Efficient Electronic Integrals and their Generalized Derivatives for Object Oriented Implementations of Electronic Structure Calculations

N. FLOCKE, V. LOTRICH
Quantum Theory Project, Department of Chemistry and Physics, University of Florida, Gainesville, Florida 32611

Received 3 December 2007; revised 20 March 2008; accepted 24 March 2008
DOI 10.1002/jcc.21018
Published online in Wiley InterScience (www.interscience.wiley.com).

Abstract: For the new parallel implementation of electronic structure methods in ACES III (Lotrich et al., in preparation) the present state-of-the-art algorithms for the evaluation of electronic integrals and their generalized derivatives were implemented in new object oriented codes with attention paid to efficient execution on modern processors with a deep hierarchy of data storage including multiple caches and memory banks. Particular attention has been paid to define proper integral blocks as basic building objects. These objects are stand-alone units and are no longer tied to any specific software. They can hence be used by any quantum chemistry code without modification. The integral blocks can be called at any time and in any sequence during the execution of an electronic structure program. Evaluation efficiency of these integral objects has been carefully tested and it compares well with other fast integral programs in the community. Correctness of the objects has been demonstrated by several application runs on real systems using the ACES III program.

 J Comput Chem 00: 000–000, 2008

Key words: Quantum chemistry integral evaluation; open-ended *n*-th order integral derivatives; quantum chemistry ab-initio programs; object oriented computation and software design

Introduction

Efficient gaussian type orbital (GTO) integral evaluation is crucial to the success of any direct quantum chemistry code that recalculates the integrals during every iteration cycle either at Hartree-Fock or at correlated Coupled Cluster level. Any excessive time needed for integral evaluation can seriously degrade the performance of any direct serial or parallel code. Recalculation of integrals, rather than storing them on disk, has become the standard approach in quantum chemistry programs for two reasons: limited disk size and slow disk retrieval.

Evaluation of integrals over GTOs has a long and rich history (P. Gill gives a good summary of articles up to 1993[1]). Early formulations in the 1960s were based on Boys original article,[2] in which the integrals are expressed in terms of auxiliary functions. However, this approach leads to inefficient and slow algorithms, particularly for integrals involving high angular momentum quantum numbers. A major improvement was the introduction of the Rys quadrature method,[3–5] in which the integrals are expressed in terms of so-called two-dimensional integrals, which are themselves evaluated using vertical (VRR) and horizontal recurrence relations (HRR). The Rys method needs the evaluation of quadrature roots and weights of polynomials involving the nonclassical Rys weight function, which are generally time consuming to compute. Hence, the Rys method is most suitable for higher angular momentum integrals, where the time to compute the roots and weights is small compared with other integral manipulations. Another important integral evaluation method is the McMurchie-Davidson scheme (MD).[6] In this method, the integrals are first computed in Hermite Gaussian functions and are later back-transformed to the Cartesian or spherical Gaussian basis. A scheme in which uncontracted Cartesian integrals are evaluated using only few-step recurrence relations of mixed VRR and HRR type on basic *s*-type integrals was presented by Obara and Saika (OS)[7] and implicitly in the context of integral derivatives by Schlegel a few years earlier.[8] The OS scheme is very suitable for vectorization and has been further analyzed and optimized by Head-Gordon and Pople (HGP)[9] by reducing the number of necessary recurrence relation steps. However, as the angular momentum increases, memory requirements for storing all intermediate integrals becomes a burden and the OS and HGP procedures are thus problematic for integrals involving higher than *f*-type functions.

Correspondence to: N. Flocke; e-mail: flocke@qtp.ufl.edu
Contract/grant sponsor: US Department of Defence's High Performance Computing Modernization Program (HPCMP)

The general consensus reached so far, after decades of experience in coding electronic integral evaluation packages, is that each particular approach works best for particular angular momentum combinations. For very large angular momentum integrals the Rys approach is the method of choice. For lower angular momenta it seems better to use OS or HGP related schemes. Several well-known quantum chemistry codes use either one or a mixture of these three basic approaches. The decision as to which method to choose is further complicated by the fact that the final integrals needed are generally between contracted GTOs, i.e., linear combinations of several primitive GTO functions. Hence contraction steps must be performed during integral evaluation and the time for their execution must be optimized with respect to the primitive integral calculation. Another crucial step is the Cartesian to spherical transformation in case integrals over spherical GTOs are needed.

Recently, it has been shown that the Rys approach can be sped up considerably by using the reduced multiplication scheme, in which products of 2D integrals are reused to assemble different primitive integrals.[10] Using this scheme, computational savings are more pronounced for high angular momentum integrals, but also low angular momentum integral evaluation profits from this scheme by avoiding unnecessary multiplications with intermediate integrals equal to one. There is also considerable experience in using the Rys method to evaluate integral derivatives.[11–13] The Rys approach is very economical as far as memory needs is concerned, especially for higher order derivatives. Also usage of the Rys algorithm for both the original integrals and their derivatives allows for considerable code development savings, as many routines can be reused for evaluating the derivatives as well.

For the ACES III integral program, it was thus decided to focus on the Rys integral calculation scheme and to consider, during the design and implementation phase, the experiences accumulated over the last three decades. Particular attention has been paid to the proper layout of the code such that cache memory is used effectively. While the basic strategy in integral evaluation is nowadays well-known theoretically, the proper usage of cache, which ultimately leads to peak performance, is not so well known. It is in this area where considerable performance improvements can still be made. Maximizing the number of fat loops with unit stride access between all its elements is the ultimate goal for a proper integral package design. Another important issue, particularly for designing parallel quantum chemistry programs, is the clean separation of integral evaluation objects from the rest of the code. This allows for the objects to concentrate purely on the task of integral evaluation without additional bookkeeping and decisions related to other nonintegral quantities. An example of the latter situation would be the passing of Fock density matrix elements to the integral evaluation routines for screening purposes during the SCF procedure. This adds more functionality to the integral objects but at the same time brings down their performance. As a general rule, the more functional generality the computational objects have the slower their performance. The ACES III integral evaluation routines are pure objects in the sense that they know nothing about quantities that are not strictly related to the integral calculation process, and very good CPU performance in integral evaluation has been achieved using these objects in the parallel ACES III framework.

Although integral evaluation algorithms have by now been extensively and exhaustively discussed in the literature, integral evaluation research is still far from being over. In recent years, several groups have tried to add to and/or improve certain aspects of integral computation. In connection with the Rys quadrature method, useful integral transforms have been established for certain classes of molecular property integrals.[14] Another effort in reducing the overall integral evaluation FLOP count, also related to the Rys quadrature method, was geared toward improvement of evaluating integrals between highly contracted basis sets.[15] In essence it combines a 5-term 6D ERI recursion formula with the contraction process to achieve a more favorable FLOP count during ERI evaluation. Recent papers by Japanese researchers[16,17] showed overall integral evaluation timing improvements when considering evaluation of the entire set of integrals. Improvement is achieved by realizing that large molecular structures are composed of only a handful of different atoms, hence those basis set quantities relevant for integral evaluation which are independent of the atomic coordinates can be calculated beforehand in two-index loops between all possible atom pairs and stored. Savings in calculations can occur for example during the integral contraction steps, where the contraction coefficients are independent of the atomic coordinates and can hence be premultiplied for all atomic pairs. For our ACES III integral package, this prestoring scheme of large amounts of data seemed to be less attractive, not only because of larger memory demands outside the integral routines but also because cache-efficient implementation of the 4-index contraction scheme is best realized in quarter transformation steps, as explained later on in the integral contraction section, requiring separate handling of the contraction coefficients.

Integral and Integral Derivative Evaluation

Electron Repulsion Integrals

We will start our integral evaluation discussion on the two-electron repulsion integrals (ERI). All basic concepts will be introduced here. Formulas for the one-electron integrals are very similar to the ones for the ERIs but much simpler. A Cartesian ERI integral block is defined as a set of six-dimensional integrals between four sets of Cartesian gaussian type orbital (CGTO) functions over the two electron coordinates $\mathbf{r}_1$ and $\mathbf{r}_2$:

$$\mathrm{ERI}_{\mathrm{block}} = \iint \phi^{\mathbf{A}}_{\boldsymbol{\alpha},L_a}(\mathbf{r}_1)\phi^{\mathbf{B}}_{\boldsymbol{\beta},L_b}(\mathbf{r}_1)\frac{1}{|\mathbf{r}_1-\mathbf{r}_2|}\phi^{\mathbf{C}}_{\boldsymbol{\gamma},L_c} \times (\mathbf{r}_2)\phi^{\mathbf{D}}_{\boldsymbol{\delta},L_d}(\mathbf{r}_2)d\mathbf{r}_1 d\mathbf{r}_2. \quad (1)$$

Each set $\phi^{\mathbf{A}}_{\boldsymbol{\alpha},L_a}$ of CGTOs is characterized by a set of nuclear coordinates $\mathbf{A} = A_x, A_y, A_z$, a set of primitive or contracted gaussian exponents $\boldsymbol{\alpha} = \alpha_1, \alpha_2, \ldots$ and a set of *xyz* monomials corresponding to a particular angular momentum quantum number L_a. Each Cartesian ERI integral block is thus labeled by four angular momentum quantum numbers corresponding to the four sets of CGTOs involved and its dimensionality is given as the product between the four CGTO set cardinalities (number of exponents times number of *xyz* monomials).

An unnormalized primitive CGTO is given by the following expression:

$$\phi^{\mathbf{A}}_{\alpha,\ell_a,m_a,n_a}(\mathbf{r}) = (x-A_x)^{\ell_a}(y-A_y)^{m_a}(z-A_z)^{n_a}e^{-\alpha(\mathbf{r}-\mathbf{A})^2} \quad (2)$$

Hence each ERI in the block in (1) is characterized by four sets of nuclear coordinates, four exponents, and four sets of $x^\ell y^m z^n$ monomial exponents $\ell+m+n=L$

$$\text{ERI} = \iint \phi^{\mathbf{A}}_{\alpha,\ell_a,m_a,n_a}(\mathbf{r}_1)\phi^{\mathbf{B}}_{\beta,\ell_b,m_b,n_b}(\mathbf{r}_1)\frac{1}{|\mathbf{r}_1-\mathbf{r}_2|}\phi^{\mathbf{C}}_{\gamma,\ell_c,m_c,n_c} \times (\mathbf{r}_2)\phi^{\mathbf{D}}_{\delta,\ell_d,m_d,n_d}(\mathbf{r}_2)d\mathbf{r}_1 d\mathbf{r}_2. \quad (3)$$

The first step toward any ERI evaluation is the introduction of the Laplace transform:

$$\frac{1}{|\mathbf{r}_1-\mathbf{r}_2|} = \frac{2}{\pi^{1/2}}\int_0^\infty e^{-(\mathbf{r}_1-\mathbf{r}_2)^2u^2}du, \quad (4)$$

which, when inserted into eq. (3) gives

$$\text{ERI} = \frac{2}{\pi^{1/2}}\int_0^\infty I^*_x(\ell_a,\ell_b,\ell_c,\ell_d,u)\cdot I^*_y(m_a,m_b,m_c,m_d,u) \cdot I^*_z(n_a,n_b,n_c,n_d,u)\ du, \quad (5)$$

or, dropping the reference to the individual ℓ, m, and n monomial exponents for simplification

$$\text{ERI} = \frac{2}{\pi^{1/2}}\int_0^\infty I^*_x(a,b,c,d,u)\cdot I^*_y(a,b,c,d,u)\cdot I^*_z(a,b,c,d,u)du, \quad (6)$$

with each of the Cartesian component unscaled (starred) 2D integrals given as (showing only the x-component):

$$I^*_x(a,b,c,d,u) = \iint \phi^{A_x}_{\alpha,\ell_a}(x_1)\phi^{B_x}_{\beta,\ell_b}(x_1)\phi^{C_x}_{\gamma,\ell_c}(x_2)\phi^{D_x}_{\delta,\ell_d}(x_2) e^{-u^2(x_1-x_2)^2}dx_1dx_2 \quad (7)$$

where

$$\phi^{A_x}_{\alpha,\ell_a}(x_1) = (x_1-A_x)^{\ell_a}e^{-\alpha(x_1-A_x)^2} \quad (8)$$

and similar expressions for the others. Using the well-known gaussian product theorem, where the product of the exponential parts of two GTOs centered on **A** and **B** can be written as a single exponential centered at a point **P** on the line joining **A** with **B**

$$e^{-\alpha(\mathbf{r}-\mathbf{A})^2}e^{-\beta(\mathbf{r}-\mathbf{B})^2} = \mathbf{E}_{AB}e^{-p(\mathbf{r}-\mathbf{P})^2}, \quad (9)$$

with the new exponent p and the center **P** being given as

$$p = \alpha+\beta, \quad (10)$$

$$\mathbf{P} = \frac{\alpha\mathbf{A}+\beta\mathbf{B}}{\alpha+\beta}, \quad (11)$$

and the exponential prefactor $\mathbf{E}_{AB}$ as

$$\mathbf{E}_{AB} = e^{-\frac{\alpha\beta}{\alpha+\beta}\cdot(\mathbf{A}-\mathbf{B})^2}, \quad (12)$$

we can rewrite the expression for the $I^*_x(a,b,c,d,u)$ integral in eq. (7) as

$$E^x_{AB}E^x_{CD}\iint (x_1-A_x)^a(x_1-B_x)^b(x_2-C_x)^c(x_2-D_x)^d e^{-p(x_1-P_x)^2} e^{-q(x_2-Q_x)^2}e^{-u^2(x_1-x_2)^2}dx_1dx_2, \quad (13)$$

where q, **Q** and E^x_{CD} are the respective new exponent, combined center and x component of the exponential prefactor for the gaussian exponential product between the centers **C** and **D** given by analogous expressions as in eqs. (10), (11), and (12). Evaluation of the simplest 2D integral leads to:

$$I^*_x(0,0,0,0,u) = E^x_{AB}E^x_{CD}e^{\frac{-\rho u^2}{\rho+u^2}\cdot(P_x-Q_x)^2}\pi(\rho+u^2)^{-1/2}(p+q)^{-1/2}, \quad (14)$$

where ρ denotes the following exponent ratio

$$\rho = \frac{pq}{p+q}. \quad (15)$$

Introducing a new variable t,

$$t^2 = \frac{u^2}{\rho+u^2} \quad (16)$$

$$u^2 = \frac{t^2\rho}{1-t^2} \quad (17)$$

$$du = \sqrt{\rho}(1-t^2)^{-3/2}dt \quad (18)$$

we obtain

$$I^*_x(0,0,0,0,t) = E^x_{AB}E^x_{CD}e^{-\rho t^2\cdot(P_x-Q_x)^2}\pi(1-t^2)^{1/2} \times \rho^{-1/2}(p+q)^{-1/2}, \quad (19)$$

Defining now rescaled 2D integrals

$$I_x(a,b,c,d,t) = \frac{I^*_x(a,b,c,d,t)}{I^*_x(0,0,0,0,t)} \quad (20)$$

with similar expressions for the y and z component, we get a new expression of the ERI from eq. (6) in terms of integration over t alone, using also the relation between du and dt in eq. (18) and the fact that if the variable u ranges from 0 to ∞, then t ranges from 0 to 1:

$$\text{ERI} = \frac{2\pi^{5/2}}{pq\sqrt{p+q}}\mathbf{E}_{AB}\mathbf{E}_{CD}\int_0^1 e^{-\rho t^2\cdot(\mathbf{P}-\mathbf{Q})^2}I_x(a,b,c,d,t)\cdot I_y(a,b,c,d,t)\cdot I_z(a,b,c,d,t)dt. \quad (21)$$

This last equation for the ERIs is the basis for applying the Rys quadrature scheme. Any integral of the form

$$\int_0^1 P(t)e^{-Tt^2}dt = \sum_n P(t_n)w_n \tag{22}$$

involving a polynomial $P(t)$ in t can be evaluated exactly by an n-point numerical quadrature formula using n roots t_n and weights w_n whose values depend on the exponential parameter T. In the above ERI equation we identify T as:

$$T = \rho(\mathbf{P} - \mathbf{Q})^2. \tag{23}$$

The 2D integrals are polynomials in t, which can clearly be seen from the vertical and horizontal recursion relations (VRR and HRR, respectively), which we will now state. Differentiating the expression for the 2D integrals in eq. (13) with respect to the two variables x_1 and x_2 and regrouping terms we arrive at the following basic 2D integral relations:

$$\begin{aligned} I_x(a+1,b,c,d,t) = {} & C_{00x} \cdot I_x(a,b,c,d,t) \\ & + a \cdot B_{10} \cdot I_x(a-1,b,c,d,t) \\ & + b \cdot B_{10} \cdot I_x(a,b-1,c,d,t) \\ & + c \cdot B_{00} \cdot I_x(a,b,c-1,d,t) \\ & + d \cdot B_{00} \cdot I_x(a,b,c,d-1,t) \end{aligned} \tag{24}$$

$$\begin{aligned} I_x(a,b+1,c,d,t) = {} & C'_{00x} \cdot I_x(a,b,c,d,t) \\ & + a \cdot B_{10} \cdot I_x(a-1,b,c,d,t) \\ & + b \cdot B_{10} \cdot I_x(a,b-1,c,d,t) \\ & + c \cdot B_{00} \cdot I_x(a,b,c-1,d,t) \\ & + d \cdot B_{00} \cdot I_x(a,b,c,d-1,t) \end{aligned} \tag{25}$$

$$\begin{aligned} I_x(a,b,c+1,d,t) = {} & D_{00x} \cdot I_x(a,b,c,d,t) \\ & + a \cdot B_{00} \cdot I_x(a-1,b,c,d,t) \\ & + b \cdot B_{00} \cdot I_x(a,b-1,c,d,t) \\ & + c \cdot B_{01} \cdot I_x(a,b,c-1,d,t) \\ & + d \cdot B_{01} \cdot I_x(a,b,c,d-1,t) \end{aligned} \tag{26}$$

$$\begin{aligned} I_x(a,b,c,d+1,t) = {} & D'_{00x} \cdot I_x(a,b,c,d,t) \\ & + a \cdot B_{00} \cdot I_x(a-1,b,c,d,t) \\ & + b \cdot B_{00} \cdot I_x(a,b-1,c,d,t) \\ & + c \cdot B_{01} \cdot I_x(a,b,c-1,d,t) \\ & + d \cdot B_{01} \cdot I_x(a,b,c,d-1,t), \end{aligned} \tag{27}$$

where the coefficients are expressions in terms of t^2:

$$\begin{aligned} C_{00x} &= (P_x - A_x) - \frac{q(P_x - Q_x)}{p+q}t^2 \\ C'_{00x} &= (P_x - B_x) - \frac{q(P_x - Q_x)}{p+q}t^2 \\ D_{00x} &= (Q_x - C_x) - \frac{p(P_x - Q_x)}{p+q}t^2 \\ D'_{00x} &= (Q_x - D_x) - \frac{p(P_x - Q_x)}{p+q}t^2 \\ B_{00} &= \frac{1}{2(p+q)}t^2 \\ B_{10} &= \frac{1}{2p} - \frac{q}{2p(p+q)}t^2 \\ B_{01} &= \frac{1}{2q} - \frac{p}{2q(p+q)}t^2. \end{aligned} \tag{28}$$

The VRR equations as stated in ref. 4 are obtained from eqs. (24) and (26) by setting $b, d = 0$ giving:

$$\begin{aligned} I_x(a+1,0,c,0,t) = {} & C_{00x} \cdot I_x(a,0,c,0,t) \\ & + a \cdot B_{10} \cdot I_x(a-1,0,c,0,t) \\ & + c \cdot B_{00} \cdot I_x(a,0,c-1,0,t) \end{aligned} \tag{29}$$

$$\begin{aligned} I_x(a,0,c+1,0,t) = {} & D_{00x} \cdot I_x(a,0,c,0,t) \\ & + a \cdot B_{00} \cdot I_x(a-1,0,c,0,t) \\ & + c \cdot B_{01} \cdot I_x(a,0,c-1,0,t). \end{aligned} \tag{30}$$

The HRR equations we get by subtracting eq. (24) from eq. (25) and eq. (26) from eq. (27):

$$I_x(a,b+1,c,d,t) = I_x(a+1,b,c,d,t) + (A_x - B_x) \cdot I_x(a,b,c,d,t) \tag{31}$$

$$I_x(a,b,c,d+1,t) = I_x(a,b,c+1,d,t) + (C_x - D_x) \cdot I_x(a,b,c,d,t). \tag{32}$$

The starting value for applying the VRR equations is the 2D integral $I_x(0,0,0,0,t)$, which due to its definition in eq. (20) is equal to 1. A very important difference between the VRR and the HRR is that in the former the coefficients are dependent on the individual gaussian exponents through the variables p and q, whereas in the HRR only center coordinate differences appear. This has lead to applications of the HRR equations after all the exponent contraction steps have been performed, resulting in considerable time savings during the integral evaluation. To apply the HRR at contracted level one needs to evaluate via the VRR all x, y, z-component 2D integrals $I(e,f,t_n) = I(e,0,f,0,t_n)$ for all the n quadrature roots t_n with $e = \max(a,b), \ldots, a+b$ and $f = \max(c,d), \ldots, c+d$ being the

angular momentum ranges needed for both electrons. The necessary primitive ERIs are then calculated using the n-point quadrature formula from eq. (22):

$$\mathrm{ERI} = \frac{2\pi^{5/2}}{pq\sqrt{p+q}}\mathbf{E}_{AB}\mathbf{E}_{CD}\sum_n I_x(e,f,t_n)\cdot I_y(e,f,t_n)\cdot I_z(e,f,t_n)\cdot w_n. \tag{33}$$

Since e and f represent angular momentum ranges, certain x and y monomial exponents will be common to different z monomial exponents. Hence certain I_xI_y products in eq. (33) are common to different I_z factors and can be reused, if one designs the assembly of the primitive ERIs such that the outer loops run over the x and y exponents while the inner one loops over all possible z exponents allowed by the angular momentum ranges e and f. This is the basis of the reduced multiplication scheme of the Rys quadrature.[10]

General n-th Order Electron Repulsion Integral Derivatives

Let us now focus on evaluation of general open-ended n-th order ERI derivatives. We could start using eq. (21), however, dependence of the prefactors and the gaussian exponent on center coordinates complicates matters. Better to use eq. (6). Again we show the derivation for the x-component only, the other two being analogous. For derivative on a center E we have

$$\frac{\partial}{\partial E_x}\mathrm{ERI} = \frac{2}{\pi^{1/2}}\int_0^\infty \frac{\partial I_x^*(a,b,c,d,u)}{\partial E_x}\cdot I_y^*(a,b,c,d,u)\cdot I_z^*(a,b,c,d,u)du. \tag{34}$$

From the 2D integral expression in eq. (13), the 2D integral derivative becomes

$$\begin{aligned}\frac{\partial I_x^*(a,b,c,d,u)}{\partial E_x} &= I_x^*(a,b,c,d,u)_E\\ &= \delta_{AE}\big[-aI_x^*(a-1,b,c,d,u)+2\alpha I_x^*(a+1,b,c,d,u)\big]\\ &\quad+\delta_{BE}\big[-bI_x^*(a,b-1,c,d,u)+2\beta I_x^*(a,b+1,c,d,u)\big]\\ &\quad+\delta_{CE}\big[-cI_x^*(a,b,c-1,d,u)+2\gamma I_x^*(a,b,c+1,d,u)\big]\\ &\quad+\delta_{DE}\big[-dI_x^*(a,b,c,d-1,u)+2\delta I_x^*(a,b,c,d+1,u)\big].\end{aligned} \tag{35}$$

Double differentiation leads to:

$$\begin{aligned}&\frac{\partial^2 I_x^*(a,b,c,d,u)}{\partial E_x\partial F_x}\\ &= \delta_{AF}\big[-aI_x^*(a-1,b,c,d,u)_E+2\alpha I_x^*(a+1,b,c,d,u)_E\big]\\ &\quad+\delta_{BF}\big[-bI_x^*(a,b-1,c,d,u)_E+2\beta I_x^*(a,b+1,c,d,u)_E\big]\\ &\quad+\delta_{CF}\big[-cI_x^*(a,b,c-1,d,u)_E+2\gamma I_x^*(a,b,c+1,d,u)_E\big]\\ &\quad+\delta_{DF}\big[-dI_x^*(a,b,c,d-1,u)_E+2\delta I_x^*(a,b,c,d+1,u)_E\big],\end{aligned} \tag{36}$$

which, after inserting eq. (35) becomes

$$\begin{aligned}\frac{\partial^2 I_x^*(a,b,c,d,u)}{\partial E_x\partial F_x} &= \text{linear function of the original } I_x^*\text{'s}\\ &= \mathcal{F}_x\big(I_x^*\big).\end{aligned} \tag{37}$$

Letting $\mathcal{F}_x$, $\mathcal{F}_y$, $\mathcal{F}_z$ stand for the linear functions in I_x^*, I_y^*, I_z^* obtained for any particular differentiation sequence $\mathcal{D}_x, \mathcal{D}_y, \mathcal{D}_z$ (any order, any center sequence) on the respective x, y, z-component 2D integrals we can thus write any differentiated ERI as

$$\mathcal{D}_x\mathcal{D}_y\mathcal{D}_z\mathrm{ERI} = \frac{2}{\pi^{1/2}}\int_0^\infty \mathcal{F}_x\big(I_x^*\big)\cdot\mathcal{F}_y\big(I_y^*\big)\cdot\mathcal{F}_z\big(I_z^*\big)du. \tag{38}$$

Since the $\mathcal{F}$ are linear in I^*, we can divide all I^* by the corresponding rescaling factors $I^*(0,0,0,0,t)$ as shown in eq. (20) and have, after changing variables from u to t

$$\mathcal{F}_x\big(I_x^*\big) = I_x^*(0,0,0,0,t)\cdot\mathcal{F}_x(I_x), \tag{39}$$

since $I_x^*(0,0,0,0,t)$ represents just a scaling number whose value is given in eq. (19). Hence the working equation for the derivative ERIs is similar to eq. (21)

$$\mathcal{D}_x\mathcal{D}_y\mathcal{D}_z\mathrm{ERI} = \frac{2\pi^{5/2}}{pq\sqrt{p+q}}\mathbf{E}_{AB}\mathbf{E}_{CD}\int_0^1 e^{-\rho t^2\cdot(\mathbf{P}-\mathbf{Q})^2}\times\mathcal{F}_x(I_x)\cdot\mathcal{F}_y(I_y)\cdot\mathcal{F}_z(I_z)dt \tag{40}$$

and is conveniently evaluated using the Rys quadrature technique:

$$\frac{2\pi^{5/2}}{pq\sqrt{p+q}}\mathbf{E}_{AB}\mathbf{E}_{CD}\sum_n \mathcal{F}_x[I_x(a,b,c,d,t_n)]\cdot\mathcal{F}_y[I_y(a,b,c,d,t_n)]\times\mathcal{F}_z[I_z(a,b,c,d,t_n)]\cdot w_n. \tag{41}$$

Evaluation of the final derivative 2D integrals $\mathcal{F}(I)$ is done by first setting up the undifferentiated 2D integrals $I(a+\partial_a, b+\partial_b, c+\partial_c, d+\partial_d, t)$ with increased angular momenta by $\partial_a, \partial_b, \partial_c, \partial_d$ due to differentiations on the respective centers A, B, C, D using the VRR and HRR relations and applying the differentiation sequence $\mathcal{D}$ for each Cartesian component recursively. Note that the presence of derivative operators on centers A, B, C, D prevents the use of the HRR at contracted level. A partial application of the HRR at contraction level either on the bra or ket side of the ERI can still be done, however, in special situations in which the derivative operators operate only on the the bra or ket side.

General n-th Order Nuclear Attraction Integral Derivatives

While formulation of general n-th order derivatives of the one-electron overlap and kinetic energy integrals is straightforward, an attractive recursive formulation for the general n-th order derivatives of the one-electron nuclear attraction integrals (NAI) is somewhat

more complicated due to the presence of the nuclear center(s). Crucial in this case is the order in which the partial derivatives have to be applied to the basic 1D integrals.

Using the same notation as in eq. (3), a simple primitive Cartesian three center NAI integral between two CGTOs is defined as

$$\mathrm{NAI} = \int \phi^{\mathbf{A}}_{\alpha,\ell_a,m_a,n_a}(\mathbf{r}) \frac{-Z_C}{|\mathbf{r}-\mathbf{C}|} \phi^{\mathbf{B}}_{\beta,\ell_b,m_b,n_b}(\mathbf{r}) d\mathbf{r}, \tag{42}$$

where Z_C is the nuclear charge at center C. Using the Laplace transform

$$\frac{1}{|\mathbf{r}-\mathbf{C}|} = \frac{2}{\pi^{1/2}} \int_0^\infty e^{-(\mathbf{r}-\mathbf{C})^2 u^2} du, \tag{43}$$

we get in analogy to eq. (6)

$$\mathrm{NAI} = -Z_C \frac{2}{\pi^{1/2}} \int_0^\infty I^*_x(a,b,C,u) \cdot I^*_y(a,b,C,u) \cdot I^*_z(a,b,C,u) du, \tag{44}$$

with each of the unscaled 1D integrals given as (only x-component shown):

$$I^*_x(a,b,C,u) = \int (x-A_x)^a (x-B_x)^b e^{-\alpha(x-A_x)^2} e^{-\beta(x-B_x)^2} \times e^{-u^2(x-C_x)^2} dx. \tag{45}$$

The gaussian product theorem from eq. (9) transforms this into

$$I^*_x(a,b,C,u) = E^x_{AB} \int (x-A_x)^a (x-B_x)^b e^{-p(x-P_x)^2} e^{-u^2(x-C_x)^2} dx. \tag{46}$$

The value of the simplest 1D integral is

$$I^*_x(0,0,C,u) = E^x_{AB} e^{\frac{-pu^2}{p+u^2}\cdot(P_x-C_x)^2} \sqrt{\frac{\pi}{p+u^2}}, \tag{47}$$

which after introduction of the same variable t as defined in eq. (16) leads to

$$I^*_x(0,0,C,t) = E^x_{AB} e^{-pt^2\cdot(P_x-C_x)^2} \sqrt{\frac{\pi(1-t^2)}{p}}. \tag{48}$$

Defining the rescaled 1D integrals

$$I_x(a,b,C,t) = \frac{I^*_x(a,b,C,t)}{I^*_x(0,0,C,t)} \tag{49}$$

we can reformulate eq. (44) as

$$\mathrm{NAI} = -Z_C \frac{2\pi}{p} \mathbf{E}_{AB} \int_0^1 e^{-pt^2\cdot(\mathbf{P}-\mathbf{C})^2} I_x(a,b,C,t) \cdot I_y(a,b,C,t) \cdot I_z(a,b,C,t) dt. \tag{50}$$

The exponential parameter T is thus

$$T = p(\mathbf{P}-\mathbf{C})^2. \tag{51}$$

The VRR relations for the 1D integrals are[4]

$$I_x(a+1,b,C,t) = R_{1x} \cdot I_x(a,b,C,t) + a \cdot R_2 \cdot I_x(a-1,b,C,t) + b \cdot R_2 \cdot I_x(a,b-1,C,t) \tag{52}$$

$$I_x(a,b+1,C,t) = R'_{1x} \cdot I_x(a,b,C,t) + a \cdot R_2 \cdot I_x(a-1,b,C,t) + b \cdot R_2 \cdot I_x(a,b-1,C,t) \tag{53}$$

with VRR coefficients

$$\begin{aligned} R_{1x} &= (P_x - A_x) - (P_x - C_x)t^2 \\ R'_{1x} &= (P_x - B_x) - (P_x - C_x)t^2 \\ R_2 &= \frac{1-t^2}{2p}. \end{aligned} \tag{54}$$

The corresponding HRR formula is

$$I_x(a,b+1,C,t) = I_x(a+1,b,C,t) + (A_x - B_x) \cdot I_x(a,b,C,t). \tag{55}$$

Let us now apply a center differentiation operator on a NAI integral. We get from eq. (44)

$$\frac{\partial}{\partial E_x}\mathrm{NAI} = -Z_C \frac{2}{\pi^{1/2}} \int_0^\infty \frac{\partial I^*_x(a,b,C,u)}{\partial E_x} \cdot I^*_y(a,b,C,u) \cdot I^*_z(a,b,C,u) du. \tag{56}$$

From the definition of the 1D integral in eq. (45) its derivative is

$$\begin{aligned} \frac{\partial I^*_x(a,b,C,u)}{\partial E_x} &= I^*_x(a,b,C,u)_E \\ &= \delta_{AE}\big[-aI^*_x(a-1,b,C,u) + 2\alpha I^*_x(a+1,b,C,u)\big] \\ &\quad + \delta_{BE}\big[-bI^*_x(a,b-1,C,u) + 2\beta I^*_x(a,b+1,C,u)\big] \\ &\quad + \delta_{CE}\frac{2pt^2}{1-t^2}\big[I^*_x(a+1,b,C,u) + (A_x - C_x)I^*_x(a,b,C,u)\big], \end{aligned} \tag{57}$$

where we have used the expression for variable u in eq. (17) and the identity $x - C_x = x - A_x + A_x - C_x$. The last term can be modified by expanding the $I_x^*(a+1,b,C,u)$ integral in terms of lower angular momentum integrals using the VRR in eq. (52) and inserting the values of the VRR coefficients to give

$$\begin{aligned} &I_x^*(a,b,C,u)_E \\ &= \delta_{AE}\big[-aI_x^*(a-1,b,C,u) + 2\alpha I_x^*(a+1,b,C,u)\big] \\ &\quad + \delta_{BE}\big[-bI_x^*(a,b-1,C,u) + 2\beta I_x^*(a,b+1,C,u)\big] \\ &\quad + \delta_{CE}t^2\big[2p(P_x - C_x)I_x^*(a,b,C,u) + aI_x^*(a-1,b,C,u) \\ &\quad + bI_x^*(a,b-1,C,u)\big]. \end{aligned} \tag{58}$$

This last equation is much more convenient, as it does not involve a larger angular momentum on either sites a or b due to possible derivatives on the nuclear attraction center. However, the presence of the term $(P_x - C_x)$ on the r.h.s. of the equation leads to extra terms when higher derivatives are involved. Double differentiation gives:

$$\begin{aligned} &\frac{\partial}{\partial F_x}\left(\frac{\partial I_x^*(a,b,C,u)}{\partial E_x}\right) \\ &= \frac{\partial}{\partial E_x}\left(\frac{\partial I_x^*(a,b,C,u)}{\partial F_x}\right) \\ &= \delta_{AF}\big[-aI_x^*(a-1,b,C,u)_E + 2\alpha I_x^*(a+1,b,C,u)_E\big] \\ &\quad + \delta_{BF}\big[-bI_x^*(a,b-1,C,u)_E + 2\beta I_x^*(a,b+1,C,u)_E\big] \\ &\quad + \delta_{CF}t^2\big[2p(P_x - C_x)I_x^*(a,b,C,u)_E + aI_x^*(a-1,b,C,u)_E \\ &\quad + bI_x^*(a,b-1,C,u)_E\big] \\ &\quad + \delta_{CF}t^2(2\alpha\delta_{AE} + 2\beta\delta_{BE} - 2p\delta_{CE})I_x^*(a,b,C,u), \end{aligned} \tag{59}$$

showing that the second derivative does not only depend on the first derivative but also on the original undifferentiated 1D integral if differentiation on the nuclear center is involved. To avoid proliferation of terms it is thus mandatory to perform the differentiation of each center separately to the desired order. Applying the last equation several times we obtain two differentiation schemes, one for the nuclear center

$$\begin{aligned} \frac{\partial^m}{\partial C_x^m}I_x^*(a,b,C,u)_E &= at^2\frac{\partial^{m-1}}{\partial C_x^{m-1}}I_x^*(a-1,b,C,u)_E \\ &\quad + bt^2\frac{\partial^{m-1}}{\partial C_x^{m-1}}I_x^*(a,b-1,C,u)_E \\ &\quad + 2pt^2\left[(P_x - C_x)\frac{\partial^{m-1}}{\partial C_x^{m-1}}I_x^*(a,b,C,u)_E\right. \\ &\quad \left. + (1-m)\frac{\partial^{m-2}}{\partial C_x^{m-2}}I_x^*(a,b,C,u)_E\right], \end{aligned} \tag{60}$$

with subscript E denoting any past differentiation steps not involving the nuclear center, and the other for the gaussian centers

$$\begin{aligned} \frac{\partial^m}{\partial F_x^m}I_x^*(a,b,C,u)_E &= \delta_{AF}\left[-a\frac{\partial^{m-1}}{\partial F_x^{m-1}}I_x^*(a-1,b,C,u)_E\right. \\ &\quad \left. + 2\alpha\frac{\partial^{m-1}}{\partial F_x^{m-1}}I_x^*(a+1,b,C,u)_E\right] \\ &\quad + \delta_{BF}\left[-b\frac{\partial^{m-1}}{\partial F_x^{m-1}}I_x^*(a,b-1,C,u)_E\right. \\ &\quad \left. + 2\beta\frac{\partial^{m-1}}{\partial F_x^{m-1}}I_x^*(a,b+1,C,u)_E\right], \end{aligned} \tag{61}$$

where F can be any one of the two gaussian centers A and B and E symbolizes now any past differentiation steps including those involving the nuclear center. Note in particular that each nuclear differentiation step in eq. (60) increases the final differentiated 1D integral polynomial by t^2, a fact that has to be taken into consideration when evaluating the number of necessary quadrature roots. A special situation arises if the nuclear attraction center coincides with one of the gaussian centers. In these cases eq. (60) cannot be applied. Instead one can reformulate the derivative sequence in such a way that only differentiation on the gaussian center different from the nuclear attraction center in involved. We have the following 1D integral identities

$$\frac{d}{dA_x}I_x^*(a,b,A,u) = -\frac{d}{dB_x}I_x^*(a,b,A,u) \tag{62}$$

$$\frac{d}{dB_x}I_x^*(a,b,B,u) = -\frac{d}{dA_x}I_x^*(a,b,B,u) \tag{63}$$

and thus

$$\begin{aligned} \frac{d^m}{dA_x^m}\frac{d^n}{dB_x^n}I_x^*(a,b,A,u) &= (-1)^m\frac{d^{m+n}}{dB_x^{m+n}}I_x^*(a,b,A,u) \\ \frac{d^m}{dA_x^m}\frac{d^n}{dB_x^n}I_x^*(a,b,B,u) &= (-1)^n\frac{d^{m+n}}{dA_x^{m+n}}I_x^*(a,b,B,u), \end{aligned} \tag{64}$$

which we can evaluate using eq. (61). Following the same logic as for the general n-th order ERI derivative, we have for the general n-th order NAI derivative

$$\mathcal{D}_x\mathcal{D}_y\mathcal{D}_z\mathrm{NAI} = -Z_C\frac{2}{\pi^{1/2}}\int_0^\infty \mathcal{F}_x\big(I_x^*\big)\cdot\mathcal{F}_y\big(I_y^*\big)\cdot\mathcal{F}_z\big(I_z^*\big)du, \tag{65}$$

where the $\mathcal{D}$ symbolize any differentiation sequence. Using again the linearity of $\mathcal{F}$ in terms of the I^*'s we have

$$\mathcal{F}_x\big(I_x^*\big) = I_x^*(0,0,C,t)\cdot\mathcal{F}_x(I_x) \tag{66}$$

and eq. (65) transforms to

$$\mathcal{D}_x\mathcal{D}_y\mathcal{D}_z\mathrm{NAI} = \frac{2\pi}{p}\mathbf{E}_{AB}\int_0^1 e^{pt^2\cdot(\mathbf{P}-\mathbf{C})^2}\mathcal{F}_x(I_x)\cdot\mathcal{F}_y(I_y)\cdot\mathcal{F}_z(I_z)dt. \tag{67}$$

Rys Quadrature

In this section we state briefly the gaussian quadrature procedure used for the Rys weight function. The exposition is rather sketchy, for details the reader is referred to the cited references. The general gaussian quadrature theory states that, given a definite integral with integrand of the form polynomial $P(x)$ times a weight function $W(x)$, its exact value can be found by summing up products of certain weights w_n and polynomial values at roots x_n

$$\int_a^b P(x)W(x)dx = \sum_{n=1}^{N} P(x_n)w_n \tag{68}$$

for all polynomials of order up to $2N-1$. The roots and weights can be found by constructing a set of monic (coefficient of largest power of x equals 1) polynomials p_i, which are orthogonal over the weight function $W(x)$

$$\langle p_i|p_j\rangle = \int_a^b W(x)p_i(x)p_j(x)\,dx = 0, \quad i \neq j \tag{69}$$

Because of this orthogonality property, these polynomials obey a 3-term recurrence relation

$$\begin{aligned} p_0(x) &= 1 \\ p_1(x) &= (x-a_0) \\ p_i(x) &= (x-a_{i-1})p_{i-1}(x) - b_{i-1}p_{i-2}(x), \quad i>1 \end{aligned} \tag{70}$$

where the recursion coefficients are, using the notation in eq. (69)

$$\begin{aligned} a_i &= \frac{\langle xp_i|p_i\rangle}{\langle p_i|p_i\rangle} \quad i=0,1,\ldots \\ b_i &= \frac{\langle p_i|p_i\rangle}{\langle p_{i-1}|p_{i-1}\rangle} \quad i=1,2,\ldots \end{aligned} \tag{71}$$

Setting up the $N\times N$ tridiagonal matrix

$$\mathbf{T} = \begin{bmatrix} a_0 & \sqrt{b_1} & & & \\ \sqrt{b_1} & a_1 & \sqrt{b_2} & & \\ & \cdot & \cdot & & \\ & \cdot & \cdot & & \\ & & \sqrt{b_{N-2}} & a_{N-2} & \sqrt{b_{N-1}} \\ & & & \sqrt{b_{N-1}} & a_{N-1} \end{bmatrix} \tag{72}$$

the roots can now be obtained[18,19] as the eigenvalues of $\mathbf{T}$ and the associated weights are evaluated using the first component c_{1n} of each eigenvector

$$w_n = c_{1n}^2\int_a^b W(x)dx. \tag{73}$$

Hence for obtaining the roots and weights all that is needed is the values of the recurrence coefficients from eq. (71) and the value of the integral on the r.h.s. of eq. (73). For some weights and specific integration limits, the associated set of orthogonal polynomials and the values of the recurrence coefficients are very well known.[20] These are the so called classical cases. The situation is different if one encounters a weight function different from the classical weights and/or the integration limits differ from the classical cases. Here we need to evaluate first the a's and b's defining the orthogonal polynomials. A straightforward approach would be to expand each polynomial $p_i(x)$ in terms of all powers of x and to evaluate the recurrence coefficients in eq. (71) using the $2N$ moments of the weight function

$$\int_a^b x^i W(x)dx, \quad i=0,1,\ldots,2N-1 \tag{74}$$

This approach turns out to be extremely unstable numerically. To get a numerically stable procedure,[21] it is best to represent the polynomials $p_i(x)$ in terms of a set of auxiliary polynomials $\pi_i(x)$, obeying a 3-term recurrence relation with known coefficients α and β

$$\begin{aligned} \pi_0(x) &= 1 \\ \pi_1(x) &= (x-\alpha_0) \\ \pi_i(x) &= (x-\alpha_{i-1})\pi_{i-1}(x) - \beta_{i-1}\pi_{i-2}(x), \quad i>1. \end{aligned} \tag{75}$$

Then, using the $2N-1$ modified moment values involving the weight $W(x)$

$$\mu_i = \int_a^b \pi_i(x)W(x)dx, \quad i=0,1,\ldots,2N-1 \tag{76}$$

a very elegant algorithm due to Wheeler[22] can be used to evaluate the a's and b's in terms of the α's and β's and the modified moments μ_i. Details of Wheeler's algorithm can be found in refs. 23 and 24. Important to note is that the algorithm is numerically very stable if the set of polynomials $\pi_i(x)$ is such that they represent a set of orthogonal polynomials for some weight function resembling the weight $W(x)$ in the integration range of interest. Note that the integration range for the definition of the orthogonal $\pi_i(x)$ can be different from ours.

From the discussions in the previous section, we know that both the ERI and NAI integrals as well as their general n-th order derivatives can all be expressed as an integral of the form

$$\int_0^1 P(t^2)e^{-Tt^2}dt. \tag{77}$$

Performing the variable substitution $x = t^2$ this integral is equivalent to

$$\int_0^1 P(x)\frac{e^{-Tx}}{2\sqrt{x}}dx. \tag{78}$$

from which we identify the Rys weight function as $W_{Rys}(x) = e^{-Tx}x^{-1/2}$. This weight function is the same as one of the classical generalized T-scaled Laguerre weights $W^{\alpha}_{\text{Laguerre}}(x) = e^{-Tx}x^{\alpha}$ with associated set of orthogonal generalized T-scaled Laguerre polynomials $L_i^{\alpha}(Tx)$ defined over the integration range $x \in [0,\infty]$ and exponent range $\alpha > -1$:

$$\int_0^{\infty} W^{\alpha}_{\text{Laguerre}}(x)L_i^{\alpha}(Tx)L_j^{\alpha}(Tx)dx = 0, \quad i \neq j. \tag{79}$$

Clearly $W_{\text{Rys}}(x) = W^{-1/2}_{\text{Laguerre}}(x)$, however the nonmatching integration limits $[0,1]$ for $W_{\text{Rys}}(x)$ and $[0,\infty]$ for $W^{-1/2}_{\text{Laguerre}}(x)$ prevents the direct use of the generalized T-scaled Laguerre recursion coefficients for setting up the tridiagonal matrix in eq. (72). The monic generalized T-scaled Laguerre polynomials obey the following 3-term recursion relation:

$$\begin{aligned}
L_0^{\alpha}(Tx) &= 1\\
L_1^{\alpha}(Tx) &= \left(x - \frac{1+\alpha}{T}\right)\\
L_i^{\alpha}(Tx) &= \left(x - \frac{2i+\alpha-1}{T}\right)L_{i-1}^{\alpha}(Tx)\\
&\quad - \frac{(i-1)(i+\alpha-1)}{T^2}L_{i-2}^{\alpha}(Tx), \quad i > 1.
\end{aligned} \tag{80}$$

and the set $L_i^{-1/2}(Tx)$ with $\alpha = -1/2$ will constitute our set of auxiliary polynomials in eq. (75) with the 3-term recursion coefficients obtained from eq. (80):

$$\alpha_0 = \frac{1}{2T} \quad \alpha_{i-1} = \frac{4i-3}{2T} \quad \beta_{i-1} = \frac{(i-1)(2i-3)}{2T^2} \qquad i = 2,3,\ldots,2N-1 \tag{81}$$

The needed modified moments from eq. (74) can be given in terms of values of the $\alpha = +1/2$ monic generalized T-scaled Laguerre polynomials at the scaling factor T

$$\begin{aligned}
\mu_0 &= \int_0^1 \frac{e^{-Tx}}{2\sqrt{x}}dx = F_0(T)\\
\mu_i &= \int_0^1 L_i^{-1/2}(Tx)\frac{e^{-Tx}}{2\sqrt{x}}dx = -\frac{e^{-T}}{2T}L_{i-1}^{+1/2}(T)dx,\\
&\quad i = 1,2,\ldots,2N-1.
\end{aligned} \tag{82}$$

Note that all $L_{i-1}^{+1/2}(T)$ values and thus all modified moments μ_i can be very easily obtained from the general recursion in (80) by setting $\alpha = +1/2$ and $x = 1$. We will refer to the use of the Laguerre polynomials as the Laguerre case.

As T gets progressively smaller the use of the Laguerre polynomials becomes problematic. For small T we have to use a different set of auxiliary polynomials. A very convenient set consists of the shifted Jacobi polynomials $G(p,q,x)$ with $p = q = 1/2$. These polynomials obey the following 3-term recursion relation:

$$\begin{aligned}
G_0\left(\frac{1}{2},\frac{1}{2},x\right) &= 1\\
G_1\left(\frac{1}{2},\frac{1}{2},x\right) &= (x-\alpha_0)\\
G_i\left(\frac{1}{2},\frac{1}{2},x\right) &= (x-\alpha_{i-1})G_{i-1}\left(\frac{1}{2},\frac{1}{2},x\right)\\
&\quad - \beta_{i-1}G_{i-2}\left(\frac{1}{2},\frac{1}{2},x\right), \quad i > 1,
\end{aligned} \tag{83}$$

with recursion coefficients:

$$\begin{aligned}
\alpha_i &= \frac{2i(i+1/2)-1/4}{(2i+1/2)^2-1}\\
\beta_i &= \frac{i^2(i-1/2)^2}{(2i-1/2)^2(2i-3/2)(2i+1/2)}.
\end{aligned} \tag{84}$$

It can be shown that for the modified moments

$$\mu_i = \int_0^1 G_i\left(\frac{1}{2},\frac{1}{2},x\right)\frac{e^{-Tx}}{2\sqrt{x}}dx, \quad i = 0,1,\ldots,2N-1 \tag{85}$$

a 3-term recursion relation can be derived

$$\mu_{i+1} = R(i,T)\mu_i + S(i)\mu_{i-1}, \tag{86}$$

where

$$\begin{aligned}
R(i,T) &= \frac{2i+1}{2T} + \frac{(2i+1)}{(4i-1)(4i+3)}\\
S(i) &= \frac{2i(2i+1)(2i-1)^2}{(4i-3)(4i+1)(4i-1)^2}.
\end{aligned} \tag{87}$$

Upward evaluation of the moments (increasing i) turns out to be very unstable numerically, hence Miller's algorithm[25] must be applied, especially since the minimal solution of eq. (86) is the one wanted. Miller's algorithm proceeds by downward recursion starting from a tiny seed value for a large i moment to the zeroth moment and rescaling the obtained sequence of moments such that $\mu_0 = 1$. The use of the shifted Jacobi polynomials will be referred to as the Jacobi case.

Table 1. Absolute Root Differences (Δ Root) Between the Quadruple Precision and Double Precision Values of the Roots for the Jacobi and Laguerre Quadratures for Several Values of T in the Range $10 \le T \le 30$.

T	Δ Root	# of Roots
Jacobi quadrature		
10	5×10^{-14}	6
15	9×10^{-12}	7
20	7×10^{-10}	10
25	2×10^{-7}	8
30	1×10^{-5}	9
Laguerre quadrature		
15	2×10^{-11}	10
	4×10^{-8}	12
	2×10^{-2}	14
20	6×10^{-13}	10
	1×10^{-9}	12
	9×10^{-7}	14
25	4×10^{-15}	10
	5×10^{-13}	12
	1×10^{-9}	14
	1×10^{-5}	16
30	3×10^{-16}	10
	1×10^{-14}	12
	2×10^{-11}	14
	3×10^{-8}	16
	8×10^{-5}	18

Numerical Considerations on the Rys Quadrature

Both Laguerre and Jacobi Rys quadratures exhibit different numerical properties when implemented on a computer. Both are found to be very stable in quadruple precision up to a number of roots $N \le 30$. The main question that remains is how stable are they in double precision. Using a quadruple precision version of the code as a standard of reference, Table 1 of absolute root differences between the quadruple precision and double precision values of the roots has been established for the most problematic range $10 \le T \le 30$:

The numerical behavior of both quadratures is quite different. The Jacobi quadrature proceeds in a stable fashion beyond any number of roots, exhibiting its largest root error for the above number of root cases in the third column. In other words, for the $T = 20$ case, the largest root error of 7×10^{-10} occurs when only 10 roots are evaluated and the error progressively decreases as the number of roots requested grows. On the other hand the Laguerre quadrature root errors are upper bounds in the number of roots sense. For example for $T = 20$, the largest error for 12 requested roots is 1×10^{-9} and will get progressively larger as the number of requested roots increases, ultimately resulting in a complete numerical breakdown of the algorithm. For 14 roots the optimum division line between the two quadrature cases occurs at $T = 23$ with maximum root error of about 1×10^{-8}. For 12 roots we have $T = 20$ with maximum root error of about 1×10^{-9}. The absolute weight error is always less than the corresponding root error and they run in opposite directions, i.e., the largest root error occurs around the smallest weight values, a fortunate effect as far as integral accuracy is concerned.

To generate the roots and weights on the fly for each T in an efficient way using the above algorithms, several points have to be observed. First, evaluation of the Jacobi recurrence coefficients for both the Jacobi polynomials and the modified moments using eqs. (84) and (87) is costly and is best performed as an include file with the first 100 terms precalculated and stored. Second, the complete sequence of algorithms (setting up the polynomial recursion coefficients, evaluation of the modified moments, calculating the Rys polynomial recurrence coefficients, setting up the tridiagonal matrix and solving for the roots and weights) are placed in only one routine without calling subroutines and all possible T's must be handled in one loop inside the routine. Five vectors of size $2N - 1$ and 2 vectors of size N are required as working space, making it very cache efficient to evaluate all the roots and weights at once. The number of roots and weights N is never very large, hence all the working arrays stay most of the time in fast cache. Note that Wheeler's algorithm, although presented in the literature in terms of a two-dimensional $(2N - 1) \times (2N - 1)$ array,[24] can actually be formulated using just 2 vectors of size $2N - 1$.

Integral Contraction Procedure

The general ERI integral contraction is a 4-index transformation procedure

$$(rs|tu) = \sum_{ijk\ell} C_{ir} C_{js} C_{kt} C_{\ell u} [ij|k\ell], \tag{88}$$

in which primitive ERI integrals $[ij|k\ell]$ are transformed to contracted ERI integrals $(rs|tu)$ using the contraction coefficient matrices $\mathbf{C}$. A straightforward implementation of this equation would consist in splitting the overall summation into four partial summations, reusing intermediatly transformed integrals as much as possible. From a computational performance view, however, this strategy is not enough, especially for large contracted basis sets. First, there is usually some structure associated with the contraction columns in $\mathbf{C}$, which can range from having only one or very few elements different from zero (the so called segmented contraction case) to having all elements equal to some finite number (the fully contracted case). Furthermore, the segmented case presents itself usually with all nonzero elements clustered together (shown as stars ⋆ below) and thus we have pictorially the following structure for a $\mathbf{C}$ column corresponding to the r-th contraction

$$\mathbf{C}_r = \left| \begin{array}{c|l} \star & \longleftarrow r_{\text{begin}} \\ \star & \\ \star & \\ \star & \\ \star & \longleftarrow r_{\text{end}} \end{array} \right| , \tag{89}$$

where r_{begin} and r_{end} denote, respectively, the first and last nonzero element delimiters. Different columns of C will generally have different nonzero delimiters, which are passed to the contraction routine and allow skipping over unnecessary zero multiplications.

Note that this scheme handles the uncontracted cases equally well, which can easily be identified by checking if $r_{\text{begin}} = r_{\text{end}}$, in which case no contraction loop is ever entered for that particular column.

The second, and more important, observation relates to the number of cache faults when applying the four partial summations sequentially one after the other.[26] The key idea is to introduce a small cache-fitting auxillary array containing intermediately transformed integrals which are reused as much as possible before being thrown out from the cache. To illustrate this issue, consider the ket side half contraction transformation in eq. (88) involving the contracted t, u and the primitive k, ℓ indices

$$[n|ku] = \sum_{\ell} C_{\ell u}[n|k\ell], \tag{90}$$

$$[n|tu) = \sum_{k} C_{kt}[n|ku], \tag{91}$$

where n denotes all those indices not taking part in the contraction (note that n can be quite large, representing not only the bra primitive or contraction indices but also the complete set of all cartesian monomial quartuples associated with the integrals). The overall number of quarterly transformed integrals $[n|ku]$ is often much larger than the cache size. Hence if step (91) is applied after the first quarter transformation (90) is completed over the entire n range, multiple cache misses will result with degradation of performance. A much better way would be to do the following:

$$\begin{array}{l}\text{Loop over } \bar{n} \text{ ranges}\\ \quad\text{Loop over } k\\ \qquad \{\bar{n}|k\} = \sum_{\ell} C_{\ell u}\ [\bar{n}|k\ell]\\ \quad\text{End loop}\\ \quad [\bar{n}|tu) = \sum_{k} C_{kt}\ \{\bar{n}|k\}\\ \text{End loop}\end{array} \tag{92}$$

where now the size of the $\bar{n}$ range is selected such that the entire intermediate array $\{\bar{n}|k\}$ fits into the cache together with the needed contraction coefficients and the $\bar{n}$-sections of the primitive and half transformed integrals. Of course the cache size available for use at contraction time must be approximately known or at least estimated, taking into account the possibility of simultaneous cache usage by other CPU processes.

Further tweaking of contraction performance can be achieved by realizing that in those cases where the bra and ket indices correspond to the same atomic center and the same contracted shell, only those integrals need to be explicitly contracted which belong to the lower triangles of the corresponding bra and ket index pairs. Also the order of contraction within a bra or ket pair is important. Looking at the above example in (92), if the number of k-contractions is larger than the ℓ-contractions, the k and ℓ loops would be reversed, thus demanding less space for the intermediate array which in turn allows for a larger $\bar{n}$ range. In our code every possible contraction scenario is handled by a separate piece of code inside one routine, thus avoiding if-statements in the innermost compute intensive contraction loops. Overall our experience with the contraction routines, implemented as outlined above, leads us to conclude that even for fully contracted basis sets the contraction steps are not the most compute intensive. In CPU time it ranks below the 2D to 6D cartesian integral assembly routine, the latter being an inherently cache-nonfriendly procedure requiring non-unit stride access to relatively large arrays.

Cartesian to Spherical Transformation and HRR at Contracted Level

Once integrals over Cartesian GTOs have been calculated, one needs to transform them to integrals over spherical GTOs. The monomial basis $\{xyz\}$ of angular momentum L is thereby transformed into the spherical basis $\{rY\}$ consisting of a radial part r and a spherical harmonic part Y labeled by magnetic quantum numbers $M = -L, -L+1, \ldots, L-1, L$. In matrix form, we write this as

$$\mathbf{rY} = \mathbf{xyz} \cdot \mathbf{T}_{\text{sph}}. \tag{93}$$

The Cartesian $\rightarrow$ spherical transformation matrix $\mathbf{T}_{\text{sph}}$ is $|xyz| \times |rY|$ dimensional and its individual entries can be obtained from the $\{rY\}$ expansion formula below in terms of the monomial basis $\{xyz\}$:

$$rY_M^L = \frac{|M|!S}{2^L\left(2L - 2\left[\frac{L-|M|}{2}\right] - 1\right)!!} x^{2\ell-2i+|M|-\delta} y^{2k-2\ell+2i+\delta} z^{L-|M|-2k}$$

$$S = \sum_{i=0}^{\left[\frac{|M|-\delta}{2}\right]} \sum_{j=0}^{\left[\frac{L-|M|}{2}\right]} \sum_{k=0}^{j} \sum_{\ell=0}^{k} \frac{(-1)^{i+j}(2L-2j)!}{(2i+\delta)!(|M|-2i-\delta)!(L-j)!(L-|M|-2j)!(j-k)!\ell!(k-\ell)!}, \tag{94}$$

where δ equals 1 for $M < 0$ and 0 for $M \geq 0$ and [] denotes the integer part. Although (94) looks formidable it is actually quite easy to implement, the only important thing to remember here is not to predetermine and use factorials and double factorials, but rather to evaluate the monomial prefactors as a series of multiplications of appropriate fractions. In this way, overflow of integers and loss of accuracy of the reals is avoided and Cartesian to spherical transformations of very large angular momentum functions ($L > 20$) are possible. In our integral package, the transformation turns out to require almost negligible time and is always performed on the fly.

Table 2. % of Non-Zero Elements in $\mathbf{T}_{\text{hrr}}$ and $\mathbf{T}_{\text{sph}}$ Matrices for Equal Angular Momenta $a = b$.

a	$\mathbf{T}_{\text{hrr}}$	$\mathbf{T}_{\text{sph}}$
2	11.6	26.7
4	5.9	20.7
6	4.2	19.2
8	3.4	18.4
10	3.0	17.7

Let us now turn our attention to the HRR at the contracted level. Given two angular momentum quantum numbers a and b, the HRR at the contracted level represents a matrix $\mathbf{T}_{\text{hrr}}$ that brings integrals from one direct product angular momentum monomial basis $\{xyz\}_{e0} = \{xyz\}_e \otimes 1$ to another $\{xyz\}_{ab} = \{xyz\}_a \otimes \{xyz\}_b$, where e denotes the angular momentum range $e = a, a+1, \ldots, a+b$:

$$\mathbf{xyz}_{ab} = \mathbf{xyz}_{e0} \cdot \mathbf{T}_{\text{hrr}}. \tag{95}$$

The assembly of $\mathbf{T}_{\text{hrr}}$ can be broken down to several elementary HRR steps, in which each step decreases the e angular momentum range by 1 and increases b by 1:

$$\mathbf{xyz}_{e-1,b+1} = \mathbf{xyz}_{e,b} \cdot \mathbf{T}_{\text{step}}. \tag{96}$$

Clearly then

$$\mathbf{T}_{\text{hrr}} = \prod_{\text{steps}} \mathbf{T}_{\text{step}}. \tag{97}$$

From the HRR relation in eq. (31), we see that each HRR step combines only up to two different basis elements of $\mathbf{xyz}_{e,b}$ for each new basis element of $\mathbf{xyz}_{e-1,b+1}$. The resulting matrix $\mathbf{T}_{\text{step}}$ is thus very sparse having at most two entries for each column. But also the final $\mathbf{T}_{\text{hrr}}$ matrix is considerably sparse, which is shown in Table 2 for several values of $a = b$ together with the sparseness in % of non-zero elements of the corresponding spherical transformation matrices.

The degree of sparseness for $\mathbf{T}_{\text{hrr}}$ increases even further if one or more of the coordinate component differences in the HRR equations in (31) is equal to zero. Note that a in eq. (95) can by itself be an angular momentum range. Efficient coding of 3-center overlap integrals require for example two HRR procedures in which the total angular momentum $f = a + b + c$ is decomposed first to separate angular momentum c followed by separation of angular momentum b:

$$\begin{aligned} \mathbf{xyz}_{e0,c} &= \mathbf{xyz}_{f00} \cdot \mathbf{T}^{I}_{\text{hrr}} \\ \mathbf{xyz}_{abc} &= \mathbf{xyz}_{e0,c} \cdot \mathbf{T}^{II}_{\text{hrr}} \end{aligned} \tag{98}$$

At implementation level there are two possible approaches. The first one is to perform each spherical and each HRR transformation separately on a set of integrals. This allows for usage of the sparseness of both $\mathbf{T}_{\text{hrr}}$ and $\mathbf{T}_{\text{sph}}$. The second approach would be to first combine both types of matrices and to perform the integral transformation with a combined matrix. Unfortunately the combined matrix is no longer sparse, in fact only a small fraction of its elements are zero. For large angular momentum values storage of the large combined matrices becomes an issue. On the other hand the use of the combined matrices avoids twice a call to the spherical transformation routine with a large set of intermediate half-transformed integrals. Our decision for using the first approach with separate routines handling the spherical and HRR transformations stems mainly from simplicity: less code is needed to achieve the goal. This is also justified from the sharp profile output of the entire integral evaluation code: by far the largest portion of CPU time is spent in the cartesian 2D to 6D integral assembly routine. The spherical and HRR transformations require only negligible CPU time.

Design and Implementation Details

The design and implementation of the ACES III integral package was governed by the following requirements. Maximum priority was set on evaluation speed. This requires efficient cache use and minimization of subroutine calls (fat routines). Each main integral evaluation routine, i.e. those that will be called from ACES III, evaluates one block of integrals according to the definition in eq. (1). Hence each such routine has in argument only the basic GTO info: exponents, center coordinates, contraction coefficients, possibly differentiation orders for each center and flags for internal screening and Cartesian or spherical integrals for output. No additional info is passed, like for example density matrix elements, to perform manipulations for screening judgements inside the integral code. Such handling of density elements and screening decisions based on their values is better left to the ACES III level outside the integral evaluation. Only internal screening for primitive exponent pairs (and nuclear centers in the case of nuclear attraction integrals) is done in the integral code. Another feature of the integral code is the complete absence of common blocks. We felt that the inclusion of common blocks compromises the use of the integral package as a universal portable tool that can be used by other programs without possible common block name clashes. Also the use of common blocks makes it harder to maintain the software. All needed data for each routine is hence passed as an argument list.

The following scheme shows the basic steps that are performed for an ERI block (**ab**|**cd**) evaluation:

1. Determine angular momentum map $\mathcal{L} : \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d} \rightarrow \mathbf{a} \geq \mathbf{b}$ and $\mathbf{c} \geq \mathbf{d}$
2. Determine contraction indices map $\mathcal{C} : r, s, t, u \rightarrow r \geq s$ and $t \geq u$
3. Screen primitive exponent pairs $\alpha\beta, \gamma\delta \rightarrow \{\alpha\beta\}, \{\gamma\delta\}$
4. Optimum cache $\{\alpha\beta\}$ and $\{\gamma\delta\}$ block size
5. Loop over $\{\alpha\beta\}$ and $\{\gamma\delta\}$ blocks:
   5a Evaluate block $[\mathbf{e0}|\mathbf{f0}]_{\gamma\delta,\alpha\beta,xyz}$
   5b Move xyz indices $\rightarrow$ $[\mathbf{e0}|\mathbf{f0}]_{xyz,\gamma\delta,\alpha\beta}$
   5c 1st half contraction $\rightarrow$ $(\mathbf{e0}|\mathbf{f0}]_{xyz,\gamma\delta,rs}$
   5d Move rs indices $\rightarrow$ $(\mathbf{e0}|\mathbf{f0}]_{xyz,rs,\gamma\delta}$
   5e 2nd half contraction $\rightarrow$ $(\mathbf{e0}|\mathbf{f0})_{xyz,rs,tu}$

6. Reorder contraction indices *rstu* by $\mathcal{C}^{-1}$
7. Transpose batch $(\mathbf{e0}|\mathbf{f0})_{xyz,rstu} \rightarrow (\mathbf{e0}|\mathbf{f0})_{rstu,xyz}$
8. Generate $xyz \rightarrow rY$ transformation data
9. Apply HRR on **f0** part ($n = rstu$): $(\mathbf{e0}|\mathbf{f0})_{n,xyze,xyzf} \rightarrow (\mathbf{e0}|\mathbf{cd})_{n,xyze,xyzc,xyzd}$
10. Spherical transformation on **d** part $\rightarrow (\mathbf{e0}|\mathbf{cd})_{n,xyze,xyzc,rYd}$
11. Move *rYd* indices into *n* index part using $\mathcal{L}^{-1} \rightarrow (\mathbf{e0}|\mathbf{cd})_{n,xyze,xyzc}$
12. Spherical transformation on **c** part $\rightarrow (\mathbf{e0}|\mathbf{cd})_{n,xyze,rYc}$
13. Move *rYc* indices into *n* index part using $\mathcal{L}^{-1} \rightarrow (\mathbf{e0}|\mathbf{cd})_{n,xyze}$
14. Apply steps (9) through (13) on **e0** part $\rightarrow$ final ERI block (**ab**|**cd**)

Introduction of $\mathcal{L}$ and $\mathcal{C}$ in steps (1) and (2) is crucial for efficiency and optimum performance of the HRR and contraction transformations. The contraction map is determined such that during each of the half contraction steps (5c) and (5e) the intermediate quarter contraction result requires the least amount of intermediate array space (see the contraction scheme in eq. (92) and comments therein).

Screening of the primitive exponent pairs is done in the following way. Consider the ERI formula for a normalized (**00**|**00**) integral:

$$(\mathbf{00}|\mathbf{00}) = \frac{16(\alpha\beta\gamma\delta)^{3/4}}{pq\sqrt{\pi(p+q)}}\mathbf{E}_{AB}\mathbf{E}_{CD}F_0\left(\frac{pq}{p+q}(\mathbf{P}-\mathbf{Q})^2\right). \quad (99)$$

where p, q and the prefactors $\mathbf{E}_{AB}, \mathbf{E}_{CD}$ are defined in eqs. (10) and (12) and F_0 denotes the 0-th auxiliary integral. We always have $|(\mathbf{00}|\mathbf{00})| \geq |(\mathbf{ab}|\mathbf{cd})|$ hence the rhs of eq. (99) denotes an upper bound for any integral in (**ab**|**cd**). Screening of the exponent pairs proceeds by finding the minimum values of each set $(\alpha_{\min}, \beta_{\min}, \gamma_{\min}, \delta_{\min})$ and screening first each $\alpha\beta$ pair against the pair $\gamma_{\min}\delta_{\min}$ following by screening of each $\gamma\delta$ pair against $\alpha_{\min}\beta_{\min}$. One obvious problem is the determination of $(\mathbf{P}-\mathbf{Q})^2$ in the auxiliary integral argument. Since we know that P is somewhere on the line joining center A with B and the same holds for Q with centers C and D, we simply set $(\mathbf{P}-\mathbf{Q})^2$ equal to the square of the minimum distance between the two line segments joining atomic centers A with B and C with D.

Step (4) tries to decompose both primitive exponent sets into subblocks of appropriate size such that the most time consuming step (5a) has minimal cache misses. Note the index rearrangements in (5b) and (5d) [and also in steps (7), (11), and (13)] such that the loops in the routines following these steps run over the nonrelevant integral indices with unit stride. The generation of the Cartesian to spherical transformation data in step (8) is done in such a way that duplicates are excluded. For example, if an integral block (**pf**|**ff**) is requested, only one sparse $\mathbf{T}_{\text{sph}}$ matrix is generated for the **f** angular momentum transformation (the **p** angular momentum obviously requires no transformation at all).

Great care has been exercised to provide a transparent layout of the code. Subroutine names were given names up to 31 characters long, where the first 3 letters indicate the module and the rest is a description of what is done in the routine. As an example we present a listing of subroutine names in the ERD module (*E*lectron *R*epulsion *D*irect), which were given to the routines handling each of the corresponding above steps in the scheme of the ERI block (**ab**|**cd**) evaluation:

1),2)	erd__set_abcd
3)	erd__set_ij_kl_pairs
4)	erd__e0f0_def_blocks
5a),5b)	erd__e0f0_pcgto_block
	erd__rys_roots_weights
	erd__2d_coefficients
	erd__2d_pq_integrals
	erd__int2d_to_e0f0
5b) to 5e)	erd__ctr_4index_block
	erd__transpose_batch
	erd__ctr_1st_half
	erd__map_ijkl_to_ikjl
	erd__ctr_2nd_half_update
6)	erd__ctr_4index_reorder
7)	erd__transpose_batch
8)	erd__xyz_to_ry_abcd
9)	erd__hrr_matrix
	erd__hrr_transform
10),12)	erd__spherical_transform
11),13)	erd__move_ry

Special care has been taken to have the code well structured, such that future developers will not have to spend too much time in getting familiar with the code. Extensive comments (about 30% of the line count) in each routine guide the user and developer about the steps being performed. The argument section of each routine is written in such a way that incoming arguments needed by the routine are well separated from those which are calculated and returned back. This adds legibility to the code and aids when additions and/or changes to the code need to be performed.

Testing and Performance

A crucial aspect of every direct integral package is the time it takes to evaluate the individual integral blocks. The present two-electron integral code (module ERD) timings have been compared with other well established two-electron integral packages implemented in some widely distributed quantum mechanical software programs like GAMESS[27] and MOLCAS.[28] Modified versions (no writing of integrals to disk, no Schwarz screening of integrals, no evaluation of one-electron integrals, etc...) of these codes have been compiled and timing tests were performed on the C_2H_6 ethane molecule in its staggered D3d configuration at a standard geometry of $R_{\text{CC}} = 1.54$ Å, $R_{\text{CH}} = 1.09$ Å and all angles equal to the tetrahedral angle, using a variety of different spherical basis sets. Table 3 shows the results obtained in CPU seconds.

The timings are only a guide to the performance of our code and should by no means be taken too literally. A rough margin error of at least -10% should be applied to all the timings of the other programs. This allows for human error in successfully suppressing all unwanted integral evaluation steps in the MOLCAS and GAMESS program. The main reason for performing the timings is to show that the ACES III integral package is not just merely another integral package created, but it is one that compares favorably with state of the art programs of its kind. The integral code of MOLCAS is one of the fastest integral codes available. The fact that the timings are comparable with the MOLCAS integral package assures us of

Table 3. Complete ERI Evaluation CPU Timings in Seconds for Staggered D3d Ethane C_2H_6 with Geometry $R_{CC} = 1.54$ Å, $R_{CH} = 1.09$ Å, and All Angles = 109.47.

Basis	ERD	MOLCAS	GAMES	Basis size
CC-PVDZ	1.66	2.10	10.2	58
DZP	1.81	2.26	4.01	60
6-311++G**	3.98	5.15	13.9	86
ROOS ADZP	38.9	40.6	4210	100
AUG-CC-PVDZ	5.50	6.51	31.3	100
AUG-CC-PVTZ	114.3	125.0	280.5	230
AUG-CC-PVQZ	1682	2201	2540	436
NASA Ames	2135	2492	≈3 days	290

All timings obtained on a IBM/RS 6000 machine.

no obvious design and implementation mistakes. One observation that can be made, however, from the above table is the fact that the GAMESS integral package slows down considerably when fully contracted basis sets like ROOS ADZP are used. In this case, orders of magnitude slower timings are observed. The slow performance can be traced to the recalculation of primitive integrals for each contraction coefficient quadruplet contribution. As the basis sets contain larger portions of uncontracted GTOs the timings become better, as is seen for example for the AUG-CC-PVQZ basis set.

The same molecular system has also been used for gradient, 2nd and 3rd ERI derivatives. We were interested in the overhead each derivative order creates when compared with evaluation of the basic ERI integrals. Since many more variables arise when evaluating derivatives (which derivative component, which atomic center, etc...), we have to use some kind of averaging in order to compare with the basic integral timings. Consider the following pseudocode for evaluating all the ERI gradients:

```
nblock = 0
nblockder = 0
CPU-total = 0
Loop over all ERI blocks a, b, c, d
        nblock = nblock + 1
        Loop over all x,y,z components
        Loop over all distinct atomic centers
                nblockder = nblockder + 1
                Evaluate ERI gradient block (ab|cd)
                - > CPU-block
                CPU-total = CPU-total + CPU-block
        End do
        End do
End do
CPU = CPU-total * (nblock / nblockder)
```

If the innermost two loops over the components and the centers are omitted and only basic ERI integrals are evaluated we would have nblock = nblockder and variable CPU would contain the values of the basic integral evaluation timings in Table 3. For the gradients, CPU would thus measure how much more time on average it takes to evaluate a derivative integral block. For the higher derivatives we did the following. For the 2nd derivatives, we added an extra x-component differentiation on the first atomic center in the block.

Table 4. *Idem* as Table 3 but Including 1st, 2nd, and 3rd Order Derivatives.

Basis	ERD	1st	2nd	3rd	Basis size
CC-PVDZ	1.66	3.29	4.40	5.32	58
DZP	1.81	3.66	5.05	6.24	60
6-311++G**	3.98	9.69	14.1	18.1	86
ROOS ADZP	38.9	98.3	156.2	207.8	100
AUG-CC-PVDZ	5.50	11.4	16.4	20.9	100
AUG-CC-PVTZ	114.3	172.3	236.3	283.3	230
AUG-CC-PVQZ	1682	2360	3137	3619	436
NASA Ames	2135	4098	5810	7007	290

For the 3rd derivatives, the same as for the 2nd but an additional y-component differentiation on the second atomic center in the block. Results are shown in Table 4, where we obtained average CPU values in seconds for the 1st,2nd and 3rd derivatives. The basic integral timings from Table 3 have also been added for comparison.

From the table, we conclude that evaluation of the 1st, 2nd, and 3rd order derivatives take about 2x, 3x, and 4x longer than evaluation of the basic ERI integrals. Timings become more favorable in basis sets with segmented contractions like for example for the AUG-CC-PVQZ case. The longer timings for the derivatives can partly be explained by the fact that the HRR at the contracted level can no longer be fully used. Instead the HRRs have to be performed at the 2D integral level, thus increasing the number of integrals to be contracted. Overall the obtained overhead factors for the derivative integrals are very satisfactory.

The ERD integral module, as well as the OED module comprising all *O*ne-*E*lectron *D*irect integrals and their open-ended *n*-th order derivatives, has been extensively tested for accuracy and correctness, and is currently being used by several groups running ACES III for real systems calculations involving energy, gradient and Hessian evaluations at Hartree-Fock, MP2 and CC level. All these calculations involve up to 2nd order derivatives. 3rd and 4th order derivative integrals were tested by comparing with numerical differentiation between integrals of one lower order over the entire range of differentiation possibilities. Thus the 3rd derivatives were tested using the 2nd derivative integrals, which we already knew were correct. Once the 3rd derivative integrals passed the correctness test they constitute the basis for the 4th derivative integral test. The 5th and higher order derivatives have not been tested to their completion so far, however, random selected integral blocks for testing showed no errors so far.

Acknowledgments

The development of ACES III was largely supported by the US Department of Defense's High Performance Computing Modernization Program (HPCMP) under two complementary programs: Common High Performance Computing Software Initiative (CHSSI), project CBD-03, and User Productivity Enhancement and Technology Transfer (PET). The development of the integral package was primarily supported by the U.S. Air Force Office of Scientific Research under contract number FA-9550-04-1-01.

References

1. Gill, P. M. W. Adv Quant Chem 1994, 25, 141.
2. Boys, S. F. Proc Roy Soc A (London) 1950, 200, 542.
3. Dupuis, M.; Rys, J.; King, H. F. J Chem Phys 1976, 65, 111.
4. King, H. F.; Dupuis, M. J Comput Phys 1976, 21, 144.
5. Rys, J.; Dupuis, M.; King, H. F. J Comput Chem 1983, 4, 154.
6. McMurchie, L. E.; Davidson, E. R. J Comput Phys 1978, 26, 218.
7. Obara, S.; Saika, A. J Chem Phys 1986, 84, 3963.
8. Schlegel, H. B. J Chem Phys 1982, 77, 3676.
9. Head-Gordon, M.; Pople, J. A. J Chem Phys 1988, 89, 5777.
10. Lindh, R.; Ryu, U.; Liu, B. J Chem Phys 1991, 95, 5889.
11. Schlegel, H. B.; Binkley, J. S.; Pople, J. A. J Chem Phys 1983, 80, 1976.
12. Dupuis, M.; King, H. F. In Geometrical Derivatives of Energy Surfaces and Molecular Properties; Jorgensen, P.; Simons, J. D., Eds.; Reidel Publishing Company: Dordrecht, 1986; p. 167
13. Lindh, R. Theor Chim Acta 1992, 85, 423.
14. Dupuis, M. Comput Phys Commun 2001, 134, 150.
15. Dupuis, M.; Marquez, A. J. Chem. Phys. 2001, 114, 2067.
16. Takashima, H.; Kitamura, K. Chem Phys Lett 2003, 377, 43.
17. Nakai, H.; Kobayashi, M. Chem Phys Lett 2004, 388, 50.
18. Wilf, H. S. Mathematics for the Physical Sciences; Wiley: New York, 2001; p. 80.
19. Golub, G. H.; Welsch, J. H. Math Comput 1969, 23, 221.
20. Abramowitz, M.; Stegun, I. A. Handbook of Mathematical Functions; Dover Publications: New York, 1968, p. 771.
21. Sack, R. A.; Donovan, A. F. Num Math 1971, 18, 465.
22. Wheeler, J. C. Rocky Mountain J Math 1974, 4, 287.
23. Gautschi, W. In Recent Advances in Numerical Analysis; deBoor, C.; Golub, G. H., Eds.; Academic Press: New York, 1978, p. 45.
24. Press, W. H.; Teukolsky, S. A.; Vetterling, W. T.; Flannery, B. P. Numerical Recipies in C++, 2nd ed.; Cambridge University Press: New York, 2002, p. 163.
25. Wimp, J. Computation with Recurrence Relations; Applicable Mathematics Series; Pitman Advanced Publishing Program: Boston, MA, 1984, p. 82.
26. Lindh, R. In Encyclopedia of Computational Chemistry, Vol. 2; V. R. Schleyer, P., Ed.; Wiley: Chichester, 1998, p. 1337.
27. Schmidt, M. W.; Baldridge, K. K.; Boatz, J. A.; Elbert, S. T.; Gordon, M. S.; Jensen, J. H.; Koseki, S.; Matsunaga, N.; Nguyen, K. A.; Su, S.; Windus, T. L.; Dupuis, M.; Montgomery, J. A. J Comput Chem 1993, 14, 1347.
28. Karlström, G.; Lindh, R.; Malmqvist, P.-A.; Ryde, U.; Veryazov, V.; Widmark, P.-O.; Cossi, M.; Schimmelpfennig, B.; Neogrady, P.; Seijo, L. Comp Mat Sci 2003, 28, 222.