then runs the real compiler again to assemble and/or link the resulting modified assembly files (see Figure 1). The script and support files will be made freely available on the Web at http://www.theochem.uni-stuttgart.de/random_rounding/.

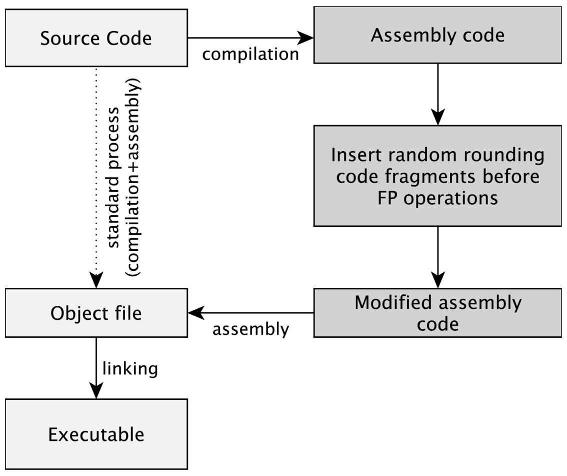

Source Code
compilation
Assembly code
standard process (compilation+assembly)
Insert random rounding code fragments before FP operations
Object file
assembly
Modified assembly code
linking
Executable

Figure 1. Modified compilation process for random rounding analysis. The script causes the compiler to generate assembly code, modifies the assembly to include the random rounding code,and then again uses the real compiler for assembly and linking.

The rounding mode of the FPU determines if intermediate FP computations round nonprecise digits toward the next representable number ("nearest"), toward $+\infty$ ("up"), or toward $-\infty$ ("down"), with "nearest" being the default choice for obvious reasons. But by changing this rounding mode randomly to up/down before the operations, computations are influenced only on the order of magnitude of the machine epsilon, which is exactly what we want for introducing noise. (On hardware platforms not supporting x87-style rounding modes, this behavior can be emulated by multiplying FP operands with $1 \pm \varepsilon$ where $\varepsilon \approx 2^{-53}$ before executing the original FP operations, as Vignes originally suggested.[43])

The technical details of how the random rounding mode is implemented are described in the Appendix. Here we just note that the implementation allows for switching between the rounding modes "nearest, 64bit", "random up/down, 64bit", "nearest, 32bit", and "random up/down, 32bit" during the runtime of the program by simple function calls. To apply the propsed analysis method it is thus sufficient to recompile the parts of the program which are to be analyzed (or the whole program, if this is desired) with the noise-injecting meta-compiler and to call the functions for switching between the rounding modes at the appropriate places. Then the program is executed multiple times, and the random scattering of the numerical results is used to assess its numerical stability. The modified program is significantly slower than the original program, typically by a factor of 5−50. But as we will demonstrate later on, meaningful calculations can still be performed easily, since the bad scaling of QC algorithms allows one to offset this cost by choosing a somewhat smaller molecule for numerical testing. Note also that the slowdown is specific to the x87 pipeline hardware architecture and can be greatly reduced on other architectures.[52]

2.1. Pitfalls of Random Rounding Analysis. In calculations with FP numbers, by far the most important and most intransparent source of numerical problems is the elimination of significant digits due to subtraction, because this process leads to an amplification of relative accuracy loss (sketched around eq 1. In general we always eliminate relative accuracy when subtracting FP numbers of similar magnitude: If $a \approx -b$, then $a + b$ is inaccurate unless both a and b are exact.). This is called amplification error in this section. This process is very accurately modeled by the random rounding approach described above and is hard to describe in other ways. However, random rounding may lead to problems in two situations sketched in this section.

2.1.1. Iterative Processes. Although we have not personally seen this, it is possible that the analysis of iterative processes is jeopardized by their self-adjusting nature. However, provided that the same sequence of floating point operations is executed in each iteration, an numerical analysis can still be carried out easily. For this it is sufficient to reseed the pseudorandom number generator used for the rounding mode generation to the same seed value at the beginning of each iteration. In this way the same rounding sequences are used in each iteration during a single run of the program, and the results obtained in one run of the program are equivalent to a sequence one could get if the entire program was run on another machine (which would lead to different rounding errors, but in always the same way in each iteration). Additionally it might be advantageous to fix the number of iterations to some specified value because, otherwise, the number of iterations could differ between different runs in a sample if convergence thresholds are only closely met or missed.

2.1.2. Summing Up Large Amounts of Very Small Numbers. Apart from the elimination of significant digits due to subtraction, there is also another, qualitatively different kind numerical accuracy loss: the simple, nonamplified accumulation of less-than-machine precision (ε) sized individual errors which just happen to be done often enough to still cause significant deviations in final results. This case can occur in some quantum chemistry algorithms due to the immense amount of data handled, and it requires special consideration (think of the assembly of correlation energies from integrals and wave function amplitudes via four- or six-dimensional sums).

For an extreme example, let us assume that we work with single precision FP numbers (which have approximately 7.2 decimal digits of precision) and that we want to calculate the sum:

$$s = (\ldots(((1.0 + \underbrace{1e\text{–}8) + 1e\text{–}8) + \ldots) + 1e\text{–}8}_{10^8 \text{ times}}. \qquad (3)$$

The exact value of the sum is $s_{\text{exact}} = 2.0$. But with default single precision FP arithmetic (i.e., using round to nearest) we would get $s_{\text{FP}} = 1.0$, because in each single operation $s := s + 1e{-}8$ the change in s is smaller than 0.5 units in the last place (ulp), and it remains at $s = 1.0$. Note that there is no amplification of rounding errors at all—every single operation has an accurate result. And yet the entire process combined results in a relative error of 100%. If we would execute this operation with random rounding instead of standard FP arithmetic, then we would get a different result. For each operation $s := s + a$, the random rounding process introduces a numerical scattering on the order of ± 0.5 ulp on s, the direction of which depends only on the sign of $(s + a) - s$ (here: always positive, since $a = 1e{-}8$). As described, typically this is exactly what one wants to have, but in the current case, this does not lead to a noticeable *scattering* in the sum s but rather to a systematic *bias* in its probabilistic mean value. The random rounding result obtained is $s \approx 62.736$ with standard deviation $\delta s \approx 0.016$, instead of $s = 2.0$ with $\delta s \approx 1$ as one might expect. And in fact, one would get the same result of $s \approx 62.7$ when 10^8 times summing up $1e{-}9$, $1e{-}15$, or $1e{-}30$ to 1.0. In the latter cases the default round-to-nearest results of $s_{\text{FP}} = 1.0$ would be almost exact by accident.

We have seen that the mass truncation error is not properly reflected in the scattering produced by random rounding. However, the method is still useful for this case because a systematic bias is easily measured (by comparing the random-rounding mean value to the default round-to-nearest result), and its presence indicates that the investigated algorithm may suffer from mass truncation errors. While the possible truncation error can be greatly exaggerated by the difference between mean and round-to-nearest results (unlike the amplification error, the straight truncation error is reflected more in a worst-case than average way), the absence of such a bias is a very strong indication of the absence of truncation errors. Thus, if an algorithm shows neither sizable bias in the mean values nor sizable scattering, it is most likely numerically stable.

Note that a mass truncation error of this kind is: (i) rare and easily identified in the source code, due to the immense amount of data required to produce nonentirely negligible effects and (ii) easily fixed by employing a compensated summation algorithm[53,54] (five lines of code!) or by performing the sum in higher precision.

Note also that an artificial version of the bias problem will arise when treating molecules with higher point group symmetry but using only the $D2h$ subgroups of this symmetry explicitly. In this case there are many almost zero values in the calculation (which would be exactly zero if wave functions were exactly converged). These values have no influence on default round-to-nearest results but may create systematic shifts in random rounding results. We recommend to avoid this problem by choosing suitable test molecules for analyzing algorithms.

3. APPLICATIONS

In order to demonstrate the practical feasibility of the method, we apply it to some numerical questions which arose during our own development activities on the Molpro program package.[51] Concretely, these are: (i) Which parts of correlation calculations can safely be done in single precision floating point arithmetic, now that hardware is gaining momentum which actually profits from that?, (ii) How numerically accurate are molecular integrals for high angular momenta?, and (iii) How do various ways of implementing the density fitting equations perform in relation to one another in terms of accuracy? These questions are answered in the following sections.

3.1. Perturbative (T) Correction of CCSD(T). We investigated whether the perturbative (T) correction[36] of the coupled cluster singles and doubles (CCSD) method can be evaluated accurately with single precision FP arithmetic. This correction is a prime candidate for acceleration on new high-performance platforms, because it is very expensive and its working equations are simple and well-conditioned if evaluated in an orthogonal basis. Concretely, for a converged Hartree–Fock reference function the closed-shell working equations are[55,56]

$$E_{(T)} = \frac{1}{3}\sum_{ijk}^{\text{occ}}\sum_{abc}^{\text{vir}}(4W_{abc}^{ijk} + W_{bca}^{ijk} + W_{cab}^{ijk}) \times (V_{abc}^{ijk} - V_{cba}^{ijk})/D_{abc}^{ijk} \qquad (4)$$

Table 1. Numerical Stability of the Single Precision (T) Triples Correction[a]

	SO_2/AV5Z	naphthalene/VDZ
	Double Precision, Round to Nearest	
RHF	−547.319 262 404 907	−383.382 869 789 653
CCSD	−0.720 154 348 436	−1.347 263 987 000
(T)	−0.040 519 128 711	−0.063 745 284 695
	Single Precision, Random Rounding	
(T), #1	−0.040 519 128 614	−0.063 745 284 643
(T), #2	−0.040 519 128 781	−0.063 745 284 703
(T), #3	−0.040 519 128 515	−0.063 745 284 682
(T), #4	−0.040 519 128 717	−0.063 745 284 712
(T), #5	−0.040 519 128 709	−0.063 745 284 708
(T), #6	−0.040 519 128 883	−0.063 745 284 648
(T), #7	−0.040 519 128 690	−0.063 745 284 757
(T), #8	−0.040 519 128 673	−0.063 745 284 659
mean	−0.040 519 128 698	−0.063 745 284 689
rmsd	0.000 000 000 102	0.000 000 000 036

[a] The first three lines give energies (in au) of standard double precision runs, and the next eight lines give individual values obtained with single precision and random rounding.

$$W^{ijk}_{abc} = P^{ijk}_{abc}[(bd|ai)T^{kj}_{cd} - (ck|jl)T^{il}_{ab}] \quad (5)$$

$$V^{ijk}_{abc} = W^{ijk}_{abc} + (bj|ck)t^i_a + (ai|ck)t^j_b + (ai|bj)t^k_c \quad (6)$$

$$D^{ijk}_{abc} = \varepsilon_i + \varepsilon_j + \varepsilon_k - \varepsilon_a - \varepsilon_b - \varepsilon_c \quad (7)$$

where i,j,k,l and a,b,c,d run over canonical occupied and virtual orbitals, respectively, t^i_a and T^{ij}_{ab} are the converged CCSD cluster amplitudes, ε_r are the canonical orbital energies, and

$$P^{ijk}_{abc}[X^{ijk}_{abc}] = X^{ijk}_{abc} + X^{ikj}_{acb} + X^{kij}_{cab} + X^{kji}_{cba} + X^{jki}_{bca} + X^{jik}_{bac} \quad (8)$$

is a permutation operator.

In order to quantify the rounding errors introduced by single precision arithmetic in the (T) correction, we calculated Hartree−Fock reference functions, CCSD wave functions, and transformed four-index integrals in standard double precision and then made multiple calculations of the (T) correction with single precision random rounding. The numerical stability was estimated from the numerical scattering produced in eight runs of the (T) correction as explained before. For the test cases, we chose SO_2/AV5Z,[57,58] a small molecule with a large basis set, and a naphthalene/VDZ,[59] a larger molecule with a small basis set. The SO_2 molecular geometry was optimized on MP2/AVTZ, and naphthalene was optimized on DF-KS(PBE)[60]/def2-TZVPP.[61,62]

In these calculations it turned out to be essential that the final correlation energy (eq 4) is summed up in double precision when using large basis sets, because otherwise the previously discussed mass truncation error occurs and therefore the results become rather inaccurate (or, alternatively, a single precision compensated summation algorithm should be used[53,54]). This summation is a $O(o^3v^3)$ process, with o/v the number of occupied/virtual orbitals, respectively, and thus its computational expense is insignificant compared to the $O(o^3v^4)$ and $O(o^4v^3)$ steps (eq 5) in the (T) correction. Similarly, for the case of CD-MP2 correlation energies, Vysotskiy et al. found that the final summation should be done in double precision.[37]

The results obtained are given in Table 1. The numbers were obtained in the default (T) triples implementation[63] of Molpro, which does not take any special measures to preserve numerical accuracy; apart from switching back to double precision for the energy summation, no further adjustments were done. Nevertheless, even so the obtained (T) triples energies from single precision calculations were far more accurate than the about ≈0.1 kJ/mol (40 μE_h) which would be sufficient for single point calculations. In fact, we have seen not an amplification of rounding errors but rather a cancellation: The final results obtained are *more* accurate than the machine epsilon for single precision arithmetic (≈6e−8), since all the individual energy contributions in the $O(o^3v^3)$ sum are tiny and well-behaved. These are the entries the single precision arithmetic applies to because, as described, the final summation is done in double precision. This high accuracy is certainly unexpected. We cross-checked the result by performing additional calculations of (T) contributions in which all standard matrix multiplications have been replaced by a routine which explicitly converts its input to single precision FP numbers, calls sgemm (single precision BLAS matrix multiplication), and then converts the results back to double precision. These tests showed that the random rounding results are correct: deviations between final (T) energies with single and double precision matrix multiplications were in the 10th or 11th digit (e.g., for SO_2/AVQZ we obtained −0.039 463 660 530 au and −0.039 463 660 633 au for single and double precision matrix multiplications, respectively).

The results obtained here clearly show that the $O(N^7)$ steps of (T) calculations should be done with single precision arithmetic if that leads to large computational savings and that it is unlikely that this can cause problems in any situation, except maybe for the calculation of higher numerical derivatives.

3.2. Three-Index Molecular Integrals with High Angular Momenta. One of the most fundamental processes in QC is the calculation of integrals over Gaussian-type orbital (GTO) basis functions. The calculation of such integrals is based on deep recurrence relations, in which intermediate results for derivatives of integrals over lower angular momenta (AM) are linearly recombined into integrals over higher AM.[64] Integral evaluation for high AM (≥g) is thus prone to numerical problems, unless designed carefully.

Here we analyze the numerical properties of a very efficient method for evaluating three-center two-electron integrals $(ab|c)$, namely the Obara−Saika scheme[65−67] in Ahlrichs' three-center solid harmonic modification,[68] as implemented in Molpro's AIC integral core.[69] Such integrals occur in both conventional and explicitly correlated QC methods involving the density fitting approximation (see below). For high-accuracy benchmark calculations it would be desirable to be able to calculate integrals with angular momenta of up to $(ki|l)$ (i.e., l = 7, 6, and 8), as they would occur in explicitly correlated QC methods applied with sextuple-ζ orbital basis sets. An interest in such highly accurate calculations was indicated in some previous articles.[70−74]

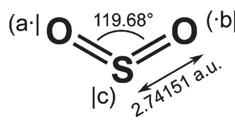
While it is straightforward to write code for calculating these integrals, the numerical properties of the resulting quantities are completely unknown, and furthermore, in this regard there exists no prior experience to draw upon. We thus applied our numerical analysis method to this integration algorithm in order to check its numerical properties. The concrete equations are noted in the Appendix, where the process is also described in more detail.

Table 2. Order of Magnitude (v) and Instability (r) of Integrals $(ab|c)$ with $\alpha = \beta$ and $\gamma = 0.1$ on the SO_2 Geometry (a,b on O, c on S)[a]

$\alpha = \beta = ...$	$(ss\|s)$	$(pp\|p)$	$(dd\|d)$	$(ff\|f)$	$(gg\|g)$	$(hh\|h)$	$(ii\|i)$	$(kk\|k)$	$(ll\|l)$
v, $\alpha = \beta = 10.0$	1.17e−48	1.39e−47	1.40e−46	1.02e−45	5.57e−45	2.37e−44	8.16e−44	2.34e−43	5.71e−43
r, $\alpha = \beta = 10.0$	1.39e−13	1.42e−13	1.20e−13	1.61e−13	1.65e−13	3.39e−13	4.17e−13	2.14e−12	7.01e−12
v, $\alpha = \beta = 1.0$	9.65e−05	1.07e−04	1.03e−04	8.10e−05	5.51e−05	3.39e−05	1.97e−05	1.11e−05	6.19e−06
r, $\alpha = \beta = 1.0$	4.98e−15	3.70e−15	4.45e−15	2.71e−14	9.21e−14	6.69e−13	1.93e−12	1.12e−11	5.93e−11
v, $\alpha = \beta = 0.1$	2.03e+00	2.66e−01	8.38e−02	3.49e−02	1.98e−02	1.14e−02	7.81e−03	5.18e−03	3.99e−03
r, $\alpha = \beta = 0.1$	4.38e−15	3.28e−15	5.38e−15	3.47e−14	7.39e−14	3.37e−13	8.73e−13	2.15e−12	7.88e−12
v, $\alpha = \beta = 0.01$	2.88e+00	9.56e−02	2.46e−02	4.99e−03	1.47e−03	4.71e−04	1.47e−04	5.51e−05	1.82e−05
r, $\alpha = \beta = 0.01$	1.85e−15	1.90e−15	4.79e−15	1.08e−14	1.60e−14	4.71e−14	1.41e−13	1.25e−12	2.73e−12
v, $\alpha = \beta = 0.001$	1.10e+00	4.52e−03	1.07e−03	2.46e−05	6.60e−06	2.46e−07	6.23e−08	3.05e−09	7.07e−10
r, $\alpha = \beta = 0.001$	2.13e−15	1.65e−15	4.42e−15	1.21e−14	1.83e−14	6.43e−14	2.63e−13	4.81e−12	9.35e−12

[a] Where v is the rms batch average of the integral values (see text) and r the relative numerical instability (theoretical optimum: 1e−16).

We tested the integration accuracy by calculating batches of primitive $(ab|c)$ integrals, for a fixed geometry and varying exponents α, β, γ and angular momenta l_a, l_b, l_c (where $l_a \geq l_b$, see Appendix). (An integral batch is the collection of integrals for all combinations of solid harmonic components of a, b and c. E.g., in a (pd|f) batch there are $3 \cdot 5 \cdot 7 = 105$ entries.) The functions a and b were centered on the oxygen atoms of a SO_2 molecule, and the function c was centered on the sulfur atom:



This combination of distributing a, b, and c turned out to be the most challenging one because there is significant overlap between the $(ab|$ and the $|c)$ charge distributions, and yet there is nontrivial angular momentum transfer. The SO_2 geometry was chosen because we previously experienced numerical problems with SO compounds during early stages of the development of F12 methods; SO_2, particularly, has short bond lengths and high exponents in the basis functions. For each test we calculated an integral batch 16 times with random rounding turned on and stored the calculated final integrals. From the 16 runs we then calculated root-mean-square (rms) deviations from the mean for each individual component of the integral batch. As a summary we report the value v = rms *values* of the integrals (to give an indication of average order of magnitude of the integrals in the batch) and the value r = rms of the *rms deviations* for the components, divided by v (r is thus some measure of the relative instability of the calculated values). The best possibly achievable rates are around 1e−16, which is the machine precision. Any value of r below 1e−10 is to be considered good enough for practical use.

Our analysis shows two main results: First, integrals of type $(as|c)$ (i.e., where b is a s function) and general integrals $(ab|c)$ where $\alpha \geq \beta$ can be computed accurately to high angular momenta, even if relatively low exponents are used on either center. For example, Table 2 gives the integral instabilities for $\gamma = 0.1$ and varying $\alpha = \beta$. We can see that, although there does seem to be an exponential increase of the relative instability with angular momentum, the actual values of r are still small (<1e−10) even for very high angular momenta ($l_a = l_b = l_c = 8$), for tiny absolute integral values ($v \approx$ 1e−45 with $\alpha = \beta = 10$) and even for very diffuse exponents like $\alpha = \beta = 0.001$ (which are, in fact, so small that they are already far from reasonable).

Table 3. Order of Magnitude (v) and Relative Instability (r) of Integrals $(gg|s)$ with $\gamma = 0.5$ and Varying α,β on the SO_2 Geometry[a]

$(gg\|s)$, $\gamma = 0.5$	$\beta = 100.0$	$\beta = 10.0$	$\beta = 1.0$	$\beta = 0.1$
v, $\alpha = 100.0$	0.00e+00	6.95e−83	2.54e−10	1.37e−07
r, $\alpha = 100.0$		2.82e−13	4.23e−14	2.40e−11
v, $\alpha = 10.0$	6.95e−83	1.58e−42	4.25e−07	6.95e−05
r, $\alpha = 10.0$	4.25e−11	1.78e−13	2.61e−14	1.98e−13
v, $\alpha = 1.0$	2.54e−10	4.25e−07	8.72e−03	1.58e−02
r, $\alpha = 1.0$	4.93e−07	1.17e−10	9.86e−14	5.33e−14
v, $\alpha = 0.1$	1.37e−07	6.95e−05	1.58e−02	1.42e−01
r, $\alpha = 0.1$	7.81e−04	1.24e−07	2.25e−11	5.28e−14

[a] Note that the table would be symmetric under optimal conditions.

The second main result, however, is disturbing. If the exponent β is much larger than the exponent α, then the integration becomes numerically unstable even for moderate angular momenta (for this it has to be noted that in the integration procedure, as is usually done, angular momentum is first accumulated on center A (where $l_a \geq l_b$) and then transferred to B in the final step, see Appendix). Table 3 shows the average values and instabilities of $(gg|s)$ integrals with varying values of α and β and fixed γ. Note that integrals are supposed to be symmetric regarding the exchange of a and b. We can see that there is a major difference in the stabilities, depending on whether β/α is large or small, and that unacceptably large relative instabilities of almost 1e−3 are already reached in semirealistic cases [$\beta = 100$, $\alpha = 0.1$, $(gg|s)$ integrals]. While this problem is unlikely to show up in standard calculations of valence correlation energies with conventional QC methods (due to the nonpresence of high-AM functions with steep exponents), there are other cases in which it does occur. For example, in F12 treatments, high-exponent high-AM functions are required for the resolution of the identity (RI) approximation, and these are affected by this issue; basis sets for all-electron calculations of higher elements include affected functions (e.g., ANO-RCC for the Rn atom[75] includes f

primitives with exponent 689.88), and calculations involving bond centered functions (often used for calculations of potential energy surfaces[76]) are affected because distance between basis function centers is decreased, and thus, the unstable integrals have a higher relative impact on the energy. The problems get much worse with increasing ratio β/α and with increasing angular momentum l_a and l_b (the l_c dependence, however, is weak). In fact, for angular momenta k and l we have seen cases where the results were completely bogus, and the integral values suggested the presence of non-negligible integrals of size 1e−7, where the actual (negligible) integral values were in the range of 1e−60.

The reason for this asymmetric numerical instability turns out to lie in the contracted transfer equation eq 21, which is applied as final step in almost every efficient integral core in use (in both Rys quadrature[2,77−80] based and Obara−Saika[65−67] based cores). An analysis of the issue and its possible remedies are given in the Appendix. Our current understanding is that this problem cannot easily be solved without both additional computational cost and significant changes to the existing integral programs. An extensive literature search showed that this issue was discovered before,[81] but it does not appear to be well-known.

3.3. Density Fitting Equations in DF-MP2.

Density fitting (DF)[82−85] is an approximation technique widely used to accelerate the evaluation of transformed four-index molecular integrals in a large variety of contexts.[86−96] For example, in DF-MP2,[87] the two-electron exchange integrals $(ia|jb)$ are approximated as

$$(ia|jb) \approx \sum_A D_{ia}^A (A|jb) \tag{9}$$

where i,j denote occupied orbitals, a,b denote virtual orbitals, and A,B are basis functions from an auxiliary fitting basis set. The fitting coefficient D_{ia}^A approximates the $|ia)$ orbital product density such that $|ia) \approx \Sigma_A D_{ia}^A |A)$, and it is formally determined by[92,97]

$$D_{ia}^A = \sum_B [\mathbf{J}^{-1}]_{AB} (B|ia) \tag{10}$$

$$J_{AB} = (A|B) \tag{11}$$

There are multiple ways to evaluate eqs 9 and 10 (in parentheses are the central `LAPACK` operations involved):

(i) An inverse matrix $\mathbf{J}^{-1}$ is explicitly constructed from $\mathbf{J}$'s LU decomposition (`dgetrf` and `dgetri`), and eqs 9 and 10 are evaluated as written.

(ii) Equation 10 is evaluated as a system of linear equations using $\mathbf{J}$'s LU decomposition and pivoted Gaussian elimination (`dgetrf` and `dgetrs`).

(iii) A matrix $\mathbf{J}^{-1/2}$ is evaluated from $\mathbf{J}$'s spectral decomposition (`dsyev`), and eqs 9 and 10 are evaluated in the symmetric form:

$$(ia|jb) \approx \sum_C \tilde{D}_{ia}^C \tilde{D}_{jb}^C \tag{12}$$

$$\tilde{D}_{ia}^C = \sum_A (ia|A)[\mathbf{J}^{-1/2}]_{AC} \tag{13}$$

(iv) A matrix $\mathbf{J}^{-1/2}$ is evaluated as in (iii), and eq 10 is calculated as

$$D_{ia}^A = \sum_B [\mathbf{J}^{-1/2}]_{AB} \left(\sum_C [\mathbf{J}^{-1/2}]_{BC} (C|ia) \right) \tag{14}$$

(i.e., there are two transformations with $\mathbf{J}^{-1/2}$ instead of one with $\mathbf{J}^{-1}$ as done in (i); apart from that, (i) and (iv) are identical).

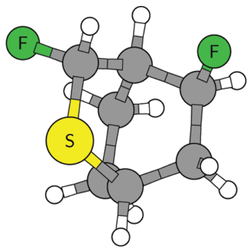


Figure 2. Cage-like molecule used for numerical testing of the DF-MP2 methods. The structure was chosen to be tightly packed yet realistic, in order to create a large redundancy when using diffuse basis sets.

The strategies (i−iii) have been used in various programs (e.g., (i) in refs 98 and 99 (ii) in refs 3 and 100−104 (iii) in refs 86 and 105), with (ii) and (iii) being the most common choices and (iv) is investigated out of curiosity. In any case, the closed-shell DF-MP2 correlation energy is given by

$$E_{\text{MP2}} = \sum_{ijab} \frac{[2(ia|jb) - (ib|ja)](ia|jb)}{\varepsilon_a + \varepsilon_b - \varepsilon_i - \varepsilon_j} \tag{15}$$

where the ε_r are the canonical orbital eigenvalues obtained in the Hartree−Fock calculation.

That (i) works poorly has been found previously, but to our knowledge, a detailed numerical stability study comparing the various methods of calculating $(ia|jb)$ has never been performed. With our statistical analysis method, we can easily determine the impact of the various choices on the final calculation result E_{MP2}.

To that end, we performed random rounding DF-MP2 calculations on an artificial cage-like molecule (Figure 2) using the aug-cc-pVQZ orbital basis set[57] (1264 basis functions) and the associated AVQZ/MP2FIT basis set of Weigend et al.[106] (2506 basis functions). As reference we used a tightly converged DF-RHF wave function, obtained with VQZ/JKFIT[98] as fitting set. This combination of molecule and orbital basis set was chosen because it presents a formidable challenge due to the high redundancy of the basis sets in the closely packed environment (e.g., the ratio between the highest and the lowest eigenvalue of the J_{AB} matrix is 1.2e+11). For each of the four described ways of evaluating the DF equation systems, we performed eight calculations employing double precision random rounding throughout the entire DF-MP2 procedure and measured the final correlation energies obtained in the runs.

The results are shown in Table 4. Here we can see that, not surprisingly, the pivoted Gaussian elimination (ii) is the most accurate way of solving the DF equations, while method (i), based on the explicit calculation of $\mathbf{J}^{-1}$, is the least accurate one. The numerical scattering obtained for method (i) is already large enough to cause significant problems when calculating numerical derivatives (for example, in calculations of vibrational frequencies or molecular properties). Method (i) is thus best avoided. The two methods based on the $\mathbf{J}^{-1/2}$ decomposition, (iii) and (iv), are similarly stable, and although their accuracy is worse than that of the pivoted elimination, it is not much worse, and the errors introduced through them are still acceptable. This is perhaps most surprising for the asymmetric formula (iv) which, after all, is just a re-expression of method (i) with two $\mathbf{J}^{-1/2}$ multiplications instead of one with $\mathbf{J}^{-1}$. While this method is obviously not something one should use for new programs

Table 4. Numerical Stability of the Different Variants of Solving the Density Fitting Equations in DF-MP2[a]

DF-MP2	(i) expl. $\mathbf{J}^{-1}$	(ii) LU/pivoted elim.	(iii) sym. $\mathbf{J}^{-1/2}$	(iv) asym. $\mathbf{J}^{-1/2}\cdot\mathbf{J}^{-1/2}$
#1	−2.004 579 669 344 919	−2.004 579 715 663 295	−2.004 579 715 665 265	−2.004 579 715 673 580
#2	−2.004 579 690 862 029	−2.004 579 715 663 242	−2.004 579 715 669 322	−2.004 579 715 679 456
#3	−2.004 579 777 141 105	−2.004 579 715 662 883	−2.004 579 715 699 197	−2.004 579 715 675 233
#4	−2.004 579 738 407 475	−2.004 579 715 662 652	−2.004 579 715 675 525	−2.004 579 715 669 635
#5	−2.004 579 722 504 768	−2.004 579 715 662 659	−2.004 579 715 661 420	−2.004 579 715 678 294
#6	−2.004 579 728 945 659	−2.004 579 715 663 033	−2.004 579 715 667 403	−2.004 579 715 670 761
#7	−2.004 579 756 469 682	−2.004 579 715 662 888	−2.004 579 715 660 053	−2.004 579 715 684 286
#8	−2.004 579 658 373 851	−2.004 579 715 661 850	−2.004 579 715 669 734	−2.004 579 715 663 029
mean	−2.004 579 717 756 186	−2.004 579 715 662 813	−2.004 579 715 670 990	−2.004 579 715 674 284
rmsd	0.000 000 039 039 628	0.000 000 000 000 426	0.000 000 000 011 605	0.000 000 000 006 174

[a] The rows contain individual energy values (in au) obtained with random rounding for the cage molecule in Figure 2 with the aug-cc-pVQZ basis set. The standard values obtained with an unmodified code are $E(\mathrm{DF}-\mathrm{HF}) = -867.520895102141$ au and $E(\mathrm{DF}-\mathrm{MP2}) = -2.004579708652200$ au.

Table 5. Results Obtained for the DF-MP2 Correlation Energy of the Ethanol Molecule Using Regular (left) and Augmented (right) Basis Sets of Double- ζ to Quadruple-ζ Cardinalities[a]

		cc-pVnZ		aug-cc-pVnZ	
SP steps	card.	mean	rmsd	mean	rmsd
int, DF, asm	DZ	−0.480 233 851	0.000 000 658	−0.507 113 084	0.000 001 830
DF, asm	DZ	−0.480 233 484	0.000 000 592	−0.507 109 818	0.000 001 928
asm	DZ	−0.480 233 835	0.000 000 005	−0.507 105 174	0.000 000 003
(none)	DZ	−0.480 233 824		−0.507 105 163	
int, DF, asm	TZ	−0.601 515 520	0.000 001 835	−0.647 189 353	0.009 018 198
DF, asm	TZ	−0.601 516 124	0.000 001 726	−0.636 752 456	0.009 256 710
asm	TZ	−0.601 514 922	0.000 000 002	−0.612 762 154	0.000 000 001
(none)	TZ	−0.601 514 915		−0.612 762 147	
int, DF, asm	QZ	−0.643 779 496	0.000 040 769	−84.512 702 083	14.999 787 051
DF, asm	QZ	−0.643 715 620	0.000 022 033	−60.558 243 374	5.506 838 535
asm	QZ	−0.643 521 307	0.000 000 001	−0.648 550 399	0.000 000 001
(none)	QZ	−0.643 521 307		−0.648 550 395	

[a] Int (three-index integrals), DF (density fitting), and Asm (assembly) refer to the steps in the DF-MP2 calculation to which single precision random rounding was applied (the other steps used double precision round-to-nearest).

[it is both slower and less accurate than (ii)], it might be used as a quick fix to older programs employing the $\mathbf{J}^{-1}$ construction or, generally, any other case outside the context of density fitting where projection operators or inverse matrices are employed.

A comparison of the random rounding mean values with the results of an unmodified code (see caption of Table 4) shows a small bias of ≈1e−8 H in all cases. As explained previously, this can indicate the presence of a mass truncation error, most likely in the energy summation. As this effect is reflected in a worst case way and still produces negligible errors, we can assume that DF-MP2 is generally numerically very stable as long as the explicit inversion of J_{AB} is avoided.

3.4. Speeding Up DF-MP2 with Mixed Precision Arithmetic. Vysostoskiy and Cederbaum[37] recently performed a numerical analysis of Choleski-decomposition MP2, where they found that CD-MP2 energies can still be evaluated accurately if the CD-analog of the DF assembly step eq 9 is performed in single precision arithmetic. We here perform a similar analysis for DF-MP2 but extend the coverage also to other steps of the total calculation.

The total DF-MP2 method consists of three steps with different formal scaling properties:

(i) The evaluation of two- and three-index molecular integrals [an $O(N^3)$ process, where N is some measure of the molecular size].

(ii) The calculation of the fitting coefficients D_{ia}^{A} according to eq 10 [$O(N^4)$].

(iii) The assembly of the final exchange integrals according to eq 9 [$O(N^5)$].

While for small molecules integral evaluation (i) dominates the computational cost, for large molecules, eventually the assembly step (iii) becomes dominant (usually if more than ≈1000−2000 basis functions are involved).

We thus investigated if mixed precision schemes can be employed to evaluate the DF-MP2 energy accurately. Concretely, we tested the resulting numerical accuracy of the following combinations:

(i) The total DF-MP2 method except for the energy summation (eq 15) and the calculation of $\mathbf{J}^{-1/2}$ is evaluated in single precision. That means that three-center integrals, fitting coefficients, and the final exchange integrals are all evaluated in single precision arithmetic.

(ii) Unlike in (i), the integrals are evaluated in standard double precision arithmetic.

(iii) Unlike in (i) and (ii), also the $\tilde{D}_{ia}^{A}$ fitting coefficients (eq 13) are evaluated in double precision arithmetic. Only the final assembly of exchange integrals is done in single precision random rounding.

In any case, the final summation of the energies (eq 15) is performed in double precision arithmetic, as suggested in ref 37. Additionally, the matrix $[\mathrm{J}]_{AB}$ and its inverse square root are calculated in double precision. The symmetric density fitting formula eq 12 is used. As a test subject we use the ethanol molecule. Again we perform eight calculations with random rounding for each of the three computational schemes (i−iii), and as basis sets, we employ VDZ−VQZ[59] and AVDZ−AVQZ[57] with the same fitting sets as before.

The results are noted in Table 5. We can see that for small, nonaugmented basis sets it appear as if reasonable accuracy can be obtained also when using single precision arithmetic globally. However, it is obvious that the accuracy decreases quickly with increasing size of the basis sets unless both the integrals and the density fitting equations are evaluated in double precision (this effect is even more pronounced for larger systems). When augmented basis sets are employed, the errors due to single precision arithmetic in integrals or density fitting equations are amplified enormously. Already at AVTZ level the results are completely bogus, and at AVQZ level the numerical errors exceed the actual calculation result by orders of magnitude.

If, however, only the assembly step is performed in single precision, then in all cases the results are very accurate. The main difference between the assembly step and the DF equations step is that the assembly step is done in an orthogonal basis set. This is the same effect we have already seen previously for the (T) correction. In fact, this leads us to the conjecture that likely in all parts of calculations which are done in an orthogonal basis, performing the actual core operations in single precision is numerically fine and produces acceptable numerical errors, as long as excessively long linear sums (and thus the mass truncation error) can be avoided. The reason for this is that in an orthogonal basis, the numerical size of the quantities in question carries information about their importance, while in an nonorthogonal basis, the relative importance may still be magnified or diminished by the same order of magnitude as the condition number of the space's overlap matrix.

4. CONCLUSIONS

Previously the means of numerical accuracy analysis for quantum chemistry programs were rather limited, and for this reason, such an analysis was rarely done in practice. However, as our discovery of the instability in the widely used angular momentum transfer equation showed, even established techniques can have serious defects in unusual circumstances. For this reason, better tools, especially simpler tools, and a more frequent application of them were in dire need.

Our applications clearly demonstrate that the proposed method is a practicable way of analyzing complex numerical software packages. Additionally, unlike the previous techniques, it is simple enough to be routinely applicable to newly developed software. We thus believe that our approach is a valuable tool in the development of new QC algorithms and for the identification of persistent problems in algorithms already existing.

APPENDIX

A. Technical Details. On the technical side, the injected code changes the rounding mode by cycling through a table of 2^{16} x87 floating point control words (FPCW), one of which is loaded to the FPU before each arithmetic FP operation. A FPCW[107] is a 16 bit integer whose bits control different aspects of the operating mode of the FPU (e.g., two bits control the rounding mode, and two other bits control the calculation precision of the FPU). Since this table can be modified at run time, one can easily switch between normal calculations (all table entries are "64 bit precision, round to nearest"), random rounding (table contains "64 bit precision, round up" or "64 bit precision, round down" randomly distributed) or reduced precision (e.g., all entries "32 bit precision, round to nearest"), such that a selective investigation of subalgorithms of the program in question can be done.

As noted previously, we generate the numerical noise by changing the FPU rounding modes randomly before each relevant floating point operation. This is realized by injecting the following x86-64 assembler code before each relevant FP operation. (These are the addition/subtraction operations `fadd`, `faddp`, `fiadd`, `fsub`, `fsubp`, `fisub`, `fsubr`, `fsubrp`, `fisubr` and the multiplication/division operations `fmul`, `fmulp`, `fdiv`, `fdivp`, `fdivr`, `fdivrp`, `fprem`, `fprem1`, `fimul`, `fidiv`, `fidivr`. We make sure that the compiler generates only x87 FPU code and does not perform the FP operations by some other means (e.g., SSE instructions)):

```
xchg rax, [LC_CWINDEX]
lea ax, [rax+1]
fldcw word ptr [LC_CWDATA+2*rax]
xchg rax, [LC_CWINDEX]
```

The variable `LC_CWDATA` is the start address of an array with 2^{16} entries of FPU control words. The variable `LC_CWINDEX` is a counter which contains the current index in this table. In each execution of the injected code, the following happens:

(i) The `xchg` instruction exchanges the current values of the general purpose register `rax` and of the memory location at `LC_CWINDEX`. This allows us to modify the table index (subsequently in rax) and to keep a copy of the original register value.

(ii) The load effective address (`lea`) instruction increments ax (the lower 16 bits of `rax`) by one and resets it to zero if it overflows 2^{16}. The effect of the instruction is equal to "`add ax,1`", apart from `lea` not affecting CPU flags. Due to the implicit reset to zero on overflow, we can directly use `rax` as index for the 2^{16} entry rounding mode table.

(iii) The load floating point control word (`fldcw`) instruction reads a FPCW from our table and activates its use by the FPU. Thus, all following FPU operations will be performed with a precision and rounding mode as specified by the FPCW, until another FPCW is loaded.

(iv) Finally, the second `xchg` instruction resets the `rax` register back to its original value and at the same time writes back our updated table index.

All of the injected instructions preserve the CPU flags register, and thus can be inserted without affecting any of the surrounding code, apart from the FPU control word.

The entries of the table `LC_CWDATA` can be changed at runtime by the program itself. For example, when switching into "random rounding" mode, a function is called which distributes "round up" and "round down" control words randomly in the table using the KISS pseudorandom number generator[108] in David Jones JKISS modification.[109] In order to investigate an isolated sub-task of the program, the table is set to "random rounding" before beginning the sub-task and is reset to "default rounding" afterwards. This way all FP operations between the two calls are subject to the random rounding behavior, and all other FP operations proceed as normal.

In practice it may be sufficient to inject code for FPU random rounding mode changes only before additions and subtractions, since these operations are most prone to causing accuracy loss. Because the rounding mode has a "state nature" and remains active until set to something different, the other instructions would still be affected by random rounding, since they occur interleaved with addition/subtraction operations. Here, however, the random rounding was also applied to multiplications and divisions. Apart from that, we reset the rounding mode to "64 bit, nearest" before calling mathematical library functions, like `exp` or `log`. The reason for this is that the implementation of these functions may depend on exact rounding properties of intermediate values, and thus the random rounding might misrepresent the accuracy loss caused by them.

B. Equations for the Molecular Integrals. A Cartesian GTO basis function has the functional form

$$a_j(\mathbf{r}) = (r_x - A_x)^{a_x}(r_y - A_y)^{a_y}(r_z - A_z)^{a_z} \times \sum_\alpha \Gamma_{j\alpha} \exp(-\alpha\|\mathbf{r}-\mathbf{A}\|^2) \tag{16}$$

where $\mathbf{A}$ is the center of the basis function, α runs over primitive exponents, j runs over contractions (with contraction coefficients $\Gamma_{j\alpha}$), and $\mathbf{a} = (a_x,a_y,a_z)$ indexes the Cartesian powers of the polynomial. In the actual basis functions used, the polynomial prefactors are transformed into solid harmonics $S^{l_a}_{m_a}(\mathbf{r} - \mathbf{A})$.[64] Similar definitions hold for $b_k(\mathbf{r})$ and $c_l(\mathbf{r})$, with their own centers and exponents. Using the notation of Ahlrichs,[67,68] the three-center primitive integral over the scalar integral kernel $K(t)$:

$$(\mathbf{ab}|\mathbf{c}) = \iint a(\mathbf{r}_1)b(\mathbf{r}_1)K(\|\mathbf{r}_1-\mathbf{r}_2\|)c(\mathbf{r}_2)\mathrm{d}^3r_1\mathrm{d}^3r_2 \tag{17}$$

is then calculated by the following recurrence relations:

$$(\mathbf{00}|\mathbf{0})^m = \left(\frac{\pi}{\zeta+\gamma}\right)^{3/2}\exp\left(-\frac{\alpha\beta}{\alpha+\beta}\|\mathbf{A}-\mathbf{B}\|^2\right)G_m(\rho,T) \tag{18}$$

$$(\mathbf{a0}|\mathbf{0})^m = (P_i - A_i)((\mathbf{a}-\mathbf{1}_i)\mathbf{0}|\mathbf{0})^m + \frac{a_i - 1}{2\zeta}((\mathbf{a}-\mathbf{2}_i)\mathbf{0}|\mathbf{00})^m - \frac{\rho}{\zeta}(P_i - C_i)((\mathbf{a}-\mathbf{1}_i)\mathbf{0}|\mathbf{0})^{m+1} - \frac{\rho(a_i-1)}{\zeta^2}((\mathbf{a}-\mathbf{2}_i)\mathbf{0}|\mathbf{0})^{m+1} \tag{19}$$

$$(\mathbf{a0}|\mathbf{c})^m = \frac{\rho}{\gamma}(P_i - C_i)(\mathbf{a0}|\mathbf{c}-\mathbf{1}_i)^{m+1} + \frac{a_i}{2(\eta+\gamma)}((\mathbf{a}-\mathbf{1}_i)\mathbf{0}|\mathbf{c}-\mathbf{1}_i)^{m+1} \tag{20}$$

$$(\mathbf{ab}|\mathbf{c}) = ((\mathbf{a}+\mathbf{1}_i)(\mathbf{b}-\mathbf{1}_i)|\mathbf{c}) - (B_i - A_i)(\mathbf{a}(\mathbf{b}-\mathbf{1}_i)|\mathbf{c}) \tag{21}$$

where $i = x, y,$ or z (no summation), $G_m(\rho,T)$ is a family of scalar functions depending only on the integral kernel K,[67] and the other occurring intermediates are $\zeta = \alpha + \beta$ and

$$\rho = \frac{\zeta\gamma}{\zeta+\gamma} \quad \mathbf{P} = \frac{\alpha\mathbf{A}+\beta\mathbf{B}}{\alpha+\beta} \quad T = \rho\|\mathbf{P}-\mathbf{C}\|^2. \tag{22}$$

The final integrals are $(\mathbf{ab}|\mathbf{c}) = (\mathbf{ab}|\mathbf{c})^{m=0}$. Equation 21 is called the transfer equation, because it transfers angular momentaum from a to b. Note that it does not contain any exponents and thus can be applied to contracted integrals instead of primitives.

In our implementation, the eqs 18−20, followed by the solid harmonic transformation of c, are evaluated for each combination of primitive integrals. The primitive integrals $(\mathbf{a0}|\mathbf{c})$ (with total AM from l_a to $l_a + l_b$ in the first label) are then accumulated into their respective general contractions. In eq 19 we evaluate $(\mathbf{P} - \mathbf{A})$ as $(\beta/(\alpha+\beta))(\mathbf{B} - \mathbf{A})$, as this increases the numerical stability when α/β is large. Finally, for each already contracted intermediate $(\mathbf{a0}|\mathbf{c})$, first the label a is transformed to solid harmonics, then the transfer equation eq 21 is applied to form $(\mathbf{ab}|\mathbf{c})$,and then label b is transformed to solid harmonics.

C. On the Asymmetric Instability of the Contracted Transfer Equation. As noted previously, the discovered asymmetric numerical instability is caused by the contracted transfer equation (eq 21), which is used as final step in not only the described integral core but also in almost every other one. This equation is popular because it can be applied to already contracted integral intermediates, as already pointed out, and can thus lead to significant time savings.

In the one-dimensional case, the numeric instability can be described as follows: We are calculating the desired integrals

$$(l_a, l_b|l_c) = \iint (x_1 - A_x)^{l_a}\mathrm{e}^{-\alpha(x_1 - A_x)^2} \cdot (x_1 - B_x)^{l_b}\mathrm{e}^{-\beta(x_1 - B_x)^2} \ldots \mathrm{d}^3x_1\mathrm{d}^3x_2 \tag{23}$$

by first calculating $(l_a + l_b,0|l_c)$, that is

$$\iint (x_1 - A_x)^{l_a + l_b}\mathrm{e}^{-\alpha(x_1 - A_x)^2}\mathrm{e}^{-\beta(x_1 - B_x)^2} \ldots \tag{24}$$

and then re-expressing some of the $(x_1 - A_x)$ as $(x_1 - B_x) + (B_x - A_x)$ (this is what this contracted transfer relation is doing). The problem now is this: if the exponent on B is much larger than that on A, then in the integral only such x_1 will have contributions where x_1 is very close to B. This means that, on average, $(x_1 - B_x)$ is a small quantity, which, we express as a difference of two large quantities, $(x_1 - A_x)$ and $(B_x - A_x)$; this situation inevitably leads to accuracy loss. The exponents do not explicitly occur in these transfer formulas, but the integral intermediates already contain them implicitly; namely encoded as the difference between the various components of the $l_a + l_b$ batch.

Since the reverse direction $\alpha > \beta$ works fine (see above) and since we have the integral identity $(ab|c) = (ba|c)$, one could think that this problem can simply be solved by always accumulating the angular momentum on the center with the large exponent first (i.e., exchanging a and b if α is much smaller than β). However, this solution leads to various technical problems, because: (i) the transfer equation is generally implemented only for $l_a \geq l_b$ (since this is much more efficient than $l_a < l_b$), and thus if $l_a \neq l_b$, the order of a and b is already fixed by their angular momenta, and more importantly, (ii) in general one deals with contracted basis functions, in which primitive integrals with several exponents are set into some fixed linear combination. Thus both $\alpha < \beta$ and $\beta > \alpha$ will occur at the same time for a single contracted integral (and the transfer equation is applied to contracted intermediates).

A preliminary investigation suggests that the quantity:

$$r = \left(\frac{\|P-B\|}{\|A-B\|}\right)^{l_b} = \left(\frac{\alpha}{\alpha+\beta}\right)^{l_b}$$

is a good indicator of the relative accuracy loss due to the angular momentum transfer. Thus, for the moment the best workaround to the problem would be to estimate the accuracy loss with this relation for each primitive combination and, if it turns out to be unacceptable for some combination to do the computationally expensive angular momentum transfer from b to a for these primitives, additionally to the a to b transfer for the other primitives in the contraction.

Our current understanding is that this issue applies to almost every efficient integral core in use and that it cannot easily be solved without additional computational cost. We might revisit this issue in the future.

AUTHOR INFORMATION

Corresponding Author

*E-mail: knizia@theochem.uni-stuttgart.de.

ACKNOWLEDGMENT

The authors thank Toru Shiozaki for helpful discussions on integral evaluation and Fabian Giesen for helpful discussions on random number generators. This work was funded by the DFG within the SRC SimTech at Stuttgart University.

REFERENCES

(1) Knuth, D. E. *The Art of Computer Programming. Vol. 2: Seminumerical Algorithms*, 2nd ed.; Addison-Wesley: Upper Saddle River, NJ, 1980; pp 213−223.
(2) Lindh, R.; Ryu, U.; Liu, B. *J. Chem. Phys.* **1991**, *95*, 5889.
(3) Werner, H.-J.; Adler, T. B.; Manby, F. R. *J. Chem. Phys.* **2007**, *126*, 164102.
(4) Knizia, G.; Adler, T. B.; Werner, H.-J. *J. Chem. Phys.* **2009**, *130*, 054104.
(5) Rauhut, G.; Knizia, G.; Werner, H.-J. *J. Chem. Phys.* **2009**, *130*, 054105.
(6) Huang, X.; Valeev, E. F.; Lee, T. J. *J. Chem. Phys.* **2010**, *133*, 244108.
(7) Bromley, M. W. J.; Mitroy, J. *Int. J. Quantum Chem.* **2007**, *107*, 1150.
(8) Ekström, U.; Visscher, L.; Bast, R.; Thorvaldsen, A. J.; Ruud, K. *J. Chem. Theory Comput.* **2010**, *6*, 1971.
(9) Takashima, H.; Amisaki, T.; Kitamura, K.; Nagashima, U. *Comput. Phys. Commun.* **2002**, *148*, 182.
(10) Helgaker, T.; Jørgensen, P.; Olsen, J. *Molecular Electronic Structure Theory*; Wiley: Chichester, U.K., 2000; pp 336−397.
(11) Bolhuis, P. G.; Chandler, D.; Dellago, C.; Geissler, P. L. *Annu. Rev. Phys. Chem.* **2002**, *53*, 291.
(12) Weigend, F. *Phys. Chem. Chem. Phys.* **2002**, *4*, 4285.
(13) Weigend, F. *J. Comput. Chem.* **2007**, *29*, 167.
(14) Weigend, F.; Köhn, A.; Hättig, C. *J. Chem. Phys.* **2002**, *116*, 3175.
(15) Hättig, C. *Phys. Chem. Chem. Phys.* **2005**, *7*, 59.
(16) Valeev, E. F. *Chem. Phys. Lett.* **2004**, *395*, 190.
(17) Yousaf, K. E.; Peterson, K. A. *J. Chem. Phys.* **2008**, *129*, 184108.
(18) Yousaf, K. E.; Peterson, K. A. *Chem. Phys. Lett.* **2009**, *476*, 303.
(19) Werner, H.-J.; Knowles, P. J. *J. Chem. Phys.* **1985**, *82*, 5053.
(20) Knowles, P. J.; Werner, H.-J. *Chem. Phys. Lett.* **1985**, *115*, 259.
(21) Werner, H.-J.; Knowles, P. J. *J. Chem. Phys.* **1988**, *89*, 5803.
(22) Asadchev, A.; Allada, V.; Felder, J.; Bode, B. M.; Gordon, M. S.; Windus, T. L. *J. Chem. Theory Comput.* **2010**, *6*, 696.
(23) Ufimtsev, I. S.; Martinez, T. J. *J. Chem. Theory Comput.* **2008**, *4*, 222.
(24) Ufimtsev, I. S.; Martinez, T. J. *Comput. Sci. Eng.* **2008**, *10*, 26.
(25) Luehr, N.; Ufimtsev, I. S.; Martinez, T. J. *J. Chem. Theory Comput.* **2011**, *7*, 949.
(26) Ufimtsev, I. S.; Martinez, T. J. *J. Chem. Theory Comput.* **2009**, *5*, 1004.
(27) Ufimtsev, I. S.; Martinez, T. J. *J. Chem. Theory Comput.* **2009**, *5*, 2619.
(28) Yasuda, K. *J. Comput. Chem.* **2007**, *29*, 334.
(29) Yasuda, K. *J. Chem. Theory Comput.* **2008**, *4*, 1230.
(30) Vogt, L.; Olivares-Amaya, R.; Kermes, S.; Shao, Y.; Amador-edolla, C.; Aspuru-Guzik, A. *J. Phys. Chem. A* **2008**, *112*, 2049.
(31) Olivares-Amaya, R.; Watson, M. A.; Edgar, R. G.; Vogt, L.; Shao, Y.; Aspuru-Guzik, A. *J. Chem. Theory Comput.* **2010**, *6*, 135.
(32) Watson, M.; livares Amaya, R.; Edgar, R. G.; Aspuru-Guzik, A. *Comput. Sci. Eng.* **2010**, *12*, 40.
(33) Brown, P.; Woods, C.; McIntosh-Smith, S.; Manby, F. R. *J. Chem. Theory Comput.* **2008**, *4*, 1620.
(34) Brown, P.; Woods, C. J.; McIntosh-Smith, S.; Manby, F. R. *J. Comput. Chem.* **2010**, *31*, 2008.
(35) Muller, J.-M.; Brisebarre, N.; de Dinechin, F.; Jeannerod, C.-P.; Lefèvre, V.; Melquiond, G.; Revol, N.; Stehlè, D.; Torres, S. *Handbook of Floating-Point Arithmetic*; Birkhäuser: Boston, 2010; pp 281−282 and 288−302.
(36) Raghavachari, K.; Trucks, G. W.; Pople, J. A.; Head-Gordon, M. *Chem. Phys. Lett.* **1989**, *157*, 479.
(37) Vysotskiy, V. P.; Cederbaum, L. S. *J. Chem. Theory Comput.* **2011**, *7*, 320.
(38) Rump, M. Algorithms for verified inclusions: Theory and Practice.*Reliability in Computing: The Role of Interval Arithmetic in Methods in Scientific Computing*; Moore, R. E., Ed.; Academic Press: Boston, 1988; pp 109−126.
(39) Loh, E.; Walster, G. W. *Reliab. Comput.* **2002**, *8*, 245.
(40) Moore, R. E.; Bierbaum, F. *Methods and applications of interval analysis*; SIAM: Philadelphia, PA, 1979; pp 9−16.
(41) Albertsena, N. C.; Chesneaux, J.-M.; Christiansen, S.; Wirgin, A. *Math. Comput. Simul.* **1999**, *48*, 307.
(42) Vignes, J.; Porte, M. L. *Proc. IFIP Congr.* **1974**, 610.
(43) Vignes, J. *Math. Comput. Simul.* **1993**, *35*, 233.
(44) Vignes, J. *Numer. Algor.* **2004**, *37*, 377.
(45) Scott, N.; Jezequel, F.; Denis, C.; Chesneaux, J.-M. *Comput. Phys. Commun.* **2007**, *176*, 507.
(46) Guilain, S.; Vignes, J. *Math. Comput. Simul.* **1994**, *37*, 73.
(47) Abbasbandy, S.; Araghi, M. A. F. *Appl. Numer. Math.* **2004**, *50*, 279.
(48) Jezequel, F.; Chesneaux, J.-M. *Comput. Phys. Commun.* **2008**, *178*, 933.
(49) Barlow, R. J. *Statistics: A Guide to the Use of Statistical Methods in the Physical Sciences*; Wiley: Chichester, U.K., 1989; pp 49−51.
(50) Li, W.; Simon, S. Numerical Error Analysis for Statistical Software on Multi-Core Systems. In Proceedings of Compstat'2010, Paris, France, August 22−27, 2010; Lechevallier, Y.; Saporta, G., Eds.; Physica: Heidelberg, Germany, 2010; pp 1287−1294.

(51) Werner, H.-J.; Knowles, P. J.; Knizia, G.; Manby, F. R.; Schütz, M.; Celani, P.; Korona, T.; Lindh, R.; Mitrushenkov, A.; Rauhut, G.; Shamasundar, K. R.; Adler, T. B.; Amos, R. D.; Bernhardsson, A.; Berning, A.; Cooper, D. L.; Deegan, M. J. O.; Dobbyn, A. J.; Eckert, F.; Goll, E.; Hampel, C.; Hesselmann, A.; Hetzer, G.; Hrenar, T.; Jansen, G.; Köppl, C.; Liu, Y.; Lloyd, A. W.; Mata, R. A.; May, A. J.; McNicholas, S. J.; Meyer, W.; Mura, M. E.; Nicklass, A.; O'Neill, D. P.; Palmieri, P.; Pflüger, K.; Pitzer, R.; Reiher, M.; Shiozaki, T.; Stoll, H.; Stone, A. J.; Tarroni, R.; Thorsteinsson, T.; Wang, M.; Wolf, A. *MOLPRO*, development version 2010.2, a package of ab initio programs; University College Cardiff Consultants Limited: Wales, U.K., 2010; http://www.molpro.net.

(52) Li, W.; Simon, S. On the Numerical Sensitivity of Computer Simulations on Hybrid and Parallel Computing Systems. In Proceedings of 2011 International Conference on High Performance Computing Simulation, Istanbul, Turkey, July 4−8, 2011, IEEE: New York, 2011; accepted.

(53) Kahan, W. *Commun. ACM* **1965**, *8*, 40.

(54) Dekker, T. J. *Numer. Math.* **1971**, *18*, 224.

(55) Rendell, A. P.; Lee, T. J.; Komornicki, A. *Chem. Phys. Lett.* **1991**, *178*, 462.

(56) Constans, P.; Ayala, P. Y.; Scuseria, G. E. *J. Chem. Phys.* **2000**, *113*, 10451.

(57) Kendall, R. A.; Dunning, T. H., Jr.; Harrison, R. J. *J. Chem. Phys.* **1992**, *96*, 6796.

(58) Woon, D. E.; Dunning, T. H. *J. Chem. Phys.* **1993**, *98*, 1358.

(59) Dunning, T. H., Jr. *J. Chem. Phys.* **1989**, *90*, 1007.

(60) Perdew, J. P.; Burke, K.; Ernzerhof, M. *Phys. Rev. Lett.* **1996**, *77*, 3865.

(61) Weigend, F.; Ahlrichs, R. *Phys. Chem. Chem. Phys.* **2005**, *7*, 3297.

(62) Weigend, F. *Phys. Chem. Chem. Phys.* **2006**, *8*, 1057.

(63) Deegan, M. J. O.; Knowles, P. J. *Chem. Phys. Lett.* **1994**, *227*, 321.

(64) Helgaker, T.; Jørgensen, P.; Olsen, J. *Molecular Electronic Structure Theory*; Wiley: Chichester, U.K., 2000; pp 207−218.

(65) Obara, S.; Saika, A. *J. Chem. Phys.* **1986**, *84*, 3963.

(66) Obara, S.; Saika, A. *J. Chem. Phys.* **1988**, *89*, 1540.

(67) Ahlrichs, R. *Phys. Chem. Chem. Phys.* **2006**, *8*, 3072.

(68) Ahlrichs, R. *Phys. Chem. Chem. Phys.* **2004**, *6*, 5119.

(69) Werner, H.-J.; Knizia, G.; Manby, F. R. *Mol. Phys.* **2010**, *109*, 407.

(70) Feller, D.; Peterson, K. A.; Hill, J. G. *J. Chem. Phys.* **2010**, *133*, 184102.

(71) Hill, J. G.; Peterson, K. A.; Knizia, G.; Werner, H.-J. *J. Chem. Phys.* **2009**, *131*, 194105.

(72) Hill, J. G.; Mazumder, S.; Peterson, K. A. *J. Chem. Phys.* **2010**, *132*, 054108.

(73) Lique, F.; Klos, J.; Hochlaf, M. *Phys. Chem. Chem. Phys.* **2010**, *12*, 15672.

(74) Barnes, E. C.; Petersson, G. A. *J. Chem. Phys.* **2010**, *132*, 114111.

(75) Roos, B. O.; Lindh, R.; Malmqvist, P.-A.; Veryazov, V.; Widmark, P.-O. *J. Phys. Chem. A* **2005**, *108*, 2851.

(76) Cybulski, S. M.; Toczyłowski, R. R. *J. Chem. Phys.* **1999**, *111*, 10520.

(77) Rys, J.; Dupuis, M.; King, H. F. *J. Comput. Chem.* **1983**, *4*, 154.

(78) Ten-no, S. *Chem. Phys. Lett.* **2004**, *398*, 56.

(79) Shiozaki, T. *Chem. Phys. Lett.* **2009**, *479*, 160.

(80) Flocke, N.; Lotrich, V. *J. Comput. Chem.* **2008**, *29*, 2722.

(81) Ishida, K. *J. Chem. Phys.* **1993**, *98*, 2176.

(82) Boys, S. F.; Shavitt, I. *Report WIS-AF-13*; University of Wisconsin: Madison, WI, 1959.

(83) Whitten, J. L. *J. Chem. Phys.* **1973**, *58*, 4496.

(84) Baerends, E. J.; Ellis, D. E.; Ros, P. *Chem. Phys.* **1973**, *2*, 41.

(85) Dunlap, B. I.; Connolly, J. W. D.; Sabin, J. R. *J. Chem. Phys.* **1979**, *71*, 3396.

(86) Weigend, F.; Häser, M. *Theor. Chim. Acta* **1997**, *97*, 331.

(87) Weigend, F.; Häser, M.; Patzelt, H.; Ahlrichs, R. *Chem. Phys. Lett.* **1998**, *294*, 143.

(88) Schütz, M.; Hetzer, G.; Werner, H.-J. *J. Chem. Phys.* **1999**, *111*, 5691.

(89) Schütz, M. *J. Chem. Phys.* **2000**, *113*, 9986.

(90) Schütz, M.; Werner, H.-J. *J. Chem. Phys.* **2001**, *114*, 661.

(91) Schütz, M. *Phys. Chem. Chem. Phys.* **2002**, *4*, 3941.

(92) Manby, F. R. *J. Chem. Phys.* **2003**, *119*, 4607.

(93) Ten-no, S.; Manby, F. R. *J. Chem. Phys.* **2003**, *119*, 5358.

(94) Neese, F.; Schwabe, T.; Grimme, S. *J. Chem. Phys.* **2007**, *126*, 124115.

(95) Neese, F.; Wennmohs, F.; Hansen, A. *J. Chem. Phys.* **2009**, *130*, 114108.

(96) Neese, F.; Hansen, A.; Liakos, D. G. *J. Chem. Phys.* **2009**, *131*, 064103.

(97) Vahtras, O.; Almlöf, J.; Feyereisen, M. W. *Chem. Phys. Lett.* **1993**, *213*, 514.

(98) Weigend, F. *Phys. Chem. Chem. Phys.* **2002**, *4*, 4285.

(99) Hättig, C.; Weigend, F. *J. Chem. Phys.* **2000**, *113*, 5154.

(100) Schütz, M.; Manby, F. R. *Phys. Chem. Chem. Phys.* **2003**, *5*, 3349.

(101) Polly, R.; Werner, H.-J.; Manby, F. R.; Knowles, P. J. *Mol. Phys.* **2004**, *102*, 2311.

(102) Schütz, M.; Werner, H.-J.; Lindh, R.; Manby, F. R. *J. Chem. Phys.* **2004**, *121*, 737.

(103) Werner, H.-J.; Manby, F. R. *J. Chem. Phys.* **2006**, *124*, 054114.

(104) Knizia, G.; Werner, H.-J. *J. Chem. Phys.* **2008**, *128*, 154103.

(105) Köhn, A.; Hättig, C. *J. Chem. Phys.* **2003**, *119*, 5021.

(106) Weigend, F.; Köhn, A.; Hättig, C. *J. Chem. Phys.* **2002**, *116*, 3175.

(107) *Intel 64 and IA-32 Architectures Software Developer's Manual*; Volume 1: Basic Architecture. Intel, 2009; ch. 8.1.5 "x87 FPU Control Word"; Intel: Santa Clara, CA; http://www.intel.com/products/processor/manuals/. Accessed March 22, 2011.

(108) Marsaglia, G. sci.math newsgroup post, message-id <lSidna-cF9d6F8ajXTWcqg@comcast.com>. 1999; http://groups.google.com/group/sci.math/msg/9959175f66dd138f. Accessed March 22, 2011).

(109) Jones, D. *Good Practice in (Pseudo) Random Number Generation for Bioinformatics Applications*. 2010);www.cs.ucl.ac.uk/staff/d.jones/GoodPracticeRNG.pdf. Accessed March 22, 2011).