EVALUATING MANY-ELECTRON MOLECULAR INTEGRALS FOR QUANTUM CHEMISTRY

James Christopher Womack

A dissertation submitted to the University of Bristol in accordance with the requirements for award of the degree of Doctor of Philosophy in the Faculty of Science. School of Chemistry, August 2015.

Word count: 55233

It's going to work!

Prof F. R. Manby (many times)

ABSTRACT

The evaluation of molecular integrals is a vital but computationally expensive part of electronic structure calculations. This computational expense is particularly problematic for the explicitly correlated methods, in which complicated and numerous integrals over more than two-electrons must be evaluated. The successful R12/F12 methods overcome this difficulty by decomposition of these many-electron integrals by means of approximate resolutions of the identity (RIs). To obtain accurate results with this approach, however, requires large auxiliary basis sets with high angular momentum functions. To address this issue, we present a new RI-free variant of MP2-F12 theory, which uses density fitting to approximate three-electron integrals, rather than RIs. This approach demonstrates improved convergence of calculated energies with respect to the size and maximum angular momentum of the auxiliary basis set compared to the standard RI-based approach. For the systems on which the method was tested, relatively small auxiliary basis sets were sufficient to reduce errors in the correlation energy to less than a millihartree.

The software implementation of the three-electron integral types needed in the new MP2-F12 variant proved to be extremely time-consuming. This difficulty inspired us to develop "Intception", a code generator which generates code for molecular integral evaluation. Intception is capable of automatically implementing code for evaluating a wide range of molecular integral types, using a general theoretical framework based on Obara-Saika-type recurrence relations [1]. To flexibly express integral definitions for use in Intception, a new domain-specific language was created. Testing revealed that the generated code evaluated integrals to a high numerical accuracy and on a reasonable timescale, though somewhat slower than existing optimized implementations. A detailed analysis of the performance of the generated code was undertaken, which suggested some possible routes to improving the efficiency of the code.

ACKNOWLEDGEMENTS

Throughout the course of this PhD, there have been many individuals who have supported and encouraged me. Chief among these is Professor Fred Manby, who I would like to thank for his supervision, advice and infectious enthusiasm. In particular, I would like to thank Fred for placing his confidence in me, and allowing me to pursue the development of "Intception" in the final year of my PhD. I would also like to thank the other members of staff who have helped me along the way, in particular Dr David Tew, whose advice and support I greatly appreciate.

I would also like to thank all my colleagues in the Centre for Computational Chemistry for the many coffee and lunch breaks we have shared, which have been sometimes stimulating, often amusing, and generally good fun.

I am grateful to the EPSRC for funding my PhD research and SCI for supporting me through a SCI scholarship.

Finally, I would like to thank my family, for their constant support and encouragement, and my wife, Lauren, for her love and understanding.

AUTHOR'S DECLARATION

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: ..... DATE:.....

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF CODE LISTINGS

CHAPTER 1

INTRODUCTION

The body of work presented in this thesis consists of two main threads, united by a single theme: the evaluation of molecular integrals in electronic structure theory. The evaluation of molecular integrals is a fundamental component of electronic structure theory, and both main threads of work deal with the computational implementation of these integrals. The first thread is presented in chapter 3 and concerns the development of a new variant of MP2-F12 theory based on an alternative approximation scheme for many-electron integrals. This required the implementation of new molecular integral types in software, which proved to be extremely complex and time-consuming. This experience inspired the second thread of work, presented in chapter 4, in which automatic code generation is considered as a means of reducing the burden of developing integral evaluation software.

This work exists in a context where high-accuracy quantum chemical calculations have become an indispensable tool for chemists and chemical physicists investigating the structure and behaviour of matter [2]. The development of software which performs these calculations is analogous to the construction of precise instruments for measuring physical phenomena, requiring a detailed understanding of the capabilities and limitations of the instrument. The creation of efficient, accurate, integral evaluation code is a significant aspect of the development of such electronic structure software.

While the two main threads of this work share a common theme, they also differ substantially in other respects. Chapter 3 is primarily concerned with the development of a new explicitly correlated electronic structure method, and emphasises the theoretical aspects of this process. In contrast, chapter 4 deals with the development of general molecular integral evaluation software, and therefore focuses on the details of implementing such software. The specific theory needed for each thread is described within the corresponding chapter.

In chapter 2 the common theoretical basis for the entire body of work is presented. Chapter 5 summarizes the key results presented in chapters 3 and 4, emphasizing the connections between the two threads of work and the broader context of the research.

CHAPTER 2

THEORETICAL BACKGROUND

2.1 The foundations of quantum chemistry

The Schrödinger equation [3, 4],

$$i\hbar\frac{\partial\Psi}{\partial t} = \hat{H}\Psi, \tag{2.1}$$

describes how the state of a quantum system, characterized by the wavefunction, Ψ, changes with time, t. The quantum system is specified by the Hamiltonian operator, $\hat{H}$, which has an expectation value equal to the total energy of the system. The time-independent Schrödinger equation,

$$\hat{H}\psi = E\psi, \tag{2.2}$$

with time-independent Hamiltonian, $\hat{H}$, can be derived by separation of space and time variables [5, pp.23-25]. The solutions to this equation, ψ_n, are stationary states with energies E_n. The time-dependence of these states may be simply expressed by an oscillating phase factor [6, pp.170–175], i.e.

$$\Psi_n(t) = \psi_n(0)\exp(-iE_nt). \tag{2.3}$$

For molecular systems, the nuclear and electronic components of the wavefunction can be separated using the Born-Oppenheimer approximation [5, pp.249-250], [7, pp.43-45], which assumes that electrons instantaneously adjust to changes in the positions of relatively much more massive nuclei. Under the Born-Oppenheimer approximation, a time-independent electronic Schrödinger equation can be derived:

$$\hat{H}_{\text{elec}}\psi_{\text{elec}}(\{\mathbf{r}_i\};\{\mathbf{R}_A\}) = E_{\text{elec}}(\{\mathbf{R}_A\})\psi_{\text{elec}}(\{\mathbf{r}_i\};\{\mathbf{R}_A\}). \tag{2.4}$$

The electronic Hamiltonian [7, p.43],

$$\hat{H}_{\text{elec}} = -\frac{1}{2}\sum_{i}^{N_{\text{elec}}}\nabla_i^2 - \sum_{i}^{N_{\text{elec}}}\sum_{A}^{N_{\text{nuc}}}\frac{Z_A}{r_{iA}} + \sum_{i>j}^{N_{\text{elec}}}\frac{1}{r_{ij}} \tag{2.5}$$

describes the motion of N_{elec} electrons in the potential resulting from N_{nuc} nuclei, where Z_A is the charge (in atomic units) of nucleus A, r_{iA} is the distance between electron i and nucleus A, and r_{ij} is the distance between electrons i and j. The first term in this equation describes the kinetic energy of all the electrons in the system, $\hat{T}$, while the second and third terms describe

nuclear-electronic attraction, $\hat{V}_{\text{ne}}$, and electron-electron repulsion, $\hat{V}_{\text{ee}}$, i.e.

$$\hat{H}_{\text{elec}} = \hat{T} + \hat{V}_{\text{ne}} + \hat{V}_{\text{ee}}. \tag{2.6}$$

In quantum chemistry, it is common to partition the electronic Hamiltonian into one-electron and two-electron components,

$$\hat{H}_{\text{elec}} = \sum_{i}^{N_{\text{elec}}} \hat{h}_i + \sum_{i>j}^{N_{\text{elec}}} \frac{1}{r_{ij}} \tag{2.7}$$

with the core-Hamiltonian,

$$\hat{h}_i = -\frac{1}{2}\nabla_i^2 - \sum_{A}^{N_{\text{nuc}}} \frac{Z_A}{r_{iA}}, \tag{2.8}$$

describing the kinetic and potential energy of electron i interacting with N_{nuc} nuclei. The electronic wavefunction $\psi_{\text{elec}}(\{\mathbf{r}_i\};\{\mathbf{R}_A\})$ is an eigenfunction of the electronic Hamiltonian, and is a function of the electronic coordinates, $\{\mathbf{r}_i\}$, while the electronic energy, E_{elec}, is the corresponding eigenvalue. Both of these quantities depend parametrically on the nuclear coordinates, $\{\mathbf{R}_A\}$.

A term accounting for the classical electrostatic internuclear repulsion can be added to the electronic energy,

$$E_{\text{total}} = E_{\text{elec}} + \sum_{A>B}^{N_{\text{nuc}}} \frac{Z_A Z_B}{r_{AB}}, \tag{2.9}$$

to obtain the total energy of a system of electrons moving in the potential resulting from nuclei at fixed coordinates, i.e. the energy of a molecular system.

The lowest-energy solution to the electronic Schrödinger equation at a set of nuclear coordinates is termed the ground state, with higher energy solutions representing excited states. For brevity, we will generally refer to the time-independent electronic Schrödinger equation as simply "the Schrödinger equation".

The ground- and excited-state electronic energies of a molecular system and the change in these with respect to the nuclear coordinates are vital quantities in understanding and predicting the chemistry of a system. Unfortunately, exact, analytic solution of the Schrödinger equation becomes intractable for systems with more than one electron, so that even the helium atom requires approximate solution [6, pp.359-360], [8, p.118]. A principal concern in quantum chemistry is therefore the development of approximate methods for solving of the electronic Schrödinger equation for many-electron systems.

Approximate solutions to eigenvalue equations, such as the Schrödinger equation (Eq. 2.4), can be found using the variational method, where parameters of an approximate trial wavefunction are varied to minimize the energy expectation value of that wavefunction. This is founded on the variational principle, which states that the energy of an approximate trial wavefunction will always be greater than the energy of the exact ground-state wavefunction, and can be derived as follows.

The solutions to the the Schrödinger equation,

$$\hat{H}|\psi_\alpha\rangle = E_\alpha|\psi_\alpha\rangle, \tag{2.10}$$

are a complete orthonormal set of eigenfunctions, $\{\psi_\alpha\}$, with corresponding real eigenvalues, $\{E_\alpha\}$, which can be ordered such that

$$E_0 \leq E_1 \leq E_2 \leq \cdots . \tag{2.11}$$

Since the exact solutions to the Schrödinger equation form a complete set, an approximate trial wavefunction, $|\widetilde{\psi}\rangle$ can be expressed as a linear combination of these,

$$|\widetilde{\psi}\rangle = \sum_\alpha |\psi_\alpha\rangle\langle\psi_\alpha|\widetilde{\psi}\rangle, \tag{2.12}$$

where we have adopted Dirac bra-ket notation [9] for clarity and generality.

For a normalized trial wavefunction,

$$\langle\widetilde{\psi}|\widetilde{\psi}\rangle = 1, \tag{2.13}$$

and by inserting Eq. 2.12, we obtain

$$\langle\widetilde{\psi}|\widetilde{\psi}\rangle = \sum_{\alpha\beta}\langle\widetilde{\psi}|\psi_\alpha\rangle\langle\psi_\alpha|\psi_\beta\rangle\langle\psi_\beta|\widetilde{\psi}\rangle = \sum_\alpha |\langle\widetilde{\psi}|\psi_\alpha\rangle|^2 = 1. \tag{2.14}$$

The expectation value of the energy for a trial wavefunction can be expanded in the same way to yield

$$\langle\widetilde{\psi}|\hat{H}|\widetilde{\psi}\rangle = \sum_{\alpha\beta}\langle\widetilde{\psi}|\psi_\alpha\rangle\langle\psi_\alpha|\hat{H}|\psi_\beta\rangle\langle\psi_\beta|\widetilde{\psi}\rangle = \sum_\alpha E_\alpha|\langle\psi_\alpha|\widetilde{\psi}\rangle|^2. \tag{2.15}$$

Since $E_0 \leq E_\alpha$ for $\alpha \geq 0$,

$$\sum_\alpha E_\alpha|\langle\psi_\alpha|\widetilde{\psi}\rangle|^2 \geq E_0 \sum_\alpha |\langle\psi_\alpha|\widetilde{\psi}\rangle|^2 = E_0. \tag{2.16}$$

where we have used Eq. 2.14. Thus we obtain the inequality

$$\widetilde{E} = \langle\widetilde{\psi}|\hat{H}|\widetilde{\psi}\rangle \geq E_0 \tag{2.17}$$

which succinctly expresses the variational principle [5, pp.183–185], [7, pp.31–33].

If a trial wavefunction is expanded as a linear combination of N basis functions,

$$|\widetilde{\psi}_\mu\rangle = \sum_i^N C_{i\mu}|\Phi_i\rangle, \tag{2.18}$$

then the best approximation of the exact ground-state wavefunction in that basis can be found by minimizing the energy with respect to the coefficients, $\{C_{i\mu}\}$. This is equivalent to solving the matrix eigenvalue problem,

$$\mathbf{HC} = \mathbf{CE}, \tag{2.19}$$

i.e. diagonalizing the $\mathbf{H}$ matrix to obtain an eigenvector matrix of coefficients, $(\mathbf{C}^\mu)_i = C_{i\mu}$, and N eigenvalues E_μ, with $H_{ij} = \langle\Phi_i|\hat{H}|\Phi_j\rangle$. The lowest eigenvalue E_μ and corresponding eigen-

vector $\mathbf{C}^{\mu}$ represent the best approximation of the exact ground-state energy and wavefunction of the system defined by Hamiltonian $\hat{H}$ in the basis set $\{\Phi_i\}$ [7, pp.33–38], [10, pp.113–115].

A physically reasonable solution to the Schrödinger equation must possess certain properties. For example, a physically reasonable wavefunction must be square-integrable and thus able to be normalized, i.e.

$$\int \mathrm{d}\tau\, \psi^*\psi \equiv \langle\psi|\psi\rangle = 1, \tag{2.20}$$

where the integration is over all space. The wavefunction cannot therefore be infinite over any finite region of space and the square of the wavefunction, $\psi^*\psi \equiv |\psi|^2$, must be single-valued. This requirement can be understood in terms of the Born interpretation of the wavefunction as a probability amplitude—the square of the wavefunction is then a probability density and thus must be both single-valued and finite when integrated over all space [5, p.22,43]. As a solution for a second-order differential equation (Eq. 2.2), the wavefunction should also possess a second derivative [5, p.44]. Additionally, the many-electron wavefunctions typically seen in quantum chemistry must obey the Pauli exclusion principle—no two identical fermions may simultaneously occupy the same quantum state. This is equivalent to requiring that the many-electron wavefunction is antisymmetric with respect to interchange of electrons, i.e.

$$|\psi(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N)\rangle = -|\psi(\mathbf{x}_2, \mathbf{x}_1, \cdots, \mathbf{x}_N)\rangle, \tag{2.21}$$

where $\mathbf{x}_i$ is a vector of the spatial and spin coordinates of electron i [7, pp.45–46], [5, pp.225–228].

A trial wavefunction which satisfies the requirements just outlined may be constructed from a determinant of suitable one-electron functions:

$$|\Phi(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N)\rangle = (N!)^{-1/2} \begin{vmatrix} \phi_i(\mathbf{x}_1) & \phi_j(\mathbf{x}_1) & \cdots & \phi_n(\mathbf{x}_1) \\ \phi_i(\mathbf{x}_2) & \phi_j(\mathbf{x}_2) & \cdots & \phi_n(\mathbf{x}_2) \\ \vdots & \vdots & & \vdots \\ \phi_i(\mathbf{x}_N) & \phi_j(\mathbf{x}_N) & \cdots & \phi_n(\mathbf{x}_N) \end{vmatrix} \tag{2.22}$$

This is a "Slater determinant" wavefunction describing a system of N-electrons in terms of a basis of one-electron spin-orbitals, $\{\phi_i(\mathbf{x})\}$ [5, p.227], [7, pp.49–53], [11]. Each row in the determinant corresponds to a single electron coordinate. The mathematical properties of determinants ensure that the antisymmetry requirement is satisfied for Slater determinants, i.e. a determinant with identical rows is zero, and swapping rows in a determinant changes the sign of the determinant. It follows that a non-zero Slater determinant wavefunction will be antisymmetric with respect to interchange of electrons (swapping rows, Eq. 2.21) and must not have electrons occupying the same quantum state (identical rows). A Slater determinant may be compactly represented using only the diagonal elements of the determinant and making the normalization factor $(N!)^{-1/2}$ implicit, i.e.

$$|\Phi(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N)\rangle \equiv |\phi_i\phi_j\phi_k\cdots\phi_n\rangle. \tag{2.23}$$

The spin-orbitals used to construct a Slater determinant wavefunction depend on the coordinates

of a single electron, and are typically considered as products of spatial and spin components,

$$\phi_i(\mathbf{x}) = \begin{cases} \psi_i(\mathbf{r})\alpha(\omega) \\ \psi_i(\mathbf{r})\beta(\omega) \end{cases} \tag{2.24}$$

with the spin component described by one of two orthonormal functions of the spin coordinate: $\alpha(\omega)$ and $\beta(\omega)$. A set of $2M$ spin orbitals, $\{\phi_i(\mathbf{x})\}$ can be formed from M spatial orbitals, $\{\psi_i(\mathbf{r})\}$. The spin-orbitals are generally considered to form an orthonormal set, since this yields simpler equations and derivations [7, p.46], and in quantum chemistry are often referred to as "molecular orbitals" (MOs).

The electronic energy of a wavefunction comprised of a single Slater determinant with an orthonormal spin-orbital basis is

$$\begin{aligned} E &= \langle\Phi|\hat{H}|\Phi\rangle \\ &= \sum_i h_i + \sum_{i>j} J_{ij} - K_{ij} \end{aligned} \tag{2.25}$$

with the core-Hamiltonian,

$$h_i = \langle\phi_i|\hat{h}|\phi_i\rangle = \int \mathrm{d}\mathbf{x}\, \phi_i^*(\mathbf{x})\hat{h}\phi_i(\mathbf{x}) \tag{2.26}$$

Coulomb,

$$J_{ij} = \langle\phi_i\phi_j|r_{12}^{-1}|\phi_i\phi_j\rangle = \int \mathrm{d}\mathbf{x}_1\mathrm{d}\mathbf{x}_2\, \phi_i^*(\mathbf{x}_1)\phi_j^*(\mathbf{x}_2)r_{12}^{-1}\phi_i(\mathbf{x}_1)\phi_j(\mathbf{x}_2) \tag{2.27}$$

and exchange

$$K_{ij} = \langle\phi_i\phi_j|r_{12}^{-1}|\phi_j\phi_i\rangle = \int \mathrm{d}\mathbf{x}_1\mathrm{d}\mathbf{x}_2\, \phi_i^*(\mathbf{x}_1)\phi_j^*(\mathbf{x}_2)r_{12}^{-1}\phi_j(\mathbf{x}_1)\phi_i(\mathbf{x}_2) \tag{2.28}$$

matrix elements [7, p.73], [12, pp.87–92]. The core-Hamiltonian matrix element h_i describes the kinetic energy and nuclear attraction of an electron in spin-orbital ϕ_i, while the Coulomb matrix element J_{ij} describes the classical Coulombic repulsion of electrons in spin-orbitals ϕ_i and ϕ_j. An electronic wavefunction which is a simple product of spin-orbitals (a "Hartree product" [7, pp.47–49])

$$|\Phi_{\mathrm{HP}}\rangle = \prod_i^{N_{\mathrm{elec}}} \phi_i(\mathbf{x}_i) \tag{2.29}$$

has an electronic energy which consists only of these matrix elements:

$$E_{\mathrm{HP}} = \sum_i h_i + \sum_{i>j} J_{ij}. \tag{2.30}$$

The exchange integral K_{ij} arises due to the antisymmetry of the Slater determinant and describes the non-classical correlation of electrons with parallel spins due to the Pauli exclusion principle.

2.1.1 Hartree-Fock theory

Given an approximate electronic wavefunction comprised of a Slater determinant of spin-orbitals $\{\phi_i\}$ (Eq. 2.23) and a Hamiltonian describing a system of electrons and nuclei, $\hat{H}$, the electronic energy, $E[\{\phi_i\}]$, is a functional of the spin-orbitals. The variational principle (Eq. 2.17) indicates that the best approximation to the exact ground-state wavefunction of the system can be obtained by minimizing the electronic energy of the Slater determinant wavefunction (Eq. 2.25) with respect to the spin-orbitals.

To minimize the electronic energy subject to the constraint that the spin-orbitals remain orthonormal,

$$\langle \phi_i | \phi_j \rangle - \delta_{ij} = 0 \qquad \forall\, i, j \tag{2.31}$$

Lagrange's method of undetermined multipliers can be employed. The Lagrangian,

$$L[\{\phi_i\}] = E[\{\phi_i\}] - \sum_{ij}^{N} \epsilon_{ji}(\langle \phi_i | \phi_j \rangle - \delta_{ij}), \tag{2.32}$$

is itself a functional of the spin-orbitals, as well as a function of the Lagrange multipliers, $\{\epsilon_{ji}\}$. Minimizing the Lagrangian with respect to the spin-orbitals, $\{\phi_i\}$, yields the general eigenvalue equations

$$\hat{f}|\phi_i\rangle = \sum_j \epsilon_{ji}|\phi_j\rangle \tag{2.33}$$

with the Fock operator

$$\hat{f} = \hat{h} + \sum_{i}^{N} \hat{J}_i - \hat{K}_i, \tag{2.34}$$

comprised of core Hamiltonian, Coulomb and exchange operators, $\hat{h}$, $\hat{J}_i$ and $\hat{K}_i$. Solving Eq. 2.33 yields the Lagrange multipliers (eigenvalues) and spin-orbitals (eigenfunctions) which minimize the energy of the Slater determinant subject to the orthonormality constraint (Eq. 2.31).

The Coulomb and exchange operators in Eq. 2.34 are defined in line with their respective matrix elements (Eqs. 2.27 and 2.28):

$$J_{ij} = \langle \phi_i | \hat{J}_j | \phi_i \rangle = \langle \phi_i \phi_j | r_{12}^{-1} | \phi_i \phi_j \rangle \tag{2.35}$$

$$K_{ij} = \langle \phi_i | \hat{K}_j | \phi_i \rangle = \langle \phi_i \phi_j | r_{12}^{-1} | \phi_j \phi_i \rangle \tag{2.36}$$

The Coulomb operator $\hat{J}_j$ is simply the Coulomb potential of the orbital product $\phi_j^* \phi_j$, i.e.

$$\hat{J}_j(\mathbf{x}_1) = \int \mathrm{d}\mathbf{x}_2\, \phi_j^*(\mathbf{x}_2) r_{12}^{-1} \phi_j(\mathbf{x}_2). \tag{2.37}$$

The exchange operator, $\hat{K}_j$ is a non-local operator which cannot be understood as a simple potential, since in permuting electron indexes, it depends on the form of the operand ($|\phi_i\rangle$ in Eq. 2.36) over all space.

The expectation values of a Slater determinant wavefunction are invariant to unitary trans-

formations of the spin-orbitals,

$$\phi_i' = \sum_j \phi_j U_{ji}, \tag{2.38}$$

as is the Fock operator (see Ref. 7 (pp.120–122)). Thus any unitary transformation of a set of spin-orbitals which solves Eq. 2.33 would leave the total energy of the wavefunction (Eq. 2.25) unchanged. There exists a unitary transformation of the spin-orbitals which diagonalizes the Hermitian matrix of Lagrange multipliers, $\epsilon_{ij} = \epsilon_{ji}^*$. This unique set of spin-orbitals, $\{\chi_i\}$, is obtained by solution of the Hartree-Fock eigenvalue equations,

$$\hat{f}|\chi_i\rangle = \epsilon_i|\chi_i\rangle, \tag{2.39}$$

and are known as the "canonical spin-orbitals".

For a closed-shell system, integration over the spin coordinates yields spatial Hartree-Fock equations,

$$\hat{f}|\psi_i\rangle = \epsilon_i|\psi_i\rangle, \tag{2.40}$$

with the closed-shell Fock operator

$$\hat{f} = \hat{h} + \sum_i^{N/2} 2\hat{J}_i - K_i, \tag{2.41}$$

and with the Coulomb and exchange operators defined as in Eqs. 2.35 and 2.36, but in terms of the spatial orbitals $\{\psi_i\}$ [7, pp.132–136]. See Refs. 7 (pp.111–122) and 13 (pp.677–680) for further detail regarding the derivation of the Hartree-Fock equations outlined above.

The canonical spin-orbitals offer a useful physical interpretation of the diagonalized Lagrange multipliers, $\{\epsilon_i\}$. The difference in the total energies of an N-electron Slater determinant and an $(N-1)$-electron Slater determinant, constructed from the same set of canonical spin-orbitals, is

$$E_{N-1} - E_N = -\epsilon_j, \tag{2.42}$$

where an electron has been removed from spin-orbital χ_j. Thus the Lagrange multipliers $\{\epsilon_i\}$ can be understood as the electron energies of the spin-orbitals $\{\chi_i\}$ that solve the Hartree-Fock eigenvalue equations (Eq. 2.39). These may also be usefully interpreted as an approximation to the ionization energies for removal of an electron from a spin-orbital (Koopmans' theorem [14], see also Refs. 7 (p.127) and 12 (p.92)).

The orbital energy interpretation of the Lagrange multipliers also arises naturally when Eq. 2.39 is considered as a Schrödinger equation (Eq. 2.2) for a one-electron system. The spin-orbital, $|\chi_i\rangle$, is then a one-electron wavefunction, and ϵ_i is the electronic energy of this wavefunction. In this interpretation, the Fock operator, $\hat{f}$ (Eq. 2.34), is a Hamiltonian operator for an electron interacting with the average potential resulting from the other electrons in a system.

The Coulomb and exchange parts of the Fock operator (Eqs. 2.35 and 2.36) are constructed using spin-orbitals and thus the Fock operator is itself a functional of the spin-orbitals. The Hartree-Fock eigenvalue equations must therefore be solved iteratively, with the Fock opera-

tor being updated in each iteration until the solution converges to within a threshold. This "self-consistent field" (SCF) method was developed in the first half of the 20th century, with significant early contributions from Hartree [15–17], Fock [18,19] and Slater [20] (see Ref. 13 for many other references relating to the early development and application of the SCF method).

Numerical solution of the Hartree-Fock equations is non-trivial for molecular systems, where the canonical spin-orbitals, or molecular orbitals (MOs), may be complicated functions. Computational application of the SCF method for polyatomic systems therefore usually involves the approximate expansion of the spatial components of the MOs as linear combinations of simple, atom-centred, one-electron basis functions, or atomic orbitals (AOs) [10, p.201]:

$$\psi_i(\mathbf{r}) = \sum_{\mu}^{M} C_{\mu i}\varphi_\mu(\mathbf{r}). \tag{2.43}$$

For closed-shell systems, the Roothaan-Hall equations [21,22],

$$\sum_{\nu}^{M} F_{\mu\nu}C_{\nu i} = \epsilon_i \sum_{\nu}^{M} S_{\mu\nu}C_{\nu i}, \tag{2.44}$$

with Fock matrix elements $F_{\mu\nu} = \langle\varphi_\mu|\hat{f}|\varphi_\nu\rangle$ and overlap matrix elements $S_{\mu\nu} = \langle\varphi_\mu|\varphi_\nu\rangle$, express the Hartree-Fock eigenvalue equations (Eq. 2.39) in terms of matrix operations over a finite AO basis and may be compactly expressed in matrix notation:

$$\mathbf{FC} = \mathbf{SC}\epsilon. \tag{2.45}$$

The overlap matrix $\mathbf{S}$ is present in the Roothaan-Hall equations because the AO basis is typically normalized, but non-orthogonal (if it was orthonormal, $\mathbf{S}$ would be the identity matrix, $\mathbf{1}$). Orthogonalizing the AO basis yields the matrix eigenvalue equation

$$\mathbf{F}'\mathbf{C}' = \mathbf{C}'\epsilon \tag{2.46}$$

which may be solved by diagonalization of $\mathbf{F}'$, and where $\mathbf{F}'$ and $\mathbf{C}'$ are the Fock matrix and the matrix of MO coefficients $C'_{\mu i}$ in the orthogonalized AO basis $\{\varphi'_\mu\}$. The orthogonalized MO coefficients may then be transformed back into the non-orthogonal AO basis to yield $\mathbf{C}$ [7, pp.142–145].

The dependence of the Fock operator on the MOs (Eq. 2.41) manifests in the Roothaan-Hall equations as a dependence of the Fock matrix

$$F_{\mu\nu} = H_{\mu\nu} + \sum_{\lambda\sigma}^{M} P_{\sigma\lambda}\left[\langle\mu\lambda|\nu\sigma\rangle - \frac{1}{2}\langle\mu\lambda|\sigma\nu\rangle\right] \tag{2.47}$$

upon the MO coefficients $\mathbf{C}$, via the density matrix:

$$P_{\sigma\lambda} = 2\sum_{i}^{N/2} C_{\sigma i}C^*_{\lambda i}. \tag{2.48}$$

The Roothaan-Hall equations must therefore be solved iteratively, using the MO coefficients resulting from the diagonalization of $\mathbf{F}'$ to form a new density matrix $\mathbf{P}$ in the non-orthogonal basis, and using this to form a new $\mathbf{F}$ with the core Hamiltonian,

$$H_{\mu\nu} = \langle\mu|\hat{h}|\nu\rangle = \int \mathrm{d}\mathbf{r}\varphi_\mu^*(\mathbf{r})\hat{h}\varphi_\nu(\mathbf{r}), \tag{2.49}$$

and two-electron Coulomb integrals

$$\langle\mu\lambda|\nu\sigma\rangle = \int \mathrm{d}\mathbf{r}_1\mathrm{d}\mathbf{r}_2\varphi_\mu^*(\mathbf{r}_1)\varphi_\lambda^*(\mathbf{r}_2)r_{12}^{-1}\varphi_\nu(\mathbf{r}_1)\varphi_\sigma(\mathbf{r}_2^*). \tag{2.50}$$

The new Fock matrix and MO coefficients are used to start the next iteration, and the cycle is repeated until some convergence criterion is satisfied (e.g. the difference in the total electronic energy for consecutive iterations is less than a threshold).

In practice, convergence can be difficult to achieve using the naive approach just outlined, and techniques for improving convergence, such as direct inversion of the iterative subspace (DIIS) [23, 24], are typically employed. For an overview of various methods for improving SCF convergence, see Ref. 12 (pp.101–104).

Solving the Hartree-Fock equations using the SCF procedure (or a variation that improves convergence) yields MO coefficients $\mathbf{C}$ which describe the MOs used to construct a Slater determinant (Eq. 2.43) that is an approximate ground-state solution to the electronic Schrödinger equation for the system. The energy of the resulting wavefunction is a minimum, which, according to the variational principle (Eq. 2.17), corresponds to the "best" Slater determinant approximation to the exact ground-state solution for a given AO basis, $\{\varphi_\mu\}$.

Diagonalizing the Fock matrix in a basis of M AOs, yields M eigenvalues $\{\epsilon_r\}$. Where M is greater than the number of occupied spatial MOs, N_{occ} ($N_{\mathrm{elec}}/2$ for a closed-shell system), only the MOs with the lowest N_{occ} eigenvalues are used to construct the ground-state Slater determinant wavefunction. The remaining $M - N_{\mathrm{occ}}$ unoccupied, or "virtual" MOs are solutions to the Hartree-Fock eigenvalue equations (Eq. 2.40), but do not feature in the ground-state Slater determinant wavefunction. As with the occupied MOs, the eigenvalues corresponding to the virtual MOs, $\{\epsilon_a\}$, may be interpreted as orbital energies, representing the energy of an additional electron occupying a virtual MO interacting with all N_{elec} electrons of the closed-shell system. The difference in energy between an N-electron and an $(N+1)$-electron Slater determinant constructed from the set of MOs corresponding to the N-electron system is

$$E_N - E_{N+1} = -\epsilon_a \tag{2.51}$$

and approximates the electron affinity for inserting an additional electron into the virtual spatial MO φ_a.

There are two main sources of error in the approximation of ionization energies and electron affinities using Hartree-Fock orbital energies (Eqs. 2.42 and 2.51), i.e. Koopmans' theorem [14]: the neglect of orbital relaxation in the $(N \pm 1)$-states, and the treatment of electron-electron interactions through an average potential in Hartree-Fock theory. In the case of ionization energies, these two errors tend to cancel, while for electron affinities, they are additive. For

this reason, Koopmans' ionization energies often offer a reasonable qualitative approximation to experimental data, while Koopmans' electron affinities are generally not useful [7, pp.123-128,pp.194–200].

2.2 The electron correlation problem

Hartree-Fock theory is a "mean-field" theory where each electron interacts with an average potential representing all the other electrons in a many-electron system. This approximation allows each electron to be considered independently, greatly simplifying the process of approximately solving the Schrödinger equation. In reality, electron-electron interactions for many-electron systems represent a complex many-body problem.

For a system of statistically uncorrelated electrons, the pair density in an N-electron system,

$$\rho_2(\mathbf{x}_1, \mathbf{x}_2) = N(N-1) \int \mathrm{d}\mathbf{x}_3 \cdots \mathrm{d}\mathbf{x}_N \Psi^*(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)\Psi(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N), \tag{2.52}$$

would be a product of one-electron densities,

$$\rho_2(\mathbf{x}_1, \mathbf{x}_2) = \frac{N-1}{N}\rho(\mathbf{x}_1)\rho(\mathbf{x}_2), \tag{2.53}$$

where

$$\rho(\mathbf{x}_1) = N \int \mathrm{d}\mathbf{x}_2 \cdots \mathrm{d}\mathbf{x}_N \Psi^*(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)\Psi(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N). \tag{2.54}$$

Since electrons interact with each other, via Coulomb repulsion and also via Pauli exclusion, pair densities for true many-electron systems do not factorize in this way [25].

In quantum chemistry, the "electron correlation energy" is typically defined as the difference between the electronic energy of an exact ground-state solution to the Schrödinger equation and the Hartree-Fock energy of the same system, i.e.

$$E_{\mathrm{corr}} = E_{\mathrm{exact}} - E_{\mathrm{HF}}. \tag{2.55}$$

Defined in this way, E_{corr} describes the total error in the Hartree-Fock electronic energy, E_{HF}, relative to the exact electronic energy, E_{exact}.

Hartree-Fock theory can provide a reasonable qualitative description of the electronic structure of a polyatomic system and in some cases can also provide useful quantitative results. Unfortunately, SCF calculations are often not sufficiently accurate to describe the behaviour of many-electron systems, and may produce qualitatively incorrect results (for examples of where Hartree-Fock theory both succeeds and fails, see Ref. 7 (pp.190-205)). Where E_{HF} represents an exact solution to the Hartree-Fock eigenvalue equations (Eq. 2.39) (e.g. where a complete AO basis set is used), E_{corr} describes the error due to the approximation of the exact wavefunction as a single Slater determinant.

Though Hartree-Fock theory is an independent particle model, with electrons interacting through an average potential, the electrons in a Hartree-Fock model are not completely uncorrelated and the pair density, $\rho_2(\mathbf{x}_1, \mathbf{x}_2)$, does not factorize as in Eq. 2.53. This correlation arises because same-spin electrons are prevented from occupying the same region in space due

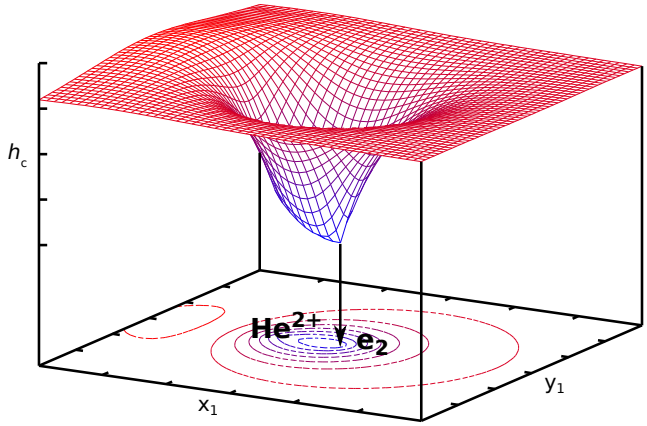

Figure 2.1: Plot illustrating the structure of the Coulomb hole for the helium atom. The Coulomb hole, h_c (Eq. 2.56), is plotted with electron 2 held at a constant position away from the nucleus and with the position of electron 1 varied in the $(x_1, y_1, 0)$ plane. Modified version of a plot originally created by D. P. Tew, reproduced with permission.

to the Pauli exclusion principle, a requirement imposed via the antisymmetric nature of the Hartree-Fock wavefunction (Eqs. 2.21 and 2.22). This is "Fermi correlation", and is distinct from "Coulomb correlation", which is the result of the Coulombic repulsive interaction between electrons, and which is not accounted for in Hartree-Fock theory [10, p.158], [25].

The "Coulomb hole" is defined as the difference between the Hartree-Fock wavefunction, Ψ_{HF}, and the wavefunction which exactly solves the Schrödinger equation:

$$h_c = \Psi - \Psi_{\mathrm{HF}}. \tag{2.56}$$

Hartree-Fock theory overestimates the probability of electrons being close together in space, so the amplitude of the Hartree-Fock wavefunction is too large where the interelectronic distance, r_{12}, is small. The hole-like structure of h_c is clearly demonstrated for the helium atom. If the position of one electron is fixed at some distance from the nucleus and h_c is plotted for the movement of the other electron in a two-dimensional plane, the result is a hole-like depression around the fixed electron's position [10, pp.256–259], [25], as illustrated in Fig. 2.1.

At the point of electron-electron coalescence ($r_{12} \to 0$), the electron repulsion term in the Hamiltonian (Eq. 2.7) becomes singular and there is a corresponding cusp in the exact wavefunction (and therefore also, h_c) where the wavefunction is nondifferentiable. The requirement that the wavefunction satisfies the Schrödinger equation at electron-electron coalescence (or equivalently, that the local energy remains constant over all space) allows the derivation of conditions on the behaviour of trial wavefunctions [26–28]. For example, for a spherically-symmetric two-electron system with singlet spin, the Coulomb cusp condition

$$\left.\frac{\partial \Psi}{\partial r_{12}}\right|_{r_{12}=0} = \frac{1}{2}\Psi(r_{12}=0), \tag{2.57}$$

describes the behaviour of the derivative of the exact wavefunction with respect to the interelectronic distance, r_{12}, at electron-electron coalescence (see Ref. 10 (pp.256–262) for a full

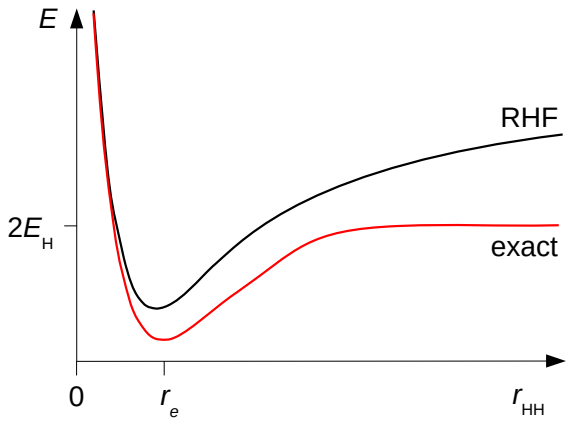

Figure 2.2: Schematic plot demonstrating the incorrect behaviour of restricted Hartree-Fock theory for the dissociation of H_2. Adapted from Fig. 5.12 of Ref. 10.

derivation). Such coalescence conditions assist in the construction of trial wavefunctions which accurately model the exact wavefunction, informing the development of the wavefunction ansatz and choice of basis function.

The electron correlation energy, E_{corr} (Eq. 2.55), may be understood as arising from a combination of "static" and "dynamic" correlation effects. Dynamic correlation arises because the mean-field model of Hartree-Fock theory describes electron-electron repulsion in an average sense, neglecting instantaneous "dynamic" interactions. As a consequence, the Coulomb cusp and Coulomb hole are not reproduced at all in Hartree-Fock wavefunctions. In contrast, static correlation results from the inadequacy of a single determinant wavefunction for describing systems with degenerate, or near-degenerate states (i.e. states equal or close to each other in energy).

Static correlation appears as a distinct effect when considering the dissociation of the H_2 molecule. At long bond lengths, dynamic correlation should be negligible, as the system dissociates to two H atoms, each with a single electron. However, E_{corr} does not go to zero in the dissociation limit for H_2, since the Hartree-Fock single-determinant closed-shell description of the system cannot represent this situation (see for example Fig. 2.2). At the dissociation limit, the bonding and antibonding singlet configurations of H_2 become degenerate in energy and a multi-determinant wavefunction is necessary to fully describe the system (see Refs. 7 (pp.221–229), 10 (pp.159–162), and 12 (pp.145–148) for more detail).

Both dynamic and static correlation effects arise because a single Slater determinant wavefunction is not sufficient to describe features of the exact wavefunction. The representation of both kinds of correlation is improved by the use of trial wavefunctions consisting of multiple determinants, and when this is done, it is generally not possible to separate "static" and "dynamic" contributions to the electronic energy.

2.2.1 Configuration-interaction theory

The problem of describing electron correlation using an N-electron trial wavefunction constructed from a basis of one-electron molecular orbitals can be approached by expanding the

trial wavefunction as in a basis of N-electron Slater determinants. This is the approach taken in configuration interaction (CI) theory, where the coefficients in a linear expansion of Slater determinants,

$$|\Psi\rangle = \sum_I c_I |\Phi_I\rangle \tag{2.58}$$

are determined by variational optimization of the electronic energy. This is equivalent to solving a matrix eigenvalue problem (Eq. 2.19) with Hamiltonian matrix elements between individual Slater determinants:

$$H_{IJ} = \langle \Phi_I | \hat{H} | \Phi_J \rangle. \tag{2.59}$$

Using the occupied and virtual MOs determined in a SCF calculation, a basis set of N-electron Slater determinants can be constructed. The individual determinants can be understood as single-determinant configurations where electrons have been excited from the occupied MOs of the Hartree-Fock ground state into virtual orbitals. The CI wavefunction is then

$$|\Psi\rangle = |\Phi_0\rangle + \sum_{i,a} c_i^a |\Phi_i^a\rangle + \frac{1}{4} \sum_{ij,ab} c_{ij}^{ab} |\Phi_{ij}^{ab}\rangle + \dots \tag{2.60}$$

where $|\Phi_0\rangle$ is the ground-state Hartree-Fock wavefunction and $|\Phi_i^a\rangle$ is a Slater determinant with occupied MO i replaced with virtual MO a, i.e. an excitation of a single electron, $i \to a$. Similarly, $|\Phi_{ij}^{ab}\rangle$ represents a doubly-excited Slater determinant with occupied MOs i, j replaced with virtual MOs a, b. In Eq. 2.60, we have used "intermediate normalization" where the coefficients are scaled such that $\langle \Psi | \Phi_0 \rangle = 1$ [7, p.237].

The set of all possible N-electron Slater determinants formed from a complete set of one-electron Hartree-Fock MOs is a complete set of N-electron functions [29]. "Full" configuration interaction (FCI) calculations employ a complete set of N-electron functions, and, with a complete one-electron MO basis, recover all of E_{corr} (Eq. 2.55) [10, pp.143–146]. A FCI calculation using the approximate MOs obtained from an SCF calculation will yield an approximate ground-state energy, E_{FCI}, variationally minimized for the one-electron AO basis used. This leads to the definition of the *basis set* correlation energy [7, p.234],

$$E_{\text{corr}} = E_{\text{FCI}} - E_{\text{HF}}, \tag{2.61}$$

which is the maximum correlation energy available for a given AO basis, an approximation to the exact correlation energy (Eq. 2.55). Since FCI recovers all the correlation energy for a given AO basis, this correlation energy is often used to benchmark other, more approximate, electronic structure methods using the same AO basis.

The number of Slater determinants in the N-electron basis increases exponentially with the size of the MO basis such that the variational optimization of the FCI wavefunction using the standard matrix eigenvalue approach becomes computationally infeasible for all but very small systems (see Ref. 10 (pp.524–525) for details). The size of the FCI wavefunction may be reduced by making use of spin and spatial symmetry, and the cost of performing FCI calculations may be decreased using iterative (direct CI [30], see also Ref. 31) and stochastic (FCIQMC [32]) approaches.

The standard approach for reducing the size of the CI wavefunction so that CI calculations may be performed on larger systems is truncation of the linear expansion (Eq. 2.60) at a particular level of excitation. For example, in the "CISD" method, the FCI wavefunction is truncated to contain only singly and doubly excited Slater determinants:

$$|\Psi\rangle = |\Phi_0\rangle + \sum_{i,a} c_i^a |\Phi_i^a\rangle + \frac{1}{4}\sum_{ij,ab} c_{ij}^{ab} |\Phi_{ij}^{ab}\rangle. \tag{2.62}$$

Calculations of this type generally capture the majority of the basis set correlation energy, while avoiding the exponential growth in the number of Slater determinants seen in the FCI expansion (Eq. 2.60).

Truncated CI calculations suffer from errors due to the incompleteness of both the one-electron (AO) basis and the N-electron (Slater determinant) basis, but may still provide a reasonable approximation to the basis set correlation energy (see Ref. 10 (pp.182–183), for example). In particular, truncation at the double excitation level provides a reasonable approximation because the doubly excited Slater determinants tend to make the largest contribution to the FCI wavefunction [12, pp.143–144]. Unfortunately, truncation of the FCI wavefunction introduces a more pernicious source of error: size-inconsistency.

A size-consistent theory predicts that the energy of two non-interacting fragments, considered as a single system, is identical to the sum of energies of those fragments treated independently [33], i.e. for non-interacting fragments A and B [34, pp.9–15]:

$$E_{AB} = E_A + E_B. \tag{2.63}$$

A related property is size-extensivity—size-extensive calculations exhibit correct scaling of the energy with the number of fragments (which may be interacting) [35]. The properties of size-consistency and size-extensivity are desirable for quantum chemical methods: a size-inconsistent method will predict incorrect dissociation energies, while the error in energies calculated by a non-size-extensive method increases with system size. Full CI calculations are both size-consistent and size-extensive, but truncated CI calculations are neither (see Refs. 7 (pp.261–265) and 34 (pp.9–15) for demonstrations of this).

In considering the lack of size-consistency in truncated CI, it is helpful to recast the FCI wavefunction for AB in terms of excitation operators, $\hat{T}_n$, acting on the Hartree-Fock reference Slater determinant, e.g.

$$\hat{T}_1|\Phi_0\rangle = \sum_{i,a} t_a^i |\Phi_i^a\rangle \tag{2.64}$$

$$\hat{T}_2|\Phi_0\rangle = \frac{1}{4}\sum_{ij,ab} t_{ab}^{ij} |\Phi_{ij}^{ab}\rangle \tag{2.65}$$

with "amplitudes" t_a^i and t_{ab}^{ij} [12, p.169]. The FCI wavefunction is then

$$|\Psi\rangle = (1 + \hat{T})|\Phi_0\rangle \tag{2.66}$$

with the excitation operator

$$\hat{T} = \sum_{n}^{N_{\text{elec}}} \hat{T}_n. \tag{2.67}$$

For simplicity, we will ignore the single excitations and consider the CID wavefunction:

$$|\Psi_{\text{CID}}\rangle = (1 + \hat{T}_2)|\Phi_0\rangle. \tag{2.68}$$

For a pair of non-interacting fragments, A and B (Eq. 2.63), the Hamiltonian is additively separable,

$$\hat{H}_{AB} = \hat{H}_A + \hat{H}_B \tag{2.69}$$

and the reference Hartree-Fock wavefunction is multiplicatively separable,

$$|\Phi_0(AB)\rangle = |\Phi_0(A)\rangle|\Phi_0(B)\rangle. \tag{2.70}$$

The total FCI wavefunction is similarly multiplicatively separable, and thus FCI is size-consistent [10, pp.126–129,527–529], i.e.

$$\hat{H}_{AB}|\Psi_{\text{FCI}}(AB)\rangle = (E_A + E_B)|\Psi_{\text{FCI}}(AB)\rangle. \tag{2.71}$$

The total CID wavefunction is

$$|\Psi_{\text{CID}}(AB)\rangle = (1 + \hat{T}_2^A + \hat{T}_2^B)|\Phi_0(AB)\rangle, \tag{2.72}$$

where $\hat{T}_n^A$ is an excitation operator which excites n electrons on fragment A. Double excitations across both fragments can be safely neglected since they do not interact with other configurations. Only double excitations occurring on a single non-interacting fragment result in non-zero off-diagonal Hamiltonian matrix elements (Eq. 2.59) which contribute to the ground-state energy. The product of CID wavefunctions for the individual fragments has an energy equal to the sum of the individual fragment energies,

$$\hat{H}_{AB}|\Psi_{\text{CID}}(A)\rangle|\Psi_{\text{CID}}(B)\rangle = (E_A + E_B)|\Psi_{\text{CID}}(A)\rangle|\Psi_{\text{CID}}(B)\rangle, \tag{2.73}$$

and contains an additional term not present in Eq. 2.72:

$$|\Psi_{\text{CID}}(A)\rangle|\Psi_{\text{CID}}(B)\rangle = |\Psi_{\text{CID}}(AB)\rangle + \hat{T}_2^A\hat{T}_2^B|\Phi_0(A)\rangle|\Phi_0(B)\rangle. \tag{2.74}$$

This term corresponds to simultaneous double excitations occurring on both fragments and is absent from $|\Psi_{\text{CID}}(AB)\rangle$, causing size-inconsistency, i.e.

$$\hat{H}_{AB}|\Psi_{\text{CID}}(AB)\rangle \neq (E_A^{\text{CID}} + E_B^{\text{CID}})|\Psi_{\text{CID}}(AB)\rangle. \tag{2.75}$$

The same arguments can be applied to truncated CI in general.

2.2.2 Coupled cluster theory

Coupled cluster theory introduces an exponential wavefunction ansatz,

$$|\Psi\rangle = \mathrm{e}^{\hat{T}}|\Phi_0\rangle, \tag{2.76}$$

where $\hat{T}$ is the excitation, or "cluster" operator of Eq. 2.67. This wavefunction ansatz was initially applied in the context of nuclear physics by Coester and Kümmel [36, 37], and was later adopted for use in solving the electronic Schödinger equation (Eq. 2.2) by Čížek [38]. In the decades since this initial work, coupled cluster theory has seen significant development with numerous variants and approximations introduced (see Refs. 39 and 40 and references therein). Coupled cluster theory (in particular, the CCSD(T) variant [41, 42]) is now considered to be a "gold-standard" benchmark for accurate electronic structure calculations on polyatomic systems where FCI calculations are infeasible [43].

One of the most attractive features of coupled cluster theory is that it exhibits correct scaling with respect to system size (i.e. it is size-consistent and size-extensive), regardless of the truncation level of the cluster operator, $\hat{T}$ (Eq. 2.67). This is in stark contrast with configuration interaction theory, where any truncation of $\hat{T}$ results in non-size-consistent/extensive methods.

That size-consistency is intrinsic to the exponential wavefunction ansatz (Eq. 2.76) can be demonstrated by considering again a pair of non-interacting fragments, A and B. With an additively separable cluster operator truncated at a specific excitation level

$$\hat{T}^{AB} = \hat{T}^{A} + \hat{T}^{B}, \tag{2.77}$$

the linear wavefunction ansatz of CI leads to size-inconsistency, since the total wavefunction is not multiplicatively separable (e.g. CID for the non-interacting system AB, Eq. 2.74). In contrast, applying the coupled cluster wavefunction ansatz with any additively separable cluster operator, leads to a multiplicatively separable total wavefunction [34, p.254]:

$$\begin{aligned} |\Psi_{\mathrm{CC}}(AB)\rangle &= \mathrm{e}^{\hat{T}^{AB}}|\Phi_0(A)\rangle|\Phi_0(B)\rangle \\ &= \mathrm{e}^{\hat{T}^{A}}|\Phi_0(A)\rangle\mathrm{e}^{\hat{T}^{B}}|\Phi_0(B)\rangle \equiv |\Psi_{\mathrm{CC}}(A)\rangle|\Psi_{\mathrm{CC}}(B)\rangle. \end{aligned} \tag{2.78}$$

This leads to additive separability of the energy,

$$(\hat{H}_A + \hat{H}_B)|\Psi_{\mathrm{CC}}(AB)\rangle = (E_{\mathrm{A}} + E_{\mathrm{B}})|\Psi_{\mathrm{CC}}(AB)\rangle \tag{2.79}$$

and thus size-consistency.

It is informative to consider the exponentiated cluster operator as a Taylor series:

$$\mathrm{e}^{\hat{T}} = \sum_{n=0}^{\infty} \frac{\hat{T}^n}{n!} = 1 + \hat{T} + \frac{1}{2}\hat{T}^2 + \frac{1}{6}\hat{T}^3 + \cdots \tag{2.80}$$

The CCSD variant of coupled cluster theory uses a cluster operator truncated at double exci-

tations:

$$\begin{aligned} \mathrm{e}^{\hat{T}_{\mathrm{CCSD}}} = 1 + \hat{T}_1 + \hat{T}_2 + \frac{1}{2}\hat{T}_1^2 + \hat{T}_1\hat{T}_2 + \frac{1}{2}\hat{T}_2^2 \\ + \frac{1}{6}\hat{T}_2^3 + \frac{1}{2}\hat{T}_1^2\hat{T}_2 + \frac{1}{2}\hat{T}_1\hat{T}_2^2 + \frac{1}{6}\hat{T}_2^3 + \cdots \end{aligned} \tag{2.81}$$

The second and third terms of the power series are equivalent to the CISD cluster operator, and these isolated $\hat{T}_n$ terms are typically labelled as "connected". The subsequent terms, which consist of products of the $\hat{T}_1$ and $\hat{T}_2$ operators, are "disconnected" and, when acting on the Hartree-Fock reference state, produce Slater determinants excited beyond the truncation level of the cluster operator. A truncated coupled cluster wavefunction can thus be considered to be an approximation of the FCI wavefunction, where the amplitudes for higher-order excitations are approximated by products of amplitudes for the lower-order excitations via the disconnected terms (e.g. for CCSD, all coefficients for higher-order terms are approximated in terms of t_i^a and t_{ij}^{ab} from Eqs. 2.64, 2.65) [10, pp.187–189].

In principle, the coupled cluster wavefunction could be optimized by variationally minimizing the energy (Eq. 2.17) with respect to the coupled cluster amplitudes. In practice, this is not the approach taken, since this requires the solution of complicated non-linear equations which involve all possible excited Slater determinants, the number of which increases exponentially with the number of MOs. Instead, the coupled cluster energy and amplitudes are obtained by projection of the Schödinger equation with individual Slater determinants from the N-electron basis, i.e.

$$\langle \Phi_I | \hat{H} | \Psi_{\mathrm{CC}} \rangle = E \langle \Phi_I | \Psi_{\mathrm{CC}} \rangle. \tag{2.82}$$

If $|\Phi_I\rangle$ is the Hartree-Fock reference state, then we obtain an equation for the coupled cluster energy,

$$\langle \Phi_0 | \hat{H} | \Psi_{\mathrm{CC}} \rangle = E, \tag{2.83}$$

when intermediate normalization is used ($\langle \Phi_0 | \Psi_{\mathrm{CC}} \rangle = 1$), while projection with other excited Slater determinants produces a series of non-linear equations, from which the amplitudes may be determined. In contrast to the variational equations, the non-linear equations only feature determinants that differ from $|\Phi_I\rangle$ by two or fewer spin-orbitals and may be practically solved by iterative methods. An alternative set of non-linear equations can be derived using a "similarity-transformed Hamiltonian",

$$\langle \Phi_I | \mathrm{e}^{-\hat{T}} \hat{H} \mathrm{e}^{\hat{T}} | \Phi_0 \rangle = E \langle \Phi_I | \Phi_0 \rangle, \tag{2.84}$$

with the advantage that the amplitude and energy equations are decoupled.

Though coupled cluster theory is generally non-variational as implemented in software, the results of calculations using truncated variants, such as CCSD and CCSD(T), are typically very accurate (provided the Hartree-Fock reference state offers a reasonable approximate starting-point)—see for example Refs. 39 and 44. For further theoretical detail regarding coupled cluster theory and information about the implementation of coupled cluster theory, see also Refs. 10 (ch.13, p.648) and 40.

2.2.3 Many-body perturbation theory

The task of solving the Schrödinger equation for a complicated many-body system may sometimes be cast as a problem of describing a small perturbation to a simpler unperturbed system, for which the solutions are known. In Rayleigh-Schrödinger perturbation theory (RSPT), the Hamiltonian operator is partitioned into an unperturbed part and a part describing the perturbation, i.e.

$$\hat{H} = \hat{H}_0 + \hat{H}'. \tag{2.85}$$

The unperturbed, or "zeroth-order" problem,

$$\hat{H}_0|\Psi_\alpha^{(0)}\rangle = E_\alpha^{(0)}|\Psi_\alpha^{(0)}\rangle \tag{2.86}$$

is an approximation to the full problem, chosen to be soluble with a known set of zeroth-order eigenfunctions, $\{\Psi_\alpha^{(0)}\}$. Solving the full Schrödinger equation,

$$(\hat{H}_0 + \hat{H}')|\Psi_\alpha\rangle = E_\alpha|\Psi_\alpha\rangle \tag{2.87}$$

can then be cast as a problem of determining a correction to the zeroth-order problem.

To derive equations for the corrections to the zeroth-order solutions, a parameter, λ, is introduced:

$$\hat{H} = \hat{H}_0 + \lambda\hat{H}'. \tag{2.88}$$

This parameter determines the extent of the perturbation, with $\lambda = 0$ representing the unperturbed system and $\lambda = 1$ the fully perturbed system. The eigenfunctions and eigenvalues of the full problem are then expanded as Taylor series in λ around $\lambda = 0$,

$$E_\alpha = E_\alpha^{(0)} + \sum_{m=1}^{\infty} \lambda^m E_\alpha^{(m)} \tag{2.89}$$

$$|\Psi_\alpha\rangle = |\Psi_\alpha^{(0)}\rangle + \sum_{m=1}^{\infty} \lambda^m |\Psi_\alpha^{(m)}\rangle \tag{2.90}$$

where the first term in each series corresponds to a solution to the zeroth-order problem and subsequent terms are mth-order corrections to the zeroth-order solution. Inserting these expansions into Eq. 2.87 leads to a series of equations, derived by equating terms with λ raised to the mth power,

$$(\hat{H}_0 - E_\alpha^{(0)})|\Psi_\alpha^{(m)}\rangle = (E_\alpha^{(1)} - \hat{H}')|\Psi_\alpha^{(m-1)}\rangle + \sum_{n=0}^{m-2} E_\alpha^{(m-n)}|\Psi_\alpha^{(n)}\rangle, \tag{2.91}$$

where the summation in the last term on the right vanishes if $m < 2$. Using intermediate normalization such that $\langle\Psi_\alpha^{(0)}|\Psi_\alpha\rangle = 1$, and projecting Eq. 2.91 with the zeroth-order wavefunction, leads to a series of equations for the mth-order corrections to the zeroth-order energy ($m > 0$):

$$E_\alpha^{(m)} = \langle\Psi_\alpha^{(0)}|\hat{H}'|\Psi_\alpha^{(m-1)}\rangle. \tag{2.92}$$

The first-order correction can therefore be obtained using only the zeroth-order wavefunction,

while evaluation of the second-order correction to the energy requires the first-order wavefunction:

$$E_\alpha^{(2)} = \langle \Psi_\alpha^{(0)} | \hat{H}' | \Psi_\alpha^{(1)} \rangle. \tag{2.93}$$

To express the second-order energy in terms of known quantities, we can expand the first-order wavefunction in the basis of eigenfunctions of the unperturbed Hamiltonian (Eq. 2.86),

$$|\Psi_\alpha^{(1)}\rangle = \sum_{\beta \neq \alpha} |\Psi_\beta^{(0)}\rangle \langle \Psi_\beta^{(0)} | \Psi_\alpha^{(1)} \rangle \tag{2.94}$$

where α is excluded from the summation over β because $\langle \Psi_\alpha^{(0)} | \Psi_\alpha^{(1)} \rangle = 0$. An expression for $\langle \Psi_\beta^{(0)} | \Psi_\alpha^{(1)} \rangle$ can be obtained by projecting the the first-order ($m = 1$) version of Eq. 2.91 with $|\Psi_\beta^{(0)}\rangle$ and rearranging:

$$\langle \Psi_\beta^{(0)} | \Psi_\alpha^{(1)} \rangle = \frac{\langle \Psi_\beta^{(0)} | \hat{H}' | \Psi_\alpha^{(0)} \rangle}{E_\alpha^{(0)} - E_\beta^{(0)}}. \tag{2.95}$$

Substituting Eqs, 2.94 and 2.95 into Eq. 2.93 results in the well-known expression for the second-order energy in terms of the solutions to the unperturbed problem:

$$E_\alpha^{(2)} = \sum_{\beta \neq \alpha} \frac{\langle \Psi_\alpha^{(0)} | \hat{H}' | \Psi_\beta^{(0)} \rangle \langle \Psi_\beta^{(0)} | \hat{H}' | \Psi_\alpha^{(0)} \rangle}{E_\alpha^{(0)} - E_\beta^{(0)}}. \tag{2.96}$$

Expressions for the higher-order corrections to the energy and wavefunction can be derived by a similar procedure, though the algebraic manipulations quickly become unwieldy. Diagrammatic techniques have been devised which simplify the derivation and manipulation of expressions in many-body perturbation theory (see Refs. 7 (ch.6) and 34 for an introduction to such techniques). For further details regarding the derivation of RSPT, see Refs. 7 (pp.322–327), 10 (pp.725–739), 12 (pp.159–162) and 34 (pp.18–27).

Møller-Plesset perturbation theory (MPPT) [45] is a formulation of RSPT where the unperturbed Hamiltonian is chosen to be a sum over one-electron Fock operators (Eq. 2.34), i.e.

$$\hat{H}_0 = \sum_n^{N_{\text{elec}}} \hat{f}(n), \tag{2.97}$$

where the index n is used to label the electron index which $\hat{f}$ acts upon. The Hartree-Fock ground-state wavefunction, $|\Phi_0\rangle$ is then an eigenfunction of the unperturbed Schödinger equation, i.e.

$$\hat{H}_0 |\Phi_0\rangle = E_0 |\Phi_0\rangle \tag{2.98}$$

where the zeroth-order energy is equal to a sum over eigenvalues of the Fock equations (Eqs. 2.39 and 2.40),

$$E_0 = \sum_i^{N_{\text{occ}}} \epsilon_i. \tag{2.99}$$

The perturbation, $\hat{H}'$, in MPPT is then the difference between the Hamiltonian for the exact

problem (Eq. 2.5) and the unperturbed Hamiltonian,

$$\begin{aligned}\hat{H}' &= \hat{H} - \hat{H}_0 \\ &= \sum_{m<n}^{N_{\text{elec}}} r_{mn}^{-1} - \sum_{n}^{N_{\text{elec}}} v^{\text{HF}}(n)\end{aligned} \tag{2.100}$$

i.e. the difference between the exact electron-electron interaction and the Hartree-Fock effective potential, where

$$v^{\text{HF}}(n) = \sum_i \hat{J}_i(n) - \hat{K}_i(n). \tag{2.101}$$

The MPPT partitioning of the Hamiltonian results in the size-extensive MPn methods, where n refers to the order of correction to the unperturbed energy (see Ref. 10 (pp.747–749) for a discussion of size-extensivity). These methods have seen wide adoption and development for the calculation of electron correlation energies in quantum chemistry.

The perturbed energy in Rayleigh-Schrödinger perturbation theory is non-variational (see Ref. 34 (p.21)) and convergence towards the exact solution to the Schödinger equation (Eq. 2.87) with increasing orders of correction is not guaranteed. For the Møller-Plesset partitioning of the Hamiltonian, it is well-established that the MPn energy does not always converge smoothly to the exact (fully perturbed) energy with increasing n and may in fact behave erratically at higher orders (see Ref. [46] and references therein). Despite the problematic convergence of the MPn series, the lower-order MPn methods, in particular MP2, typically significantly improve upon the Hartree-Fock energy (and other properties, see Ref. [10, ch.15]). The MP2 method has seen particularly widespread adoption and development because it offers an good compromise between accuracy and efficiency.

The MP2 energy is the lowest order correction to the unperturbed energy (Eq. 2.98) which improves upon the Hartree-Fock energy—the MP1 energy is equivalent to the Hartree-Fock energy, the first-order energy correcting the double-counting of the electron-electron repulsion interaction in the zeroth-order energy [7, p.351], i.e.

$$E_{\text{MP1}} = E_{\text{HF}} = E_0^{(0)} + E_0^{(1)}. \tag{2.102}$$

The expression for the second-order correction to the energy in MPPT can be derived from the general RSPT expression (Eq 2.96) by substituting the zeroth-order eigenfunctions, $\{|\Psi_\alpha^{(0)}\rangle\}$, and eigenvalues, $\{E_\alpha^{(0)}\}$, with the relevant quantities from MPPT. The zeroth-order wavefunction has already been chosen to be the Hartree-Fock ground-state, $|\Psi_0^{(0)}\rangle = |\Phi_0\rangle$, with the corresponding zeroth-order energy given by Eq. 2.99. The zeroth-order eigenfunctions in which the first-order wavefunction is expanded, $\{|\Psi_\alpha^{(0)}\rangle\}$, are the set of all possible excited Slater determinants in the MO basis (i.e. the same set of Slater determinants that would be used in an FCI calculation, Eq. 2.60).

In Eq. 2.96, the zeroth-order eigenfunctions only appear in the $\langle\Psi_\alpha^{(0)}|\hat{H}'|\Psi_\beta^{(0)}\rangle$ terms, and for the MPPT second-order energy, summation over zeroth-order eigenfunctions can be restricted to

only the doubly-excited states, $\{|\Phi_{ij}^{ab}\rangle\}$. For determinants which are more than doubly excited,

$$\langle\Phi_0|\hat{H}'|\Phi_{ij\ldots}^{ab\ldots}\rangle = 0 \tag{2.103}$$

since matrix elements over two-electron operators (such as $\hat{H}'$ in MPPT, Eq. 2.100) between Slater determinants differing by more than two spin-orbitals are always zero (see Ref. 7 (p.68)). Singly excited determinants are also excluded from the summation of zeroth-order eigenfunctions, since

$$\langle\Phi_0|\hat{H}'|\Phi_i^a\rangle = \langle\Phi_0|\hat{H}|\Phi_i^a\rangle - \langle\Phi_0|\hat{H}_0|\Phi_i^a\rangle = 0. \tag{2.104}$$

where (assuming canonical MOs and using Eq. 2.39),

$$\langle\Phi_0|\hat{H}|\Phi_i^a\rangle = \langle\Phi_0|\hat{H}_0|\Phi_i^a\rangle = \langle a|\hat{f}|i\rangle = \epsilon_i\langle i|a\rangle = 0 \tag{2.105}$$

and where $|i\rangle$ and $|a\rangle$ refer to (canonical) spin-orbitals χ_i and χ_a. The second-order energy in MPPT is then

$$E_0^{(2)} = \sum_{i<j}\sum_{a<b}\frac{\langle\Phi_0|\hat{H}'|\Phi_{ij}^{ab}\rangle\langle\Phi_{ij}^{ab}|\hat{H}'|\Phi_0\rangle}{E_0 - E_{ij,ab}}. \tag{2.106}$$

where $E_{ij,ab}$ is the zeroth-order energy of $|\Phi_{ij}^{ab}\rangle$.

The matrix elements over the perturbation simplify to two-electron integrals over spin-orbitals,

$$\begin{aligned}\langle\Phi_0|\hat{H}'|\Phi_{ij}^{ab}\rangle &= \sum_{m>n}\langle\Phi_0|r_{mn}^{-1}|\Phi_{ij}^{ab}\rangle - \sum_m\langle\Phi_0|v^{\mathrm{HF}}(m)|\Phi_{ij}^{ab}\rangle \\ &= \langle ij||ab\rangle\end{aligned} \tag{2.107}$$

where

$$\langle ij||ab\rangle = \langle ij|r_{12}^{-1}|ab\rangle - \langle ij|r_{12}^{-1}|ba\rangle. \tag{2.108}$$

Using Eqs. 2.97 and 2.99, it is clear that,

$$\hat{H}_0|\Phi_{ij}^{ab}\rangle = (E_0 - \epsilon_i - \epsilon_j + \epsilon_a + \epsilon_b)|\Phi_{ij}^{ab}\rangle, \tag{2.109}$$

and thus from Eq. 2.106 we obtain the well-known equation for the MP2 energy correction in terms of MO energies and integrals over spin-orbitals:

$$E_0^{(2)} = \sum_{i<j}\sum_{a<b}\frac{|\langle ij||ab\rangle|^2}{\epsilon_i + \epsilon_j - \epsilon_a - \epsilon_b}. \tag{2.110}$$

The preceding derivation of the second-order RSPT and MPPT energies was achieved by expanding the first-order wavefunction in a basis of zeroth-order eigenfunctions (Eq. 2.94). The first-order wavefunction and second-order energy correction may also be obtained by a variational minimization of the Hylleraas functional [47] (see also Refs. 8 (p.122), 34 (p.25), and 48 (p.124)):

$$E_2[\Psi] = 2\langle\Psi|\hat{H}'|\Psi_0^{(0)}\rangle + \langle\Psi|\hat{H}_0 - E_0^{(0)}|\Psi\rangle \tag{2.111}$$

where $|\Psi\rangle$ is a trial function approximating the first-order wavefunction, constrained to be orthogonal to the zeroth-order wavefunction, and $E_0^{(0)}$ is the lowest eigenvalue of $\hat{H}_0$ (additionally, we have assumed real zeroth- and first-order wavefunctions). E_2 represents an upper-bound to the second-order correction to the energy and approximations to $E_\alpha^{(2)}$ and $|\Psi_\alpha^{(1)}\rangle$ can be found simultaneously by minimization of $E_2[\Psi]$ with respect to $|\Psi\rangle$, i.e.

$$E_2[\Psi] \geq E_\alpha^{(2)}, \tag{2.112}$$

and

$$E_2\left[\Psi_\alpha^{(1)}\right] = E_\alpha^{(2)}, \tag{2.113}$$

where $E_\alpha^{(0)}$ is the lowest zeroth-order energy.

MP2 theory may be cast as a "pair theory", where the total correlation energy is decomposed into individual contributions from pairs of electrons in occupied spin-orbitals [7, p.272]:

$$E_{\text{corr}} = \sum_{i<j} e_{ij}. \tag{2.114}$$

That the second-order energy correction in MPPT separates into contributions for pairs of occupied orbitals is evident from Eq. 2.110, i.e.

$$E_0^{(2)} = \sum_{i<j} e_{ij}^{\text{MP2}} \tag{2.115}$$

where

$$e_{ij}^{\text{MP2}} = \sum_{a<b} \frac{|\langle ij||ab\rangle|^2}{\epsilon_i + \epsilon_j - \epsilon_a - \epsilon_b}. \tag{2.116}$$

The Hylleraas functional (Eq. 2.111) can be cast in terms of electron pairs —for the MPPT partitioning of the Hamiltonian the "Hylleraas pair functional" can be derived [49, 50], i.e.

$$e_{ij}[u_{ij}] = \langle u_{ij}|\hat{f}_1 + \hat{f}_2 - \epsilon_i - \epsilon_j|u_{ij}\rangle + 2\langle u_{ij}|r_{12}^{-1}|ij\rangle \tag{2.117}$$

where $|u_{ij}\rangle$ is the "pair function" for occupied orbitals i, j and where the antisymmetry of $|ij\rangle$ and $|u_{ij}\rangle$ is implicit. The contributions of each occupied orbital pair to the full MP2 correlation energy (Eq. 2.116) can therefore be individually optimized by minimization of this functional.

The pair function for canonical MP2 theory, where the first-order wavefunction is expanded in terms of doubly excited Slater determinants, is

$$|u_{ij}\rangle = \sum_{a<b} t_{ab}^{ij}|ab\rangle, \tag{2.118}$$

and the amplitudes t_{ab}^{ij} could be determined by minimization of Eq. 2.117. In fact, the form of these amplitudes is known, since they are effectively determined in the derivation of the MP2 energy correction (Eq. 2.110), i.e.

$$t_{ab}^{ij} = \frac{\langle ab||ij\rangle}{\epsilon_i + \epsilon_j - \epsilon_a - \epsilon_b}. \tag{2.119}$$

The approach of minimizing the Hylleraas pair functional to determine $|u_{ij}\rangle$ and e_{ij} is useful where alternative ansatzes for $|u_{ij}\rangle$ are used, for example in explicitly correlated MP2-F12 theory.

2.2.4 Explicitly correlated methods

The computational expense of an electronic structure method increases as a function of the size of the AO basis set, M. Without approximations, such as density fitting (section 2.4), the computational cost of the methods outlined in previous sections is at least $\mathcal{O}(M^4)$, due to the need to evaluate four-index, two-electron integrals (section 2.3). Methods which reach a given level of accuracy using a smaller basis set are therefore generally preferable in terms of reducing the computational cost. Unfortunately, the electronic structure methods described so far all suffer from slow convergence of the error in the correlation energy with respect to AO basis set size. To increase the accuracy of calculating the correlation energy with such methods requires a large increase in the size of the basis set and thus a significant increase in computational cost.

For the correlation-consistent cc-pVXZ basis sets [51–53], the formula

$$E_{\text{corr}}^{\text{limit}} \approx E_{\text{corr}}^{X} + AX^{-3} \tag{2.120}$$

allows extrapolation of the correlation energy for a given basis set E_{corr}^{X} to the basis set limit, E_{corr} [54,55]. This relationship implies that the error in the correlation energy, ΔE_{corr}, decays as $\mathcal{O}(X^{-3})$ for the cc-pVXZ series of basis sets. Since the number of basis functions in the cc-pVXZ series of basis sets increases as $M = \mathcal{O}(X^3)$ [10, p.310], the error in the correlation energy scales as $\Delta E_{\text{corr}} = \mathcal{O}(M^{-1})$. Given a computational expense, t, of $\mathcal{O}(M^4)$, the convergence of the error in the correlation energy with respect to computational cost is very poor: $\Delta E_{\text{corr}} = \mathcal{O}(t^{-1/4})$. This relationship between the error in the correlation energy and the computational cost is described in Ref. 56, with the authors concisely summarizing the problem as follows:

> the rewards in terms of accuracy for increasing computational effort are extremely meagre: a 10000-fold increase in computer resources gives only one order of magnitude improvement in accuracy.

The cost of improving the accuracy of the correlation energy may be reduced by improving the scaling of computational cost, t, with respect to the number of basis functions M, as in local methods (e.g. Ref. 57 and references therein). An alternative (and complementary) route to reducing the computational cost of achieving a given accuracy is to reduce the scaling of the error in the correlation energy, ΔE_{corr}, with respect to the size of the AO basis set—this is the approach taken in explicitly correlated methods.

It has long been known [58] that the poor scaling of ΔE_{corr} with respect to the size of AO basis exhibited by the electronic structure methods outlined earlier in this chapter is the result of poor description of the Coulomb hole (Eq. 2.56). These "conventional" methods express the N-electron wavefunction in terms of antisymmetrized products of atom-centred, one-electron basis functions (Slater determinants, Eq. 2.22). To accurately represent the Coulomb hole, an electron-centred feature, requires a very large number of these atom-centred basis functions, resulting in slow convergence of ΔE_{corr} with respect to size of the AO basis set. In addition,

the Slater determinant-type wavefunctions used in conventional methods are not well suited to representing the precise behaviour of the exact wavefunction at electron-electron coalescence (e.g. Eq. 2.57, for further details see Ref. 10 (pp.259–262) and 25). This suggests an approach to achieving improved convergence of ΔE_{corr} with respect to AO basis set size—improve the description of the Coulomb hole by the approximate wavefunction for a given AO basis set. The explicitly correlated methods achieve this by using an approximate wavefunction ansatz with explicit dependence on r_{12}.

In the early 20th century, Hylleraas, motivated by the slow convergence of CI-type approximate wavefunctions, pioneered the use of wavefunctions with explicit dependence on r_{12}, obtaining very accurate electronic energies for the helium atom with only a small expansion of the form,

$$\Psi(s,t,u) = \exp(-\alpha s) \sum_{k,l,m} c_{k,l,m} s^k t^{2l} u^m \tag{2.121}$$

with $s = r_1 + r_2$, $t = r_1 - r_2$ and the interelectronic distance, $u = r_{12}$ [59]. At this time, Slater also independently suggested the use of approximate wavefunctions with explicit dependence on r_{12}, having analysed the properties of the helium atom wavefunction [60] (see Ref. 61 for further historical context). Subsequent attempts were made to generalize the explicitly correlated approach to many-electron systems, including the transcorrelated method [62], explicitly correlated Gaussian geminals (notably the weak-orthogonality functional approach [63]), and Hylleraas-CI [64, 65] but all were hindered by the appearance of numerous complicated integrals over more than two electron coordinates, arising from the introduction of the "correlation factor" $f(r_{12})$.

It was not until the 1980s that the development of the R12 approach by Kutzelnigg and Klopper finally delivered a practical explicitly correlated approach for many-electron systems. This was made possible by the use of resolutions of the identity (RIs) to approximate the many-electron integrals arising from dependence on r_{12}—cutting the "Gordian knot" [66]. Insertion of a RI allows complicated many-electron integrals to be approximated as sums of products of simpler integral classes, e.g.

$$\begin{aligned} \langle ijm|r_{12}^{-1} f_{23}|mlk\rangle &\approx \langle ijm|r_{12}^{-1}\hat{X}_2 f_{23}|mlk\rangle \\ &\approx \sum_x \langle ij|r_{12}^{-1}|mx\rangle\langle mx|f_{12}|kl\rangle \end{aligned} \tag{2.122}$$

where $f_{23} \equiv f(r_{23})$ is the correlation factor and the approximate identity operator $\hat{X} = \sum_x |x\rangle\langle x| \approx \hat{1}$ projects onto some orthonormal basis, $\{|x\rangle\}$. If $\{|x\rangle\}$ is a complete orthonormal basis, then Eq. 2.122 becomes an equality.

The R12 approach was formulated with the philosophy that the "conventional" electronic structure methods are only really deficient in the region of electron coalescence, and therefore the explicit dependence on r_{12} should be introduced as a correction to this region. As a consequence, the established framework of the conventional methods, such as MPPT, coupled cluster theory and CI, can be readily adapted to benefit from the improved convergence offered by explicitly correlated methods. In the R12 approach, this is done by augmenting the conventional wavefunction with a small number of explicitly-correlated terms (i.e. terms featuring the

correlation factor $f(r_{12})$). For example, in his classic 1985 paper [66], Kutzelnigg added a single explicitly correlated reference function to the CI-type ground-state wavefunction for the helium atom, i.e.

$$|\Psi\rangle = \left(1 + \frac{1}{2}r_{12}\right)|\Phi_0\rangle + \sum_{I>0} c_I|\Phi_I\rangle \tag{2.123}$$

where $|\Phi_0\rangle$ is the single-determinant reference state, and $\{|\Phi_I\rangle\}$ are the normal excited states from the CI wavefunction (Eqs. 2.58 and 2.60).

In MP2 theory, this corresponds to modification of the pair-function (Eq. 2.118), i.e.

$$|u_{ij}\rangle = \sum_{a<b} t_{ab}^{ij}|ab\rangle + \hat{Q}_{12}f_{12}\sum_{k<l} t_{kl}^{ij}|kl\rangle, \tag{2.124}$$

where $\hat{Q}_{12}$ is the "strong orthogonality" projection operator, which projects onto the virtual orbital space, f_{12} is the correlation factor, and t_{kl}^{ij} are explicitly correlated amplitudes, which may be optimized by minimizing the Hylleraas pair functional (Eq. 2.117). In MP2-R12 theory, as first described by Kutzelnigg and Klopper [50,67,68], the correlation factor is linear ($f_{12} = r_{12}$). It was subsequently found that use of a non-linear correlation factor ($f_{12} \neq r_{12}$) had significant advantages [69]—this is MP2-F12 theory [70,71], which is central to the work on density fitting many-electron integrals in explicitly correlated methods presented in chapter 3.

It is now well known that a significant improvement in the calculated correlation energy for a given AO basis set may be obtained by using the R12/F12 variant of a conventional method. For example, in the case of the aug-cc-pVXZ basis family [51, 72] it can be expected that F12 variants of the MP2, CCSD and CCSD(T) methods return quintuple-ζ ($X = 5$) quality correlation energies using a triple-ζ ($X = T$) basis [73, 74]. These methods are attractive from a computational perspective, with the rapid convergence of the error in the energy with respect to basis set size mitigating the increased computational cost associated with the F12 correction. For some variants, e.g. CCSD(F12) theory [25,73,74], the computational scaling (section 2.2.6) of the explicitly correlated method is the same as its conventional counterpart.

For a review of recent developments in the R12/F12 methods and a description of the theoretical framework of MP2-F12 theory in the context of the work presented in this thesis, see chapter 3.

2.2.5 Density functional theory

A discourse on the treatment of electron correlation in quantum chemistry would be incomplete without mention of density-functional theory (DFT). However, since the theoretical underpinnings of DFT are not directly relevant to the research presented in this thesis, only a brief outline will be provided here. Further details are available from a variety of sources, including Refs. 12, 75 and 76.

In the past few decades, DFT has been developed into a powerful and versatile electronic structure method. The method is now one of the most popular and widely used techniques for the study of many-electron systems [76], primarily because it provides a computationally inexpensive means for treating electron correlation.

In the electronic structure methods discussed in previous sections, the Schrödinger equation

(Eq. 2.2) is solved approximately using approximations to the exact N-electron wavefunction, $\psi(\mathbf{x}_1, \cdots, \mathbf{x}_N)$, which is a complicated function depending on $3N$ spatial coordinates. The Hohenberg-Kohn theorems [77] state that there exists a functional for the energy, $E[\rho]$, which can be minimized with respect to the electron density, $\rho(\mathbf{r})$ (Eq. 2.54), to yield the exact ground-state energy and electron density of a many-electron system. In principle, an electronic structure method based on such a functional could eschew the complicated $\psi(\mathbf{r}_1, \cdots, \mathbf{r}_N)$ in favour of the much simpler $\rho(\mathbf{r})$, which depends only on three spatial coordinates.

DFT, as it is generally encountered today, is based upon the formalism described by Kohn and Sham in their seminal 1965 paper [78]. Kohn-Sham DFT (KS-DFT) greatly simplifies the problem of determining a total energy functional by assuming that the ground-state density of a many-electron system can be represented by a corresponding ground-state density for an auxiliary system of non-interacting electrons (i.e. independent electrons in a mean-field). In this case, many-body exchange and correlation effects can be consolidated into a single, relatively small, contribution, $E_{\text{xc}}[\rho]$, with the larger contributions from the non-interacting kinetic energy and classical Coulombic interaction of electrons expressed in terms of simple, known functionals [75, ch.7].

In the standard KS-DFT method, the approximate ground-state density and energy are calculated self-consistently, optimizing a set of Kohn-Sham (KS) orbitals using a procedure similar to the Hartree-Fock SCF method (section 2.1.1) and using these to construct an approximate density [75, ch.9]. An explicit functional form for $E_{\text{xc}}[\rho]$ is unknown, though many approximate forms have been proposed (see Refs. 75 (ch.8) and 76). The development of effective approximate exchange-correlation functionals has been, and continues to be an active area of research.

2.2.6 Computational cost and accuracy

In applying any method in computational science, it is important to consider the computational expense, and whether this is practical or appropriate. For the correlated electronic structure methods described in the preceding sections, this generally corresponds to considering what level of accuracy is appropriate to the problem, since more accurate calculations, are, in general, more computationally expensive.

Within the framework of methods such as configuration interaction, coupled cluster theory and Møller-Plesset perturbation theory there are two main ways in which accuracy may generally be improved:

- Increasing the size of the one-electron AO basis.
- Using a more sophisticated level of theory.

Increasing the size of the AO basis reduces the error from incompleteness in the one-electron basis. Where the AO basis is complete, the energy is said to be at the complete basis set (CBS) limit and any remaining discrepancy between the exact correlation energy and the calculated energy is the result of deficiency in the method. For CI and coupled cluster theory, increasing the sophistication of the theory generally corresponds to increasing the maximum excitation level

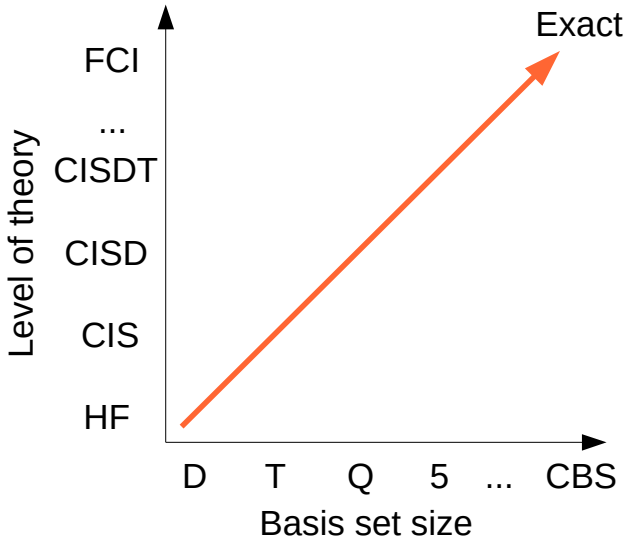

Figure 2.3: A schematic plot illustrating how increasing basis set size (here labelled in terms of the cardinal number X of the cc-pVXZ basis sets) and increasing sophistication of theory improve the result of a calculation towards the "exact" FCI CBS limit result. This plot is adapted from Ref. 12 (p.136), and is often referred to as a "Pople diagram", though it is only loosely based on Pople's original plot [79].

of the N-electron wavefunction, while for MPPT, this is achieved by increasing the maximum order of perturbation.

In the case of a method with a variationally bound energy, increasing the size of the AO basis or the sophistication of the theory can only improve the accuracy of the calculated energy with respect to the exact energy. For a non-variational method this is not guaranteed. For example, as mentioned in section 2.2.3, the non-variationally bound MPn energy does not always smoothly converge toward the exact energy for increasing orders of perturbation [46].

The convergence of calculated results to the "exact" FCI CBS limit is often illustrated using plots like Fig. 2.3, where it is understood that moving along either the x- or y-axis increases both the computational expense of the calculation and the accuracy of the result (relative to the FCI result). Fig. 2.3 is, at best, a general guide, and is perhaps misleading in its simplicity—the reality is far more opaque.

Selecting an appropriate method and basis set is made more complicated because the performance of a given method and basis set varies under different circumstances, i.e. a combination that is effective for one chemical system, may be problematic in another. In addition, there is a great diversity of approximations and corrections which may be applied within the various theoretical frameworks, and a very wide range of AO basis sets available. Selecting an appropriate level of theory and AO basis is therefore very complicated—often some trial and error is involved, particularly where the system being studied is new or unusual. This difficulty in selecting appropriate methods is somewhat alleviated by rigorous benchmarking methods for large datasets (e.g. Refs. 80 and 81).

An important theoretical indication of the computational cost of a method, alluded to in section 2.2.4, is the scaling of computational cost with respect to some measure of the extent of

a calculation. The phrase "extent of a calculation" is necessarily vague, since there are many parameters which affect the cost of a calculation, e.g. number of atoms, number of monomers, number of integration points, number of AO basis functions, number of excited determinants etc. The "computational cost" of a calculation is also intentionally vague, since this could also be measured in a variety of ways, e.g. processor cycles, number of CPU instructions, time to result.

For the correlated electronic structure methods described in this chapter, how the cost of a calculation scales with respect to the physical size of the system (e.g. number of atoms) and the number of basis functions is typically of greatest interest, since this indicates the practicality of applying a method/basis set to a given chemical system. Since the number of basis functions used typically scales linearly with the number of atoms, it is common to simply refer to a generic measure of "system size" N.

An electronic structure calculation involves multiple algorithmic steps, which may have different costs in terms of system size. The total computational cost, t, will be the sum of the costs of each step, i.e.

$$t(N) = \sum_{i}^{n_{\text{steps}}} c_i f_i(N) \tag{2.125}$$

where $f(N)$ is some non-negative function of system size. The coefficients c_i are generally not known, and may vary between implementations and for different inputs, so calculating t for a given case is non-trivial. We can, however, estimate the worst-case scenario by considering the asymptotic scaling—in the limit of large N, the term with highest order $f(N)$ will dominate. This is the asymptotic upper bound of the function $t(N)$ and can be compactly expressed using "big O notation":

$$t(N) = \mathcal{O}(g(N)) \tag{2.126}$$

i.e. there exist positive constants c and N_0 for which $0 \leq t(N) \leq cg(N)$ for all $N \geq N_0$ (see Ref. 82 (ch.3) for a more formal definition of asymptotic notation in computer science). The asymptotic upper bound scaling is often referred to as the "formal scaling" of a method in the computational chemistry literature.

In the small N regime (i.e. $N < N_0$) other terms in Eq. 2.125 may dominate. In comparing methods, which may scale differently with N, it is also worth considering the coefficient, or "prefactor", c, associated with the fastest growing term in Eq. 2.125. For example, if a lower scaling method has a relatively large prefactor compared to a higher scaling method, there may exist a range of N in which the higher scaling method has a lower computational cost (Fig. 2.4).

The formal scaling of performing a calculation using a given correlated electronic structure method is determined by the highest scaling step in the calculation, i.e. the step for which the computational cost increases most rapidly with N. When considering how to reduce the overall formal scaling of a method, it is therefore natural to focus on the highest scaling step. Significant decreases in the formal scaling of methods can be achieved by judicious use of approximations such as density fitting (see section 2.4) and local methods [83–85]. For example, while the formulation of MP2 theory outlined in section 2.2.3 has a formal scaling of $\mathcal{O}(N^5)$, the DF-LMP2 method, which combines density fitting and local approximations, scales as $\mathcal{O}(N)$ [57].

The true computational cost of a given calculation is dependent on many factors. The formal

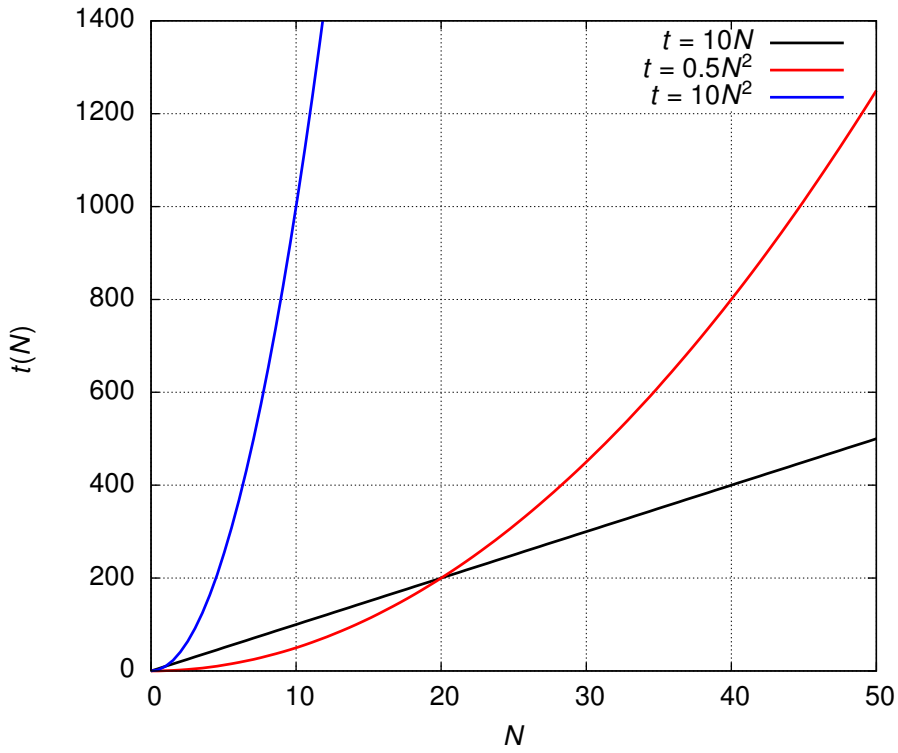

Figure 2.4: Plot demonstrating how a method with lower formal scaling, but larger prefactor (black line) may have a higher computational cost than a higher scaling method with smaller prefactor (red line) for a range of N. Where the prefactor for the higher scaling method is equal to or larger than that of the lower scaling method, there is no region in which the higher scaling method has a lower computational cost (for integer N, blue line).

scaling of the method with respect to system size and basis set size is an important determinant in performance, but other less-easily quantified factors are also present, including:

- The nature of the hardware/software on which the calculation is performed.
- The nature and configuration of compilers used to create the executable used to run the calculation.
- The efficiency of the software implementation of the theory and how appropriate this is for the hardware/software the calculation is being run on.
- The extent to which the calculation may be parallelized, and whether the implementation takes advantage of this.

This makes the *a priori* estimation of computational cost very difficult, though benchmarking can also provide useful information in this respect.

It is desirable for software to demonstrate the asymptotic scaling of the method it implements, and this can be verified for a given hardware/software environment by measuring the cost against system size. Empirical comparisons of different software implementations should be treated in the same way as physical experiments, with care taken to control the environment as far as it possible. In practice, this means ensuring that comparable hardware/software

configurations are used and that compilation is (where possible) done using the same compiler settings and optimizations.

2.2.7 Final comments

The description of correlated electronic structure methods in this section is intended only as an overview, providing necessary background theory for the work presented in this thesis. As alluded to previously, there is a great diversity of methods, with many associated variants and approximations—far too many to cover in this chapter! It is, however, worth mentioning some broad areas of study that have not yet been discussed.

Multi-reference methods

All the methods described in this section have been "single-reference" methods, meaning that they all use as starting point a single determinant reference state. In certain circumstances, a single Slater determinant reference is unable to capture even the qualitative electronic structure of a system, and thus is unsuitable as a starting point for a correlated calculation. This is typically the case where static correlation is important, i.e. where the wavefunction is best described by multiple degenerate or near-degenerate states. In such cases, methods which employ a multi-configuration reference state should be used. See the relevant sections in Refs. 10 and 12 for further details.

Combining methods

The development of schemes for the combination of electronic structure methods (and non-quantum methods) is an area of great theoretical interest. A significant motivator in this respect is the computational cost associated with high-accuracy methods—for methods with a relatively high formal scaling, calculations become prohibitively costly for larger system sizes. Schemes for combining methods allow more accurate (and costly) methods to be applied in a more restricted sense, correcting less accurate, less computationally demanding methods.

The composite methods combine results from different levels of theory (and sometimes empirical corrections) in practical schemes for the calculation of a variety of chemical properties. For an overview of these methods, see Ref. 2. In contrast, embedding methods subdivide a system into interacting spatial subregions in which different methods are applied. This allows more accurate methods to be applied only in regions where high accuracy is required, while other regions are treated more approximately. See Ref. 86 for a review of quantum chemical embedding strategies.

2.3 Molecular integrals

All the electronic structure methods described in sections 2.1.1 and 2.2 involve the evaluation of integrals over one-electron basis functions. These methods share a common theoretical framework, where approximate solutions to the Schrödinger equation are constructed from one-electron molecular orbitals (MOs), which are generally linear combinations of one-electron

atomic orbitals (AOs), with coefficients optimized in a SCF calculation. The evaluation of integrals over these basis functions—"molecular integrals"—can be considered separately to the methods themselves, since a general scheme can be used for evaluation of the many integral classes required across the various methods.

In this section, we will consider the key elements of one such general molecular integral evaluation scheme, which is relevant in the work presented in chapter 4. This scheme is based upon the recurrence relation approach originally developed by Obara and Saika [1]. Some alternative schemes will also be mentioned, some of which share features with the Obara-Saika approach.

2.3.1 Gaussian basis sets

In preceding sections, the AO basis sets used to expand the MOs (Eq. 2.43) were considered only in a general sense. We will now examine the precise nature of the AO basis sets generally used in quantum chemical calculations, as this is central to the subject of molecular integral evaluation.

Ref. 10 (ch.6), lists three properties that an ideal one-electron AO basis should have:

- It should be possible to systematically extend the basis set toward completeness.
- Molecular electronic structure should be well described with only a few basis functions, i.e. rapid convergence towards realistic solutions should be seen with increasing basis set size.
- The functional form of the basis functions should be suitable for use in computation, particularly for the evaluation of molecular integrals.

Essentially, a good AO basis set should efficiently capture the physical problem in a systematically improvable manner, while facilitating efficient computation.

Basis sets constructed from contracted Gaussian-type orbitals (GTOs) have become the *de facto* standard for molecular electronic structure calculations because they offer a good compromise in terms of satisfying the above conditions. The contracted GTOs have the general form:

$$\varphi_a(\mathbf{r};\ell_a,m_a,\mathbf{A}) = \sum_{k=1}^{K_a} d_k G(\mathbf{r};\zeta_k,\ell_a,m_a,\mathbf{A}) \tag{2.127}$$

with centre $\mathbf{A}$, "degree of contraction" K_a, contraction coefficients d_k, and angular momentum quantum numbers ℓ_a and m_a. $G(\mathbf{r};\zeta_k,\ell_a,m_a,\mathbf{A})$ is a spherical-harmonic Gaussian function,

$$G(\mathbf{r};\zeta,\ell,m,\mathbf{A}) = N_{\zeta\ell} S_{\ell m}(\mathbf{r}_A)\exp(-\zeta|\mathbf{r}_A|^2) \tag{2.128}$$

where $S_{\ell m}(\mathbf{r})$ is a real solid harmonic (as defined in Ref. 10 (pp.209–210) and appendix A), $N_{\zeta\ell}$ is a normalization constant and $\mathbf{r}_A = (r_x - A_x, r_y - A_y, r_z - A_z)$.

We will soon see that for the evaluation of molecular integrals, it is advantageous to express the spherical-harmonic Gaussians in terms of primitive Cartesian Gaussian functions,

$$g(\mathbf{r};\zeta,\mathbf{a},\mathbf{A}) = X_{\mathbf{a}}(\mathbf{r}_A)\exp(-\zeta|\mathbf{r}_A|^2) \tag{2.129}$$

with angular momentum vector $\mathbf{a} = (a_x, a_y, a_z)$, and

$$X_{\mathbf{a}}(\mathbf{r}_A) = (r_x - A_x)^{a_x}(r_y - A_y)^{a_y}(r_z - A_z)^{a_z}, \tag{2.130}$$

describing the angular part of the function. The solid harmonics can be expressed in terms of linear combinations of $X_{\mathbf{a}}(\mathbf{r})$:[1]

$$S_{\ell m}(\mathbf{r}_A) = N^S_{\ell m} \sum_{\mathbf{a}} C^{\ell m}_{\mathbf{a}} X_{\mathbf{a}}(\mathbf{r}_A) \tag{2.131}$$

where the summation runs over all Cartesian components $\mathbf{a}$ for which $|\mathbf{a}| = \ell$. The relationship between the spherical-harmonic and primitive Cartesian Gaussians is thus

$$G(\mathbf{r}; \zeta, \ell, m, \mathbf{A}) = N_{\zeta\ell} N^S_{\ell m} \sum_{\mathbf{a}} C^{\ell m}_{\mathbf{a}} g(\mathbf{r}; \zeta, \mathbf{a}, \mathbf{A}). \tag{2.132}$$

The contracted GTOs can therefore be re-expressed in terms of the primitive Cartesian Gaussians, i.e.

$$\begin{aligned} \varphi_a(\mathbf{r}; \ell_a, m_a, \mathbf{A}) &= N^S_{\ell_a m_a} \sum_{k}^{K_a} d_k N_{\zeta_k \ell_a} \sum_{\mathbf{a}} C^{\ell_a m_a}_{\mathbf{a}} g(\mathbf{r}; \zeta_k, \mathbf{a}, \mathbf{A}) \\ &= N^S_{\ell_a m_a} \sum_{\mathbf{a}} C^{\ell_a m_a}_{\mathbf{a}} \phi_a(\mathbf{r}; \mathbf{a}, \mathbf{A}) \end{aligned} \tag{2.133}$$

where $\phi_a(\mathbf{r}; \mathbf{a}, \mathbf{A})$ is a contracted Cartesian GTO.

The Cartesian GTOs may be used in electronic structure calculations directly, without transformation to the spherical-harmonic representation. However, in practice, the spherical-harmonic representation is typically used. One advantage of this approach is that it reduces the number of angular components, and thus the number of molecular integrals for a given value of ℓ. For total angular momentum ℓ, there are

$$n_{\text{cart}} = (\ell + 1)(\ell + 2)/2 \tag{2.134}$$

Cartesian components, while there are

$$n_{\text{sph}} = 2\ell + 1 \tag{2.135}$$

spherical components—for $\ell > 1$, the number of spherical components is always smaller than the number of Cartesian components.

Before the widespread adoption of contracted GTO-type basis sets, the Slater-type orbitals (STO) [87] were often used as AO basis functions in quantum chemical calculations. The spherical STOs have the general functional form

$$\varphi^{\text{STO}}_a(\mathbf{r}; \ell, m, \zeta, \mathbf{A}) = \mathcal{N}_{lm}(\theta_A, \varphi_A, \zeta) r^{\ell}_A \exp(-\zeta r_A) \tag{2.136}$$

[1] In appendix A, a more detailed definition of this transformation is provided, adapted from Ref. 10 (pp.214–215).

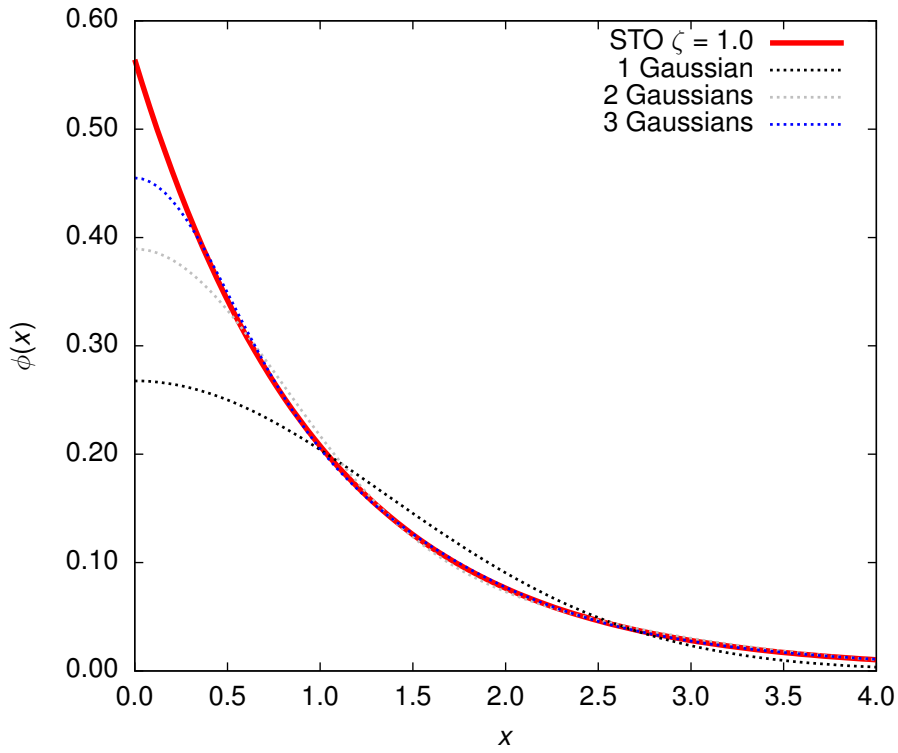

Figure 2.5: Plot demonstrating how linear combinations of Gaussian functions can be optimized to be more physically representative than a single Gaussian function. In this case, linear combinations of Gaussian functions are least-squares fitted to the more physically realistic s-type STO ($\zeta = 1.0$). While one Gaussian is a relatively poor representation, linear combinations of two and three Gaussian functions significantly improve on this. Figure adapted from Ref. 7 (pp.152–159).

where $\mathcal{N}_{\ell m}(\theta, \varphi, \zeta)$ contains the normalization and angular component of the function (details of which can be found in Ref. 10 (ch.6)) and $\mathbf{r}_A = (r_A, \theta_A, \phi_A)$. This functional form is well-suited for representing molecular electronic structure, in particular because it exhibits a cusp at electron-nucleus coalescence [7, pp.152–159]. Atom-centred Gaussian functions are inferior in this respect—individual Gaussian functions have a gradient of zero at the nucleus. The use of contracted GTOs mitigates this issue somewhat, since a linear combination of Gaussian functions can be optimized to provide a better physical representation (see Fig. 2.5 for example).

While several Gaussian functions are required to provide a physical representation comparable to a single STO, the contracted GTOs offer significant computational benefits compared to the STOs. These benefits were recognized by Boys, who in 1950 [88] demonstrated how molecular integral evaluation could be greatly facilitated by use of a Gaussian basis. There are two principal reasons for this. First, unlike the STOs, the Cartesian Gaussian functions easily factorize into Cartesian components, i.e.

$$g(\mathbf{r}; \zeta, \mathbf{a}, \mathbf{A}) = \prod_{i=x,y,z} r_{Ai}^{a_i} \exp(-\zeta r_{Ai}^2). \tag{2.137}$$

As a consequence, many of the integrals that arise in electronic structure methods can also be

factorized in this way, simplifying analytic integration, and the derivation of recurrence relations (as we shall see in section 2.3.3). Second, the product of two Gaussian functions on different centres can be expressed as a linear combination of Gaussians on a single centre using the "Gaussian product theorem" (GPT) [88], i.e.

$$g(\mathbf{r};\zeta_a,\mathbf{a},\mathbf{A})g(\mathbf{r};\zeta_b,\mathbf{b},\mathbf{B}) = \sum_{\mathbf{p}} T_{\mathbf{p}}^{\mathbf{ab}} g(\mathbf{r};\zeta,\mathbf{p},\mathbf{P}), \tag{2.138}$$

where $\zeta = \zeta_a + \zeta_b$, $\mathbf{p}$ runs from $(0,0,0)$ to $(a_x + b_x, a_y + b_y, a_z + b_z)$, and

$$\mathbf{P} = \frac{\zeta_a \mathbf{A} + \zeta_b \mathbf{B}}{\zeta}. \tag{2.139}$$

A full derivation of the transformation coefficients $T_{\mathbf{p}}^{\mathbf{ab}}$ for arbitrary $\mathbf{p}$ is presented in Ref. 89—see appendix B for a reproduction of the final result. This allows molecular integrals to be further simplified wherever there is a product of Cartesian Gaussian functions with the same electron coordinate.

The use of contracted GTOs as the AO basis raises a question—how can we optimize the contraction coefficients and exponents of the individual Gaussians? Ideally, the contracted GTOs should be optimized in a way which minimizes the number of basis functions required to obtain a given level of accuracy in our electronic structure calculations, since this will also minimize the computational cost of molecular integral evaluation. When a molecular calculation is performed, the pre-optimized AO basis functions for each atom are combined into a molecular AO basis set, and this is the set used in the electronic structure calculation. As a consequence, per-atom AO basis sets represent a compromise between the various factors that affect the basis set requirements of a given calculation, e.g.

- The electronic structure method used.
- The molecular environment.
- The quantities being calculated.

The selection of an appropriate AO basis set can be a significant factor in the quality of the result obtained from a calculation, and like the selection of an appropriate methods, may involve some trial-and-error. A typical electronic structure software package will offer a large range of Gaussian basis sets, optimized for different uses and environments. We will now briefly review some of the general features of these basis sets, as relevant to the work presented in this thesis. For more detailed accounts of the construction of various types of Gaussian basis sets, see Refs. 10 (ch.8) and 90.

A basis set which has only enough basis functions such that each electron of the neutral atom can occupy an atomic orbital of the appropriate angular momentum is termed a "minimal" basis set. For a hydrogen or helium atom, only a single s-type basis function is needed, while for an atom in the second row of the periodic table, AO basis functions corresponding to the $1s$, $2s$ and x, y, and z components of the $2p$ shell are required. The STO-nG basis sets [91] (see Fig. 2.5), are one such family of minimal AO basis sets. In these basis sets, contracted GTOs

comprised of n Gaussian functions are least-squares fitted to STOs with exponents optimized for atoms in molecular environments (see Refs. 10 (pp.288–291) and 91 for details).

For electronic structure calculations on molecules, minimal basis sets are somewhat inflexible and increasing the number of basis functions can significantly improve accuracy. Doubling the number of basis functions in a minimal basis set results in a "double-ζ" (DZ) basis set, where, for example, an atom in the second row the periodic table would have four s-type functions and two p-type shells, each with x, y and z components. If only the number of basis functions corresponding to atomic valence orbitals is doubled, then the basis set is "valence double-ζ" (VDZ). Triple-ζ (TZ), quadruple-ζ (QZ) and quintuple-ζ (5Z) basis sets may be constructed using the same approach.

A popular family of basis sets of this type are the "correlation consistent polarized" sets, denoted cc-pVXZ ($X = \mathrm{D}, \mathrm{T}, \mathrm{Q}, 5 \ldots$) [51–53]. The cc-pV$X$Z basis sets were designed to compactly describe electron correlation effects, adding diffuse "correlating" functions to the standard VXZ sets, intended to provide a more flexible representation of the virtual MOs. In these basis sets, the VXZ set of contracted basis functions is optimized to minimize the energy of an atomic Hartree-Fock calculation, while an additional set of uncontracted correlating functions is optimized in a correlated calculation. The cc-pVXZ family is "correlation consistent" in the sense that the group of correlating functions added when to the correlating set when X is incremented each contribute comparable amounts to the correlation energy. An attractive feature of the cc-pVXZ family of basis sets is that the electron correlation energy has been shown to converge smoothly to the complete basis set limit with increasing X (see Eq. 2.120 and Refs. 54 and 55). This systematic behaviour with increasing X is particularly useful when studying the behaviour of a method with basis set size, or angular momentum in the basis set (as in the work presented in chapter 3).

2.3.2 Molecular integral notation

To effectively discuss molecular integrals and methods for their evaluation, it is helpful to first establish some basic notation. The notation outlined here will be used throughout the following sections, and later chapters when discussing molecular integrals.

Molecular integrals over one-electron basis functions will be expressed using standard bra-ket [9] and Mulliken [12, p.96] notation, e.g.

$$\langle ij|r_{12}^{-1}|kl\rangle \equiv (ik|r_{12}^{-1}|jl) = \int \mathrm{d}\mathbf{r}_1 \mathrm{d}\mathbf{r}_2 \psi_i^*(\mathbf{r}_1)\psi_j^*(\mathbf{r}_2) r_{12}^{-1} \psi_k(\mathbf{r}_1)\psi_l(\mathbf{r}_2) \tag{2.140}$$

There is some scope for ambiguity in this notation, particularly for more complicated integrals featuring multiple operators, and integration over many electron coordinates. For this reason, full definitions of molecular integral classes will be provided in cases where ambiguity may arise.

Primitive Cartesian Gaussian integral indexes will be labelled $\mathbf{a}, \mathbf{b}, \mathbf{c}, \ldots$, where the index label indicates the angular momentum and the centre and exponent are implied, i.e.

$$|\mathbf{a}) \equiv g(\mathbf{r}; \zeta_a, \mathbf{a}, \mathbf{A}). \tag{2.141}$$

Where necessary, uncontracted spherical-harmonic Gaussian indexes will be distinguished by

the use of a different typeface, i.e. $\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots$

$$|\mathbf{a}) \equiv G(\mathbf{r}; \zeta_a, \ell_a, m_a, \mathbf{A}), \tag{2.142}$$

where the normalization (radial and angular) of primitive spherical Gaussian is not specified (see Eq. 2.132). Addition or subtraction of angular momentum from a primitive Cartesian Gaussian function will be indicated as follows:

$$|\mathbf{a} + \mathbf{1}_i) \equiv g(\mathbf{r}; \zeta_a, \mathbf{a} + \mathbf{1}_i, \mathbf{A}), \tag{2.143}$$

$$|\mathbf{a} - \mathbf{1}_i) \equiv g(\mathbf{r}; \zeta_a, \mathbf{a} - \mathbf{1}_i, \mathbf{A}), \tag{2.144}$$

where $\mathbf{1}_i$ is a vector with one unit of angular momentum in the ith direction, e.g. $\mathbf{1}_x = (1, 0, 0)$. For primitive Cartesian Gaussian indexes with zero angular momentum, we will use the notation

$$|\mathbf{0}_A) \equiv g(\mathbf{r}; \zeta_a, \mathbf{0}, \mathbf{A}), \tag{2.145}$$

where the subscript A may be omitted when unnecessary.

Contracted GTO indexes will be generally labelled $\mathrm{a}, \mathrm{b}, \mathrm{c}, \ldots$, and where it is important to distinguish between Cartesian and spherical-harmonic representations, the spherical-harmonic contracted GTO indexes will be labelled using a different typeface, $\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots$, i.e.

$$|\mathrm{a}) \equiv \phi_a(\mathbf{r}; \mathbf{a}, \mathbf{A}), \tag{2.146}$$

$$|\mathsf{a}) \equiv \varphi_a(\mathbf{r}; \ell_a, m_a, \mathbf{A}). \tag{2.147}$$

In this section, the molecular orbital integral indexes will be labelled $p, q, r, \ldots$, i.e.

$$|p) \equiv \psi_p(\mathbf{r}). \tag{2.148}$$

In other parts of this thesis, where the distinction between occupied and virtual orbitals is necessary, the occupied molecular orbital indexes will be labelled $i, j, m, \ldots$, and the virtual molecular orbital indexes $a, b, c, \ldots$ (this is distinguished from the contracted GTO indexes by being italic).

As an example, consider the two-index overlap integral: The primitive integral is denoted

$$(\mathbf{a}|\mathbf{b}) = \int \mathrm{d}\mathbf{r}\, g(\mathbf{r}; \zeta_a, \mathbf{a}, \mathbf{A}) g(\mathbf{r}; \zeta_b, \mathbf{b}, \mathbf{B}), \tag{2.149}$$

while the contracted integral has the form

$$\begin{aligned} (\mathrm{a}|\mathrm{b}) &= \int \mathrm{d}\mathbf{r}\, \varphi(\mathbf{r}; \mathbf{a}, \mathbf{A}) \varphi(\mathbf{r}; \mathbf{b}, \mathbf{B}), \\ &= \sum_m^{K_a} \sum_n^{K_b} d_m d_n (\mathbf{a}_m|\mathbf{b}_n) \end{aligned} \tag{2.150}$$

and the MO integral is

$$(p|q) = \int \mathrm{d}\mathbf{r}\, \psi_p(\mathbf{r}) \psi_q(\mathbf{r}). \tag{2.151}$$

Since all the quantities considered in the following discussion of molecular integrals are real-valued, the complex conjugates featuring in standard bra-ket notation can be safely ignored.

2.3.3 Recurrence relations

The use of primitive Cartesian Gaussian functions as the most basic one-electron function offers significant computational benefits for molecular integral evaluation, for the reasons outlined in section 2.3.1. However, the evaluation of these primitive integrals for arbitrary angular momentum still presents a formidable challenge. A naive approach to this problem would be to derive expressions for every combination of angular momentum vector required, as suggested by Boys [88] (see Ref. 92 for an example of this strategy in use). This approach is somewhat impractical, requiring the derivation and tabulation of numerous complicated expressions.

Over the decades since the publication of Boys's paper on the use of Gaussian-type basis functions [88], several efficient schemes for the evaluation of molecular integrals over GTOs have been developed. These include, the Pople-Hehre algorithm [93], the Dupuis-Rys-King algorithm [94–96] and McMurchie-Davidson algorithm [97] (see also Refs. 98 and 99). In the work presented in later chapters (particularly chapter 4), we will be solely concerned with one particular approach—the Obara-Saika scheme [1].

In 1986, Obara and Saika published a general molecular integral evaluation scheme which utilized recurrence relations for incrementing angular momentum in integral indexes [1]. At this time, similar approaches were already in use—earlier work by Schlegel had used a related scheme to efficiently calculate of the derivatives of two-electron integrals [100], and the older McMurchie-Davidson scheme [97] used recurrence relations for Hermite Gaussians. However, the Obara-Saika scheme represented a significant advance, offering a general recursive approach for molecular integral evaluation where recurrence relations were expressed directly using primitive Cartesian Gaussian functions.

Before we consider some examples of Obara-Saika-type recurrence relations, it is worth mentioning some key relations used in their derivation. Firstly, the derivative of a primitive Cartesian Gaussian function with respect to its centre:

$$\frac{\partial}{\partial A_i}|\mathbf{a}) = 2\zeta_a|\mathbf{a}+\mathbf{1}_i) - a_i|\mathbf{a}-\mathbf{1}_i). \tag{2.152}$$

Taking the derivative of a primitive Cartesian Gaussian with respect to its centre results in a combination of primitive Cartesian Gaussians with angular momentum incremented and decremented, and this property is pivotal in deriving recurrence relations which change angular momentum on primitive Cartesian Gaussian indexes. Second, many integrals exhibit the property of "translational invariance"—this is often useful in the derivation of recurrence relations:

$$\frac{\partial}{\partial A_i}(\mathbf{abc}\cdots) + \frac{\partial}{\partial B_i}(\mathbf{abc}\cdots) + \frac{\partial}{\partial C_i}(\mathbf{abc}\cdots) + \cdots = 0, \tag{2.153}$$

where $(\mathbf{abc}\cdots)$ is an arbitrary N index integral. A translationally invariant integral is unchanged when all centres are translated by the same spatial vector. Any operators present in such an integral must also be translationally invariant. Finally, for integrals featuring the r_{12}^{-1}

operator, the integral transform

$$\begin{aligned} r_{12}^{-1} &= \frac{2}{\pi^{1/2}} \int_0^\infty \mathrm{d}u \, \exp(-|\mathbf{r}_1 - \mathbf{r}_2|^2 u^2), \\ &= \frac{2}{\pi^{1/2}} \int_0^\infty \mathrm{d}u \, g(\mathbf{r}_1; u^2, \mathbf{0}, \mathbf{r}_2), \end{aligned} \tag{2.154}$$

is useful, since, unlike r_{12}^{-1}, the resulting Gaussian geminal $g_{12}^{u^2} \equiv g(\mathbf{r}_1; u^2, \mathbf{0}, \mathbf{r}_2)$ factorizes into the Cartesian directions.

Let us consider again the example of the primitive two-index overlap integrals (Eq. 2.149). The expression for the zero-angular-momentum case is

$$(\mathbf{0}_A|\mathbf{0}_B) = \left(\frac{\pi}{\zeta}\right)^{3/2} \exp(-\xi|\mathbf{A} - \mathbf{B}|^2), \tag{2.155}$$

with

$$\xi = \frac{\zeta_a \zeta_b}{\zeta_a + \zeta_b}. \tag{2.156}$$

An Obara-Saika-type recurrence relation for incrementing angular momentum in $|\mathbf{a})$ is

$$(\mathbf{a} + \mathbf{1}_i|\mathbf{b}) = \mathbf{PA}_i(\mathbf{a}|\mathbf{b}) + \frac{a_i}{2\zeta}(\mathbf{a} - \mathbf{1}_i|\mathbf{b}) + \frac{b_i}{2\zeta}(\mathbf{a}|\mathbf{b} - \mathbf{1}_i) \tag{2.157}$$

where $\mathbf{PA}_i \equiv P_i - A_i$ (see the appendix in Ref 1). Starting with Eq. 2.155, it is possible to use Eq. 2.157 to build up angular momentum in $|\mathbf{a})$ to an arbitary value. The symmetry of the integral class means that the same recurrence relation can be used to increment angular momentum in $|\mathbf{b})$, by permuting the integral indexes.

As an example, we will now derive Eq. 2.157 using the key relations mentioned earlier (Eqs. 2.152 to 2.154). We start by recognizing that $(\mathbf{a}|\mathbf{b})$ is translationally invariant (Eq. 2.153),

$$\frac{\partial}{\partial A_i}(\mathbf{a}|\mathbf{b}) + \frac{\partial}{\partial B_i}(\mathbf{a}|\mathbf{b}) = 0. \tag{2.158}$$

Applying the differential operators yields

$$2\zeta_a(\mathbf{a} + \mathbf{1}_i|\mathbf{b}) - a_i(\mathbf{a} - \mathbf{1}_i|\mathbf{b}) + 2\zeta_b(\mathbf{a}|\mathbf{b} + \mathbf{1}_i) - b_i(\mathbf{a}|\mathbf{b} - \mathbf{1}_i) = 0 \tag{2.159}$$

which can be rearranged to obtain

$$(\mathbf{a} + \mathbf{1}_i|\mathbf{b}) = -\frac{\zeta_b}{\zeta_a}(\mathbf{a}|\mathbf{b} + \mathbf{1}_i) + \frac{a_i}{2\zeta_a}(\mathbf{a} - \mathbf{1}_i|\mathbf{b}) + \frac{b_i}{2\zeta_a}(\mathbf{a}|\mathbf{b} - \mathbf{1}_i). \tag{2.160}$$

The relation,

$$(r_i - B_i) = (r_i - A_i) + (A_i - B_i) \tag{2.161}$$

naturally leads to the following property of products of primitive Cartesian Gaussians:

$$|\mathbf{a}(\mathbf{b} + \mathbf{1}_i)) = |(\mathbf{a} + \mathbf{1}_i)\mathbf{b}) + \mathbf{AB}_i|\mathbf{ab}), \tag{2.162}$$

which also applies to the two-index overlap integrals, i.e.

$$(\mathbf{a}|\mathbf{b}+\mathbf{1}_i) = (\mathbf{a}+\mathbf{1}_i|\mathbf{b}) + \mathbf{AB}_i(\mathbf{a}|\mathbf{b}). \tag{2.163}$$

Substituting Eq. 2.163 into Eq. 2.160 yields:

$$(1+\frac{\zeta_b}{\zeta_a})(\mathbf{a}+\mathbf{1}_i|\mathbf{b}) = -\frac{\zeta_b}{\zeta_a}\mathbf{AB}_i(\mathbf{a}|\mathbf{b}) + \frac{a_i}{2\zeta_a}(\mathbf{a}-\mathbf{1}_i|\mathbf{b}) + \frac{b_i}{2\zeta_a}(\mathbf{a}|\mathbf{b}-\mathbf{1}_i). \tag{2.164}$$

Using the relations

$$(1+\frac{\zeta_b}{\zeta_a}) = \frac{\zeta_a+\zeta_b}{\zeta_a}, \qquad \text{and} \qquad \mathbf{PA}_i = -\frac{\zeta_b}{\zeta_a+\zeta_b}\mathbf{AB}_i,$$

Eq. 2.164 can be rearranged to

$$(\mathbf{a}+\mathbf{1}_i|\mathbf{b}) = \mathbf{PA}_i(\mathbf{a}|\mathbf{b}) + \frac{a_i}{2\zeta}(\mathbf{a}-\mathbf{1}_i|\mathbf{b}) + \frac{b_i}{2\zeta}(\mathbf{a}|\mathbf{b}-\mathbf{1}_i), \tag{2.165}$$

which is the recurrence relation presented earlier (Eq. 2.157). The above derivation can also be found in Ref. 10 (pp.345–346). More complicated recurrence relations may be derived in a similar fashion.

In their 1986 paper [1], Obara and Saika outlined the derivation of recurrence relations for the three-index overlap integrals,

$$(\mathbf{a}|\mathbf{b}|\mathbf{c}) = \int \mathrm{d}\mathbf{r} g(\mathbf{r};\zeta_a,\mathbf{a},\mathbf{A})g(\mathbf{r};\zeta_b,\mathbf{b},\mathbf{B})g(\mathbf{r};\zeta_c,\mathbf{c},\mathbf{C}), \tag{2.166}$$

and the four-index electron repulsion integrals (ERIs):

$$(\mathbf{ab}|r_{12}^{-1}|\mathbf{cd}) = \int \mathrm{d}\mathbf{r}_1\mathrm{d}\mathbf{r}_2\, g_a(\mathbf{r}_1)g_b(\mathbf{r}_1)r_{12}^{-1}g_c(\mathbf{r}_2)g_d(\mathbf{r}_2). \tag{2.167}$$

where we have introduced the shorthand notation $g_a(\mathbf{r}_1) \equiv g(\mathbf{r}_1;\zeta_a,\mathbf{a},\mathbf{A})$. The integral transform of r_{12}^{-1} (Eq. 2.154) is key to the derivation of the ERI recurrence relations, allowing the recurrence relations for the three-index overlap integrals to be reused, since

$$\begin{aligned}(\mathbf{ab}|r_{12}^{-1}|\mathbf{cd}) &= \frac{2}{\pi^{1/2}}\int_0^\infty \mathrm{d}u\,(\mathbf{ab}|g_{12}^{u^2}|\mathbf{cd}) \\ &= \frac{2}{\pi^{1/2}}\int_0^\infty \mathrm{d}u \int \mathrm{d}\mathbf{r}_2\, g_c(\mathbf{r}_2)g_d(\mathbf{r}_2)(\mathbf{a}|\mathbf{0}_{\mathbf{r}_2}|\mathbf{b})\end{aligned} \tag{2.168}$$

where $|\mathbf{0}_{\mathbf{r}_2}) \equiv g(\mathbf{r}_1;u^2,\mathbf{0},\mathbf{r}_2)$. The integration over u introduces some additional complexity, since terms with explicit dependence on u arise, preventing the derivation of a recurrence relation directly in terms of the $(\mathbf{ab}|r_{12}^{-1}|\mathbf{cd})$ integrals. Obara and Saika's elegant solution to this problem was to express the recurrence in terms of an an auxiliary integral,

$$(\mathbf{ab}|\mathbf{cd})^{(m)} = \frac{2}{\pi^{1/2}}\int_0^\infty \mathrm{d}u \left(\frac{u^2}{\rho+u^2}\right)^m (\mathbf{ab}|g_{12}^{u^2}|\mathbf{cd}) \tag{2.169}$$

with "auxiliary index" m, $g_{12}^{u^2} \equiv g(\mathbf{r}_1; u^2, \mathbf{0}, \mathbf{r}_2)$, and

$$\rho = \frac{\zeta_a \zeta_b}{\zeta_a + \zeta_b}. \tag{2.170}$$

Where $m = 0$, the "true" ERIs are obtained, i.e.

$$(\mathbf{ab}|r_{12}^{-1}|\mathbf{cd}) = (\mathbf{ab}|\mathbf{cd})^{(0)}. \tag{2.171}$$

See appendix C for full reproductions of the ERI recurrence relations and zero-angular-momentum case from Ref. 1.

To illustrate the implications of the use of auxiliary indexes, we will consider the simpler two-index Coulomb integrals $(\mathbf{a}|r_{12}^{-1}|\mathbf{b})$. The zero-angular-momentum expression and recurrence relation for this class of integrals can be derived from the corresponding expressions for the four-index ERIs (see appendix C) by setting the exponents $\zeta_c = \zeta_d = 0$. The zero-angular-momentum expression is

$$(\mathbf{0}_A|\mathbf{0}_B)^{(m)} = 2\pi^{5/2}\zeta^{-1/2}(\zeta_a\zeta_b)^{-1}F_m(T), \tag{2.172}$$

where $F_m(T)$ is the Boys function, with argument $T = \xi|\mathbf{A}-\mathbf{B}|^2$ (the Boys function is described in section 2.3.4). The recurrence relation for the two-index Coulomb integrals is

$$\begin{aligned}(\mathbf{a}+\mathbf{1}_i|\mathbf{b})^{(m)} &= \mathbf{PA}_i(\mathbf{a}|\mathbf{b})^{(m+1)} \\ &+ \frac{a_i}{2\zeta_a}\left((\mathbf{a}-\mathbf{1}_i|\mathbf{b})^{(m)} - \frac{\zeta_b}{\zeta}(\mathbf{a}-\mathbf{1}_i|\mathbf{b})^{(m+1)}\right) + \frac{b_i}{2\zeta}(\mathbf{a}|\mathbf{b}-\mathbf{1}_i)^{(m+1)}.\end{aligned} \tag{2.173}$$

To obtain the $(\mathbf{a}|r_{12}^{-1}|\mathbf{b})$ integrals of the desired combination of ℓ_a and ℓ_b using this recurrence relation, we must start by evaluating $(\mathbf{0}_A|\mathbf{0}_B)^{(m)}$ for some range of values of m. For example, in the case that the desired final result is $(\mathbf{a}|\mathbf{0}_B)$ (i.e. $\ell_b = 0$), then we must start with $(\mathbf{0}_A|\mathbf{0}_B)^{(m)}$ for $m = 0 \ldots \ell_a$. The introduction of an auxiliary index therefore complicates the evaluation of molecular integrals, increasing the number of intermediate integrals that must be evaluated and increasing the complexity of indexing these intermediates.

The recurrence relations that have been discussed so far (Eq. 2.157, Eq. 2.173) are referred to as "vertical" recurrence relations (VRRs). VRRs allow integrals with increased angular momentum in a particular index to be expressed in terms of intermediate integrals with lower angular momentum in that index (and with angular momentum in other indexes lower or unchanged). The recurrence is "vertical" in the sense that is allows integrals with higher angular momentum to be expressed using only integrals with lower angular momentum. Defining the total angular momentum of an integral as the sum of angular momentum in each index, it is immediately clear that Eqs. 2.165 and 2.173 are VRRs, since in both cases, an integral with angular momentum $\ell_a + \ell_b + 1$ is expressed in terms of integrals with $\ell_a + \ell_b$ and $\ell_a + \ell_b - 1$ units of angular momentum.

We have already mentioned two other major types of recurrence relation in the preceding derivation of the VRR for two-index overlap integrals (Eqs. 2.157 to 2.165). The first is the "horizontal" recurrence relation (HRR, see Eqs. 2.162 and 2.163). In 1988, Head-Gordon and Pople introduced the HRR in the context of the Obara-Saika scheme, using a combination of

VRRs and the HRR to accelerate the evaluation of four-index ERIs [101]. HRRs were in use prior to this in other contexts, e.g. the "transfer" equations of Ref. 96. This type of recurrence relation shifts angular momentum "horizontally", from one integral index to another, both of which must represent functions of the same electron coordinate. The HRR for the four-index ERIs (as described in Ref. 101) has the form

$$(\mathbf{a}(\mathbf{b}+\mathbf{1}_i)|\mathbf{cd})^{(m)} = ((\mathbf{a}+\mathbf{1}_i)\mathbf{b}|\mathbf{cd})^{(m)} + \mathbf{AB}_i(\mathbf{ab}|\mathbf{cd})^{(m)} \tag{2.174}$$

where angular momentum is incremented in index $|\mathbf{b})$ of the integral being assigned. In Eq. 2.174, the integral on the left has total angular momentum $\ell_a + \ell_b + \ell_c + \ell_d + 1$, and is expressed in terms of intermediate integrals with $\ell_a + \ell_b + \ell_c + \ell_d + 1$ and $\ell_a + \ell_b + \ell_c + \ell_d$ overall units of angular momentum. Unlike a VRR, where an integral of higher angular momentum is expressed solely in terms of integrals with lower overall angular momentum, a HRR redistributes angular momentum among indexes and features integrals with the same overall angular momentum on the left and right of the assignment.

Head-Gordon and Pople realized that, since there are no terms depending on the exponents of the primitive integral indexes in Eq. 2.174, the HRR can be applied to integrals after contraction with potentially significant computational savings (for details, see section 2.3.5). "Transfer" relations similar to the Eq. 2.174, but which shift more than one unit of angular momentum between centres have also been explored [102], though they have not been used in the work presented here.

The second major type of recurrence relation introduced in deriving the VRR for the two-index overlap integrals is the "translational" recurrence relation (TRR, see Eq. 2.159, also Ref. 10 (p.345)). This type of recurrence relation requires that the integral type satisfies the translational invariance relation (Eq. 2.153) and arises directly from the application of the differential operators in this relation. For example, for the two-index overlap integrals, the TRR has the form

$$(\mathbf{a}+\mathbf{1}_i|\mathbf{b}) = \frac{a_i}{2\zeta_a}(\mathbf{a}-\mathbf{1}_i|\mathbf{b}) - \frac{\zeta_b}{\zeta_a}(\mathbf{a}|\mathbf{b}+\mathbf{1}_i) + \frac{b_i}{2\zeta_a}(\mathbf{a}|\mathbf{b}-\mathbf{1}_i) \tag{2.175}$$

In general, the form of the TRR depends on the result of applying the differential operators in Eq. 2.153, which may vary between integral classes. Eq. 2.175 resembles a HRR in that the integral being assigned is expressed in terms of integrals with the same, or lower overall angular momentum. However because the coefficients for these integrals depend on the exponents of the primitive Cartesian Gaussian indexes, this relation cannot be applied after contraction of the primitive Cartesian Gaussians.

The computational implementation of the recurrence relations, and how they interact with the other elements of a molecular integral evaluation program, will be explored in section 2.3.5. For formal definitions of the various types of recurrence relation, see section 4.2.2.

2.3.4 The Boys function

In section 2.3.3, we introduced the Boys function, $F_m(T)$, which arose in the zero-angular-momentum expression for the two-index Coulomb integrals (Eq. 2.172). This function has the

form

$$F_n(T) = \int_0^1 \mathrm{d}t \exp(-Tt^2)t^{2n} \tag{2.176}$$

and is a generalization of the $F(z)$ function defined by Boys in 1950 [88, Eq. 16].

The Boys function arises where the integral transform of r_{12}^{-1} (Eq. 2.154) is used to express the Coulomb potential of a Gaussian charge distribution, i.e.

$$\begin{aligned} V_a(\mathbf{B}) &= \left(\frac{\zeta_a}{\pi}\right)^{3/2} \int \mathrm{d}\mathbf{r}_1 \, r_{1B}^{-1} \exp(-\zeta_a|\mathbf{r}_1 - \mathbf{A}|^2) \\ &= \left(\frac{2\zeta_a^{3/2}}{\pi^2}\right) \int_0^\infty \mathrm{d}t \int \mathrm{d}\mathbf{r}_1 \, \exp(-\zeta_a|\mathbf{r}_1 - \mathbf{A}|^2) \exp(-t^2|\mathbf{r}_1 - \mathbf{B}|^2) \end{aligned} \tag{2.177}$$

Applying the Gaussian product theorem (Eq. 2.138, appendix B), yields

$$\begin{aligned} V_a(\mathbf{B}) = \left(\frac{2\zeta_a^{3/2}}{\pi^2}\right) \int_0^\infty \mathrm{d}t \int \mathrm{d}\mathbf{r}_1 \, \exp(-(\zeta_a + t^2)|\mathbf{r}_1 - \mathbf{M}|^2) \\ \times \exp(-\frac{\zeta_a t^2}{\zeta_a + t^2}|\mathbf{A} - \mathbf{B}|^2) \end{aligned} \tag{2.178}$$

where

$$\mathbf{M} = \frac{\zeta_a \mathbf{A} + t^2 \mathbf{B}}{\zeta_a + t^2}.$$

Using the integral

$$\int \mathrm{d}\mathbf{r} \, \exp(-\alpha|\mathbf{r} - \mathbf{A}|^2) = \left(\frac{\pi}{\alpha}\right)^{3/2} \tag{2.179}$$

we obtain

$$V_a(\mathbf{B}) = \left(\frac{2\zeta_a^{3/2}}{\pi^{1/2}}\right) \int_0^\infty \mathrm{d}t \, (\zeta_a + t^2)^{-3/2} \exp(-\frac{\zeta_a t^2}{\zeta_a + t^2}|\mathbf{A} - \mathbf{B}|^2). \tag{2.180}$$

Finally, making the substitutions

$$u^2 = \frac{t^2}{\zeta_a + t^2} \qquad \mathrm{d}t = \frac{\zeta_a^{1/2}}{(1-u^2)^{3/2}}\mathrm{d}u \tag{2.181}$$

gives the Coulomb potential of a spherical Gaussian in terms of the Boys function:

$$\begin{aligned} V_a(\mathbf{B}) &= 2\left(\frac{\zeta_a}{\pi}\right)^{1/2} \int_0^1 \mathrm{d}u \, \exp(-u^2\zeta_a|\mathbf{A} - \mathbf{B}|^2) \\ &= 2\left(\frac{\zeta_a}{\pi}\right)^{1/2} F_0(\zeta_a|\mathbf{A} - \mathbf{B}|^2). \end{aligned} \tag{2.182}$$

The above derivation is based on the derivation presented in Ref. 10 (pp.361–364).

In the zero-angular-momentum expressions for the auxiliary integrals required to evaluate the Coulomb integrals (e.g. Eq. 2.172), it is necessary to evaluate $F_m(T)$ where $m > 0$. How this arises can be illustrated by considering a modified version of the Coulomb potential in

Eq. 2.177:

$$V_a^{(m)}(\mathbf{B}) = \left(\frac{2\zeta_a^{3/2}}{\pi^2}\right) \int_0^\infty \mathrm{d}t \int \mathrm{d}\mathbf{r}_1 \, \exp(-\zeta_a |\mathbf{r}_1 - \mathbf{A}|^2) \times \left(\frac{t^2}{\zeta_a + t^2}\right)^m \exp(-t^2 |\mathbf{r}_1 - \mathbf{B}|^2), \tag{2.183}$$

where we have introduced $[t^2/(\zeta_a + t^2)]^m$, in analogy to the $[u^2/(\rho + u^2)]^m$ term in Eq. 2.169. When $m = 0$, Eq. 2.183 is equivalent to Eq. 2.177. Repeating the steps of Eqs. 2.177 to 2.182 yields

$$\begin{aligned} V_a^{(m)}(\mathbf{B}) &= 2\left(\frac{\zeta_a}{\pi}\right)^{1/2} \int_0^1 \mathrm{d}u \, u^{2m} \exp(-u^2 \zeta_a |\mathbf{A} - \mathbf{B}|^2) \\ &= 2\left(\frac{\zeta_a}{\pi}\right)^{1/2} F_m(\zeta_a |\mathbf{A} - \mathbf{B}|^2). \end{aligned} \tag{2.184}$$

The zero-angular-momentum expression for the auxiliary two-index Coulomb integrals (Eq. 2.172) can be derived by a similar process,

$$\begin{aligned} (\mathbf{0}_A|\mathbf{0}_B)^{(m)} &= \frac{2}{\pi^{1/2}} \int_0^\infty \mathrm{d}u \, \left(\frac{u^2}{\rho + u^2}\right)^m (\mathbf{a}|g_{12}^{u^2}|\mathbf{b}) \\ &= 2\pi^{5/2} (\zeta_a \zeta_b)^{-1} \zeta^{-1/2} F_m(\xi |\mathbf{A} - \mathbf{B}|^2). \end{aligned} \tag{2.185}$$

The close relationship between the Boys function and the Coulomb operator means that the Boys function arises in many molecular integral classes. As we have already seen, the Boys function arises in the evaluation of two-electron Coulomb integrals, which are of central importance in the electronic structure methods outlined earlier in this chapter. In calculations using these methods, the three- or four-index Coulomb integrals (ERIs) are typically the most numerous and costly type of molecular integral required. For standard Hartree-Fock calculations ERI evaluation is the highest scaling step—M^4 (M is the number of AO basis functions) four-index ERIs over contracted basis functions must be evaluated, and each of these integrals generally requires the repeated evaluation of the Boys function. For this reason, it is vital that the implementation of the Boys function is computationally efficient. The efficient evaluation of the Boys function has been explored by many authors (see for example Refs. 103–106). In the work described in chapter 4, an approach which combines several approximate methods of evaluating $F_m(T)$, was adopted—for details, see appendix D.

2.3.5 Computational implementation

In the preceding sections, we have considered various theoretical aspects of the process of molecular integral evaluation. In a molecular integral evaluation program, careful consideration must be given to the computational implementation of these various elements, and how these interact inside the program. In this section, we will explore these computational considerations in a general sense, leaving more specific details to chapter 4.

An electronic structure calculation will typically involve the evaluation of a number of molec-

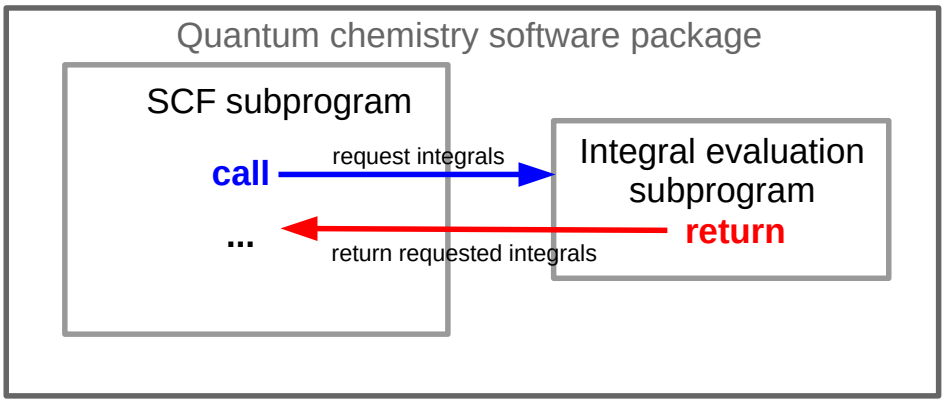

Figure 2.6: A diagram illustrating the separation of program units performing molecular integral evaluation from those performing specific calculations. Implementing molecular integral evaluation code in a general, method-independent fashion, avoids duplication of this code within a software package.

ular integrals over contracted AO basis functions. In a Hartree-Fock program (section 2.1.1), for example, these will be used to evaluate the Fock matrix elements (Eq. 2.47), while in a correlated method, they will be involved in evaluating the electron correlation energy (e.g. the MP2 energy expression, Eq. 2.110).

Molecular integral evaluation is a general procedure, required by many different methods and in many different contexts. It is therefore sensible for the code for molecular integral evaluation to be separate and independent from the program units that are specific to individual methods, within a quantum chemistry software package. In practice, this means that program units intended to perform specific calculations make calls to general molecular integral evaluation subprograms, providing the data necessary to perform integral evaluation for a specific purpose (as illustrated in Fig. 2.6).

The number, type and properties of AO integrals required in a given calculation is determined by several factors, including:

- The type of calculation being performed.
- The number and type of atoms in the system being studied.
- The AO basis set associated with each atom (and any auxiliary basis sets used).

In general, the program unit calling an integral evaluation subprogram will collate geometry and basis set data and make calls to the appropriate integral evaluation subprograms. The integral evaluation subprograms will then execute a suitable algorithm for evaluating the requested integrals, returning the result to the calling program unit (e.g. Fig. 2.6). The code generation work presented in in chapter 4 is primarily concerned with the internal workings of these integral evaluation subprograms and their interface with external code.

When called with suitable input, an integral evaluation subprogram will execute an algorithm designed to evaluate the requested integrals, returning the result to the calling program unit. The efficient design and implementation of these algorithms is the essence of the study of molecular integral evaluation and is central to chapter 4.

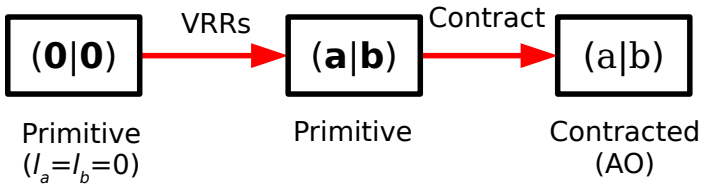

Figure 2.7: A schematic representation of the process of evaluating the two-index AO overlap integrals (a|b). In this case, the VRR (Eq. 2.157) is used to increment angular momentum in primitive Cartesian Gaussian indexes $|\mathbf{a})$ and $|\mathbf{b})$, starting with the zero-angular-momentum case (Eq. 2.155). Both indexes are then contracted with the AO basis contraction coefficients to produce (a|b). The spherical transformation step has been omitted for clarity.

A simple general algorithm for evaluating AO (contracted) integrals using an Obara-Saika-type scheme is as follows:

1. Evaluate the zero-angular-momentum expressions for the required integral type.
2. Use VRRs to increment angular momentum in the primitive Cartesian Gaussian indexes.
3. Contract the result with the contraction coefficients associated with the AO basis.

If the AO basis is in the spherical-harmonic representation, then spherical transformation (Eq. 2.131) of the contracted Cartesian GTOs will also be necessary. Fig. 2.7 is a schematic representation of this approach for the two-index overlap integrals.

As mentioned previously, for each value of total angular momentum, there is an associated number of Cartesian and spherical components (Eqs. 2.134 and 2.135). The collection of all n_{cart} or n_{sph} components is a "shell". The evaluation of AO integrals usually involves the simultaneous evaluation of all integrals for all combinations of angular momentum components, i.e. the outer product of the shells for each integral index—we shall refer to this as a "shell-block" of integrals. For example, a shell-block of two-index overlap integrals $(\mathbf{a}|\mathbf{b})$, with $\ell_a = \ell_b = 1$ has $n_{\text{cart},a} = n_{\text{cart},b} = 3$, and thus 9 possible combinations of angular momentum components.

Applying Obara-Saika-type recurrence relations (section 2.3.3) to generate shell-blocks, rather than individual integrals, avoids repeating some computations, since the integrals in a "shell-block" share common integral intermediates. As a simple example, consider the two-index overlap integral $(\mathbf{a}|\mathbf{0})$, with $\ell_a = 1$. The zero-angular-momentum case (Eq. 2.155) can be reused to evaluate all the components of the $\ell_a = 1, \ell_b = 0$ shell-block using Eq. 2.157:

$$\begin{aligned}(\mathbf{1}_x|\mathbf{0}_B) &= \mathbf{PA}_x(\mathbf{0}_A|\mathbf{0}_B),\\ (\mathbf{1}_y|\mathbf{0}_B) &= \mathbf{PA}_y(\mathbf{0}_A|\mathbf{0}_B),\\ (\mathbf{1}_z|\mathbf{0}_B) &= \mathbf{PA}_z(\mathbf{0}_A|\mathbf{0}_B).\end{aligned} \tag{2.186}$$

If each of the three integrals in the shell-block were evaluated separately, $(\mathbf{0}_A|\mathbf{0}_B)$ would need to re-evaluated in each case, or stored independently of the integral evaluation subprogram. As a consequence, integral evaluation algorithms generally process molecular integrals at the shell-block level.

The simple algorithm presented in Fig. 2.7 is not the only possible algorithm for the two-index overlap integrals. Fig. 2.8 presents an alternative algorithm, which makes use of a HRR

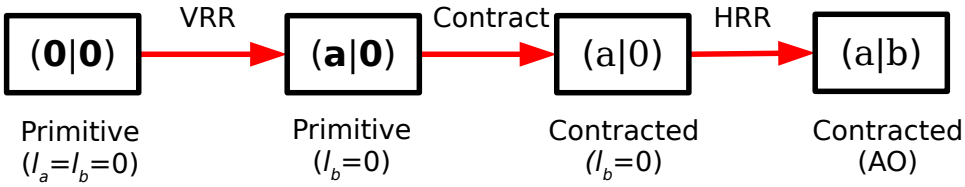

Figure 2.8: A schematic representation of the process of evaluating the two-index AO overlap integrals (a|b). In this case, the VRR (Eq. 2.157) is used to increment angular momentum in |**a**) starting with the zero-angular-momentum case (Eq. 2.155). Both indexes are contracted with the AO basis contraction coefficients to produce (a|0) and the HRR (Eq. 2.163) is applied to increment angular momentum in |b). The spherical transformation step has been omitted for clarity.

(Eq. 2.163), taking advantage of the fact that this recurrence relation can be applied to contracted integral indexes. This exemplifies one of the difficulties in developing software for the evaluation of molecular integrals—selecting the most appropriate algorithm from multiple possibilities. This is a difficult problem to solve, since the performance of a given algorithm varies, depending on the type of integral being evaluated, and the system being studied (particularly the nature of the AO basis set).

In selecting an appropriate algorithm for evaluating a particular integral type, we are usually concerned with minimizing the computational cost of evaluating AO integrals. In an Obara-Saika-type evaluation scheme, the evaluation of each shell-block of AO integrals involves the following general processes:

- Evaluation of zero-angular-momentum expressions.
- Use of recurrence relations to increment angular momentum.
- Contraction of primitive Cartesian Gaussians to contracted GTOs (AOs).
- Spherical transformation of contracted Cartesian GTOs (if spherical-harmonic GTOs are required).

Flexibility arises mainly in the choice of recurrence relations (VRR, HRR, TRR), and the ordering of the contraction, spherical transformation and angular momentum incrementing phases. For a given type of integral, there may be several plausible alternatives.

The use of algorithms where contractions occur earlier can confer significant computational savings when the AO basis is highly contracted. This was demonstrated for the four-index ERIs by Head-Gordon and Pople [101], who combined Obara and Saika's VRR for the ERIs with the HRR, to allow some angular momentum incrementing work to be done after contraction. To illustrate how such savings can arise, consider again the two-index AO overlap integrals (a|b). We have so far presented two algorithms for evaluating these integrals, a "VRR-only" algorithm (Fig. 2.7), and a Head-Gordon-Pople-like "VRR+HRR" algorithm (Fig. 2.8). The contraction step occurs earlier in the VRR+HRR algorithm, with some angular momentum incrementing work occurring after contraction has been done. The contraction step in both cases has the

form

$$(\mathrm{a}|\mathrm{b}) = \sum_{k=1}^{K_a}\sum_{l=1}^{K_b} d_k d_l (\mathbf{a}_k|\mathbf{b}_l) \tag{2.187}$$

where in the VRR+HRR case, $\mathbf{b}_l = \mathbf{0}$ and index $|\mathbf{a}_k)$ has total angular momentum $\ell_a + \ell_b$. Assuming $K_a = K_b = K$, the computational cost of computing a shell-block of (a|b) integrals rises as K^2 for both algorithms (with constant ℓ_a and ℓ_b), i.e. the total cost in terms of K is

$$t(K) = xK^2 + \mathcal{O}(K). \tag{2.188}$$

The size of x varies for the two algorithms: in the VRR-only case, the contraction involves $n_{\mathrm{cart}}(\ell_a)n_{\mathrm{cart}}(\ell_b)$ primitive integrals, while in the VRR+HRR case, $n_{\mathrm{cart}}(\ell_a + \ell_b)$ primitive integrals are required (see Eq. 2.134). Assuming that in both algorithms the cost of evaluating each primitive integral is equal, and that $xK^2 > \mathcal{O}(K)$, the VRR+HRR method would be the least costly approach. In reality, these assumptions are not correct, since the per-integral cost of the VRR phase depends on the angular momentum required and the complexity of the VRR expressions. However, this serves to illustrate the general idea behind the use of the HRR—contracting earlier makes some angular momentum incrementing work independent of K, and should therefore reduce the size of x in Eq. 2.188.

In practice, the theoretical comparison of integral evaluation algorithms requires detailed consideration of the relative cost of each step in terms of angular momentum and degree of contraction. This has been done for the four-index ERIs (see Refs. 10 (ch.9), 98, 101, and 107) where the cost of evaluating a shell-block of integrals can be expressed as

$$t(K) = xK^4 + yK^2 + z. \tag{2.189}$$

For highly-contracted (large K) cases, the K^4 steps dominate. Head-Gordon and Pople's algorithm shifts work from the K^4 scaling steps to the K^0 steps, thus offering significant computational benefits for large K. See Ref. 10 (ch.9), for a detailed theoretical cost analysis of different four-index ERI evaluation algorithms, where the scaling of each step in terms of both angular momentum and degree of contraction is considered.

It is, in fact, possible to perform contraction even earlier in an integral evaluation algorithm, using VRRs for contracted indexes with additional auxiliary indexes. This approach is used in the "PRISM" algorithm for four-index ERI evaluation, which allows the point at which contraction occurs in an algorithm to be dynamically selected [98, 108–110]. A conceptually similar early-contraction approach is used in the "J-matrix engine" algorithm, where the density matrix is summed into the ERIs before all the angular momentum incrementing work is completed [111].

We have so far been deliberately vague regarding computational "cost". Historically, the theoretical performance of integral algorithms has been measured in terms of numbers of floating point operations (FLOPs) or memory operations (MOPs) required (see Ref. 98 and references therein). An estimate of the FLOP or MOP cost of an algorithm can be made simply by counting the number of operations involved at each step in the algorithm. The benefit of such theoretical analyses is that they offer an indication of the performance of a given approach for

an "ideal" computer, isolated from the complexities of computation in the real-world. This can also be a significant issue, however, since real-world factors can be very important in determining performance—a poorly written implementation of an efficient algorithm may well be outperformed by an exemplary implementation of an inefficient algorithm. Similarly, an algorithm which appears highly efficient on paper, may be unsuited to the constraints of real hardware, and thus will perform poorly in reality. In addition, with parallel computer architectures growing in popularity, useful theoretical comparisons become increasingly difficult, since FLOP and MOP counts alone do not necessarily indicate real-world performance. For parallel hardware, the degree to which an algorithm is suitable for parallelization becomes an important determinant of its practical efficiency—a low operation-count algorithm may not be able to take advantage of parallelism, and therefore perform poorly (for some recent examples of parallel implementations of molecular integral evaluation code, see Refs. 112–115).

While theoretical cost analysis can be a useful tool for developing integral evaluation algorithms, the usefulness of a given integral evaluation algorithm is, in reality, determined by its real-world computational cost. An algorithm for integral evaluation only has real value if it can be practically and efficiently implemented in software. The most commonly used measure of real-world cost is execution time, i.e. how long does it take the software (using a chosen algorithm) to complete a particular task when run in a specific environment? This task might be simply forming a number of shell-blocks of integrals, or it may also incorporate other aspects of a calculation, such as performing a SCF iteration.

Measuring real-world performance is complicated by all the factors that theoretical analyses seek to avoid, and it is therefore important to control these as far as possible, for example by keeping constant the hardware/software environment in which the software is compiled and run. That execution timing data is subject to the complexities of real-world computation can be an advantage, provided the environment is properly controlled, since the real-world behaviour may significantly diverge from theoretical expectations as a result of these complexities.

The difficulty of comparing the performance different integral algorithms is addressed in chapter 4, where automatic code generation is shown to enable rapid implementation and real-world comparisons of different integral evaluation algorithms.

2.3.6 Final comments

The description of the theoretical and computational aspects of molecular integral evaluation presented in this section has strongly emphasized the theoretical model adopted for this programme of research, i.e. the evaluation of molecular integrals constructed from contracted Gaussian type orbitals, using Obara-Saika-type recurrence relations. For completeness, we will now briefly consider some important topics related to the study of molecular integrals and their evaluation which have not yet been covered.

Alternative approaches to integral evaluation

As has already been mentioned, the Obara-Saika-type recurrence relations are only one of several possible approaches to the evaluation of molecular integrals over Gaussian type orbitals.

Alternative general approaches include the Pople-Hehre algorithm [93], the Dupuis-Rys-King algorithm [94–96] and McMurchie-Davidson algorithm [97]. Detailed descriptions of these integral evaluation schemes can be found in Refs. 10 (ch.9), 98 and 99.

All of the approaches mentioned above have in common an algebraic approach to integral evaluation—integral evaluation is symbolic, and the method of evaluation is formally exact. Integration may also be performed approximately by using numerical, or combined symbolic/numerical approaches.

In the most general sense, numerical integration involves approximating an integral as a weighted sum of the integrand evaluated on a grid of integration points, i.e.

$$\int_a^b \mathrm{d}x f(x) \approx \sum_i W_i f(x_i). \tag{2.190}$$

The nature of the grid and weights is determined by the numerical integration method used.

Numerical integration is particularly important in DFT (section 2.2.5), where symbolic integration over the exchange-correlation potential is generally not possible (see for example Ref. 116 and references therein). Numerical integration has also been applied in evaluating the many-electron integrals that arise in explicitly correlated methods [117,118] (this is relevant to the work on density fitting these integrals presented in chapter 3).

Direct integral evaluation

In an SCF calculation, the AO integrals need only be calculated once, since the AOs themselves do not change during iterations—the density matrix (Eq. 2.48) is the only part of the Fock matrix element (Eq. 2.47) which changes in the iterative process. One might therefore assume that an efficient approach is to compute and store all the integrals needed prior to the SCF iterations. However, this approach rapidly becomes impractical as the size of the AO basis is increased.

The number of unique four-index ERIs for a given AO basis is given by the expression

$$\frac{1}{8}M(M+1)(M^2+M+2) \tag{2.191}$$

where M is the number of basis functions [119]. A SCF calculation involving 300 AO basis functions (a typical number for a molecule with 10s of atoms) would therefore require the storage of $\sim 10^9$ integrals. It is common to store the values of integrals as double precision floating point numbers, which in the IEEE standard for floating point arithmetic (IEEE 754 [120]) occupy 8 bytes (64 bits) of memory. Storing all the four-index ERIs as 8-byte double precision numbers would require $\sim$ 7800 MiB of memory. For 500 basis functions, the memory required to store all four-index ERIs rises to $\sim$ 60 GiB! Most computer systems in use today would not have sufficient memory to store all these at once, and would need to resort to writing the integrals to disk.

Aside from the difficulty of storing the integrals, the speed at which they can be accessed is also a problem. Modern CPUs can typically execute a large number of instructions in the time it takes to read data from the main memory (see Refs. 121 (ch.6) and 122 (ch.12))—if the

integrals are stored on disk, then the access time will be even greater. As a result, even if it is feasible to compute and store all integrals, accessing these integrals can present a bottleneck in the calculation.

"Direct" methods were developed to remedy these issues, by evaluating integrals only as needed. In the direct SCF method [123], the two-electron integrals are recomputed during every SCF iteration, and then discarded. Since the integrals are computed "on-the-fly" it is never necessary to store all the integrals at once, and the issue of limited storage space can be avoided. Additionally, the cost of accessing integrals is reduced, since "on-the-fly" evaluation and use of integrals favours efficient use of fast on-chip caches. The storage and access of molecular integrals can also present an issue for correlated methods (section 2.2) and to mitigate these issues, integral-direct correlated methods have also been developed [124–126].

The modern trend towards increasingly parallel computer architectures has shifted the balance further in favour of direct approaches. In developing code for massively parallel architectures, the challenge is to keep the many processing units occupied with tasks—rather than transferring data through a slow memory-access bottleneck, it is often better to make use of the abundant processing power available to recompute intermediate quantities as needed. This is particularly relevant when using GPUs for computation, where a very large quantity of processing units are available on the GPU card and the penalty for accessing main memory is currently very high (see for example Refs. 113, 127 and 128).

Approximation methods and integral screening

In the preceding sections, we have considered formally exact schemes for integral evaluation. The only errors that should arise in applying these schemes are the result of using finite precision arithmetic and approximating the Boys function (section 2.3.4). Often, the degree of accuracy offered by such methods is more than sufficient, and it is reasonable to consider introducing approximations that reduce the cost of evaluating the integrals.

One approach to approximating integrals is "screening", where integrals with magnitudes smaller than the desired accuracy are discarded. This is particularly valuable for accelerating the evaluation the ERIs, which are often the most numerous and costly integrals required in an electronic structure calculation.

For larger systems, many ERIs can be neglected because the product of two GTO basis functions rapidly decays with increasing separation of their centres (see Eq. 2.138 and appendix B). Clearly, this is only useful if we can predict which integrals are likely to be smaller than our chosen threshold in advance of calculating them. This can be achieved by employing an upper bound, such as Schwarz's inequality:

$$(\mathrm{ab}|\mathrm{cd}) \leq \sqrt{(\mathrm{ab}|\mathrm{ab})}\sqrt{(\mathrm{cd}|\mathrm{cd})}. \tag{2.192}$$

The number of diagonal integrals (ab|ab) is small ($\sim M^2$) compared to the total number of ERIs ($\sim M^4$) and it is therefore reasonable to use this upper bound to determine which integrals can be safely discarded, prior to evaluating all the ERIs. The use of Schwarz's inequality for screening ERIs was originally introduced by Häser and Ahlrichs in 1989, with the aim

of improving the efficiency of direct SCF calculations [129]. More recently, alternative upper bounds have also been derived by Lambrecht and Ochsenfeld [130] which, unlike Schwarz's inequality, account for the $1/R$ decay in the interaction between the charge distributions, $|ab)$ and $|cd)$ (R is the distance between the centres of these charge distributions).

The "multipole-moment" methods also make use of the behaviour of charge distributions with separation to approximate integrals. For well-separated charge distributions, the interaction between these may be approximated as a classical electrostatic interaction using multipole expansions of the Coulomb operator $1/r_{12}$. Thus, for well-separated |ab) and |cd), the ERI (ab|cd) may be approximated using such an expansion. Highly efficient methods for the evaluation of the Coulomb contribution to the Fock matrix have been developed around this approximation. See Refs. 10 (ch.9) and 99 for further details.

Screening and multipole-moment approximations reduce the computational expense of calculating molecular integrals by reducing the total number of integrals which need to be treated using a formally exact approach. For screening approaches, this is achieved by neglecting integrals which are smaller than some accuracy threshold. The multipole-moment methods treat some fraction of the integrals using a classical approximation which requires less computational work.

An alternative approximation approach is the use of fitting methods. We have already met one form of fitting in the preceding discussion of the R12/F12 methods (section 2.2.4), where resolutions of the identity (RIs) are used to approximate many-electron integrals (Eq. 2.122). These approximate RIs are equivalent to a fitting procedure [131], where three-index objects like

$$|\psi^r_{pq}\rangle = \int \mathrm{d}\mathbf{r}_1\, \psi_p(\mathbf{r}_1)\psi_q(\mathbf{r}_1) f_{12} \psi_r(\mathbf{r}_2) \tag{2.193}$$

are fitted in the auxiliary RI basis, i.e.

$$|\psi^r_{pq}\rangle \approx |\widetilde{\psi^r_{pq}}\rangle = \sum_x C^{r;y}_{pq} |x\rangle. \tag{2.194}$$

This can be viewed as a somewhat unusual case of "density fitting" (see appendix E). In density fitting molecular integrals, it is more common to fit products of atomic or molecular orbitals (charge densities), i.e.

$$|\mathrm{ab}) \approx |\widetilde{\mathrm{ab}}) = \sum_A D^{\mathrm{ab}}_A |A) \tag{2.195}$$

where the charge density $\varphi_a(\mathbf{r})\varphi_b(\mathbf{r})$ is fitted in the auxiliary basis $\{|A)\}$. Density fitting the products of orbitals that arise in molecular integrals reduces the number of integral indexes, and thus the number of integrals to be evaluated. Density fitting is central to the work presented in chapter 3, and is therefore described in detail in section 2.4.

A closely related approach is that of Cholesky decomposition, where the matrix of all ERIs is factorized into a lower triangular matrix and the corresponding conjugate transpose, i.e.

$$(\mathrm{ab}|\mathrm{cd}) \approx \sum_J^M L^J_{\mathrm{ab}} L^J_{\mathrm{cd}} \tag{2.196}$$

where M is determined by the desired accuracy of the approximation [132]. This decomposition has been shown to be equivalent to density fitting the integral with an automatically-generated auxiliary basis that reduces the linear dependence among the charge densities |ab) [133]. For further details on Cholesky decomposition for integral approximation, see Refs. 99, 133–135.

2.4 Density fitting

The idea of approximately expanding a function in some finite basis set is a motif repeated across the field of quantum chemistry: the exact wavefunction is expanded in a basis of Slater determinants (Eq. 2.58), MOs are expanded in a basis of AOs (Eq. 2.43), AOs are expanded in a basis of Gaussian functions (Eq. 2.127), and so on. Given the regular occurrence of products of one-electron basis functions in molecular integrals it is natural to ask whether these products can also be approximated in the same way, i.e.

$$|\mathrm{ab}) \equiv \varphi_a(\mathbf{r})\varphi_b(\mathbf{r}) \approx \sum_i c_i f_i(\mathbf{r}) \tag{2.197}$$

"Density fitting" (which has already been mentioned briefly in section 2.3.6) is one method to approximating products of basis functions in this way, and is central to the work presented in chapter 3.

We have already seen that the product of two Gaussian functions, $g_a(\mathbf{r})g_b(\mathbf{r}) \equiv |\mathbf{ab})$, may be exactly expanded in terms of a set of Gaussian functions centred between the two Gaussians $\{|\mathbf{p})\}$, via the Gaussian product theorem (GPT, appendix B):

$$|\mathbf{ab}) = \sum_{\mathbf{p}} T_{\mathbf{p}}^{\mathbf{ab}} |\mathbf{p}). \tag{2.198}$$

Density fitting is similar to the GPT, but more general, being applicable to products of more complicated one-electron functions, such as atomic and molecular orbitals, i.e.

$$|pq) \approx |\widetilde{pq}) = \sum_A D_A^{pq} |A) \tag{2.199}$$

where $|pq)$ is some product of one-electron basis functions, $|\widetilde{pq})$ is the density-fitted product, and $\{|A)\}$ is the auxiliary density fitting basis. Unlike the GPT, which is exact, density fitting is generally an approximate procedure.

The product $|pq)$ is a charge density in the sense that the ERI $(pq|r_{12}^{-1}|rs)$ describes the interaction between charge densities $\rho_{pq} = |pq)$ and $\rho_{rs} = |rs)$—hence the use of the term "density fitting" to refer to the fitting of products of one-electron basis functions. The fitting of the overall electron density $\rho(\mathbf{r})$ in an auxiliary basis is also sometimes referred to as "density fitting" (see for example Ref. 136). The fitting of products of one-electron basis functions, and the fitting of the total electron density are related, since the electron density in Hartree-Fock

theory (section 2.1.1) is a sum over products of orbitals, i.e. for the closed-shell case

$$\rho(\mathbf{r}) = 2 \sum_{i}^{N_{\mathrm{occ}}/2} |\psi_i(\mathbf{r})|^2 = \sum_{\mathrm{ab}} P_{\mathrm{ba}} |\mathrm{ab}) \tag{2.200}$$

where P_{ba} is the density matrix element for AO basis functions φ_a and φ_b:

$$P_{\mathrm{ba}} = 2 \sum_{i}^{N_{\mathrm{occ}}/2} C_{\mathrm{b}i} C^*_{\mathrm{a}i}.$$

In this section and the following chapters, the term "density fitting" will always be used to refer to the fitting of products of one-electron basis functions ("orbital products").

As alluded to in section 2.3.6, approximating molecular integrals using density fitting can yield impressive computational benefits. These benefits derive from the reduction in the number of integral indexes in density-fitted integrals—fewer integral indexes imply a reduction in the scaling of computational cost and storage requirements with respect to basis set size.

The use of density fitting to reduce the burden of evaluating molecular integrals has a long history—see Refs. 137, 138 for some early applications. In these early works, the focus was on approximating the numerous and expensive to evaluate ERIs, which posed a particular problem to the resource-constrained computers of the time. In 1973, Whitten published an important work on fitting these Coulomb integrals [139], describing an approach to fitting which was to form the foundation of modern density fitting methods. Whitten's key contribution was demonstrating that minimization of the error from density fitting an ERI,

$$(pq|r_{12}^{-1}|rs) - (\widetilde{pq}|r_{12}^{-1}|\widetilde{rs}), \tag{2.201}$$

can be achieved by minimizing the Coulomb energies of the fitting residuals,

$$\Delta_{pq} = (pq - \widetilde{pq}|r_{12}^{-1}|pq - \widetilde{pq}) \tag{2.202}$$

and identically defined Δ_{rs}. Dunlap, Connolly and Sabin arrived at the same result a few years later in the context of fitting the total electron density [136].

An alternative approach is to use a least-squares fitting to minimize the error in the fitted density. This corresponds to minimizing the overlap of the fitting residuals, i.e.

$$\Delta_{pq}^{S} = (pq - \widetilde{pq}|pq - \widetilde{pq}). \tag{2.203}$$

In their 1979 paper, Dunlap *et al* [136] compared the overlap (Eq. 2.203) and Coulomb (Eq. 2.202) fitting criteria and demonstrated that the Coulomb criterion offered a significant improvement in accuracy for fitting the total electron density. The superiority of the Coulomb criterion for fitting orbital products was later convincingly demonstrated by Vahtras, Almlöf and Feyereisen [140].

Using Whitten's approach, the density fitting coefficients of Eq. 2.199 are determined by minimizing Δ_{pq} for a given auxiliary basis set, $\{|A)\}$. This can be done by taking the derivative

of Δ_{pq} (Eq. 2.202) with respect to the density fitting coefficients,

$$\frac{\partial \Delta_{pq}}{\partial D_A^{pq}} = 2\sum_B (A|r_{12}^{-1}|B)D_B^{pq} - 2(A|r_{12}^{-1}|pq) \tag{2.204}$$

and setting this to zero, to obtain the linear equations

$$0 = \sum_B (A|r_{12}^{-1}|B)D_B^{pq} - (A|r_{12}^{-1}|pq). \tag{2.205}$$

Rearranged and recast in matrix form, these linear equations are

$$\mathbf{D}^{pq} = \mathbf{J}^{-1}\mathbf{J}^{pq} \tag{2.206}$$

with

$$\begin{aligned}(\mathbf{J}^{pq})_A &= (A|r_{12}^{-1}|pq)\\ (\mathbf{J})_{AB} &= (A|r_{12}^{-1}|B)\\ (\mathbf{D}^{pq})_B &= D_B^{pq}.\end{aligned}$$

The density fitting coefficients are thus given by

$$D_A^{pq} = \sum_B (\mathbf{J}^{-1})_{AB}(B|r_{12}^{-1}|pq). \tag{2.207}$$

Using Eq. 2.207, the ERIs can be approximated using only two- and three-index integrals, i.e.

$$\begin{aligned}(\widetilde{pq}|r_{12}^{-1}|\widetilde{rs}) &= \sum_{A,B} D_A^{pq}(A|r_{12}^{-1}|B)D_B^{rs}\\ &= \sum_{A,B}(pq|r_{12}^{-1}|A)(\mathbf{J}^{-1})_{AB}(B|r_{12}^{-1}|rs).\end{aligned} \tag{2.208}$$

To compute all the M^4 four-index ERIs using this approximate form, D^2 two-index and M^2D three-index integrals are required (D is the size of the density fitting basis). The number of these integrals is typically much less than the number of four-index ERIs, so the memory requirements for calculating the four-index ERIs can be considerably reduced.

Density fitting ERIs using Eq. 2.208 has been very successful, resulting in efficient density-fitted variants of a number of electronic structure methods, including Hartree-Fock [141–143], MP2 [57,144,145] and coupled cluster [146,147]. In discussing these methods, we will follow the nomenclature suggested in Ref. 57, and use the prefix "DF-" to refer to density-fitted variants, e.g. DF-MP2. In the literature, the prefix "RI-" is also used, alluding to the similarity of the density fitting procedure to a resolution of the identity.

One reason for the success of Coulomb-criterion-based density fitting in electronic structure theory is the small error incurred for fitting the ERIs with reasonably sized auxiliary basis sets (see Ref. 140). This can be understood by examining the expression for the error in the fitted

integrals:

$$\begin{aligned}(pq|r_{12}^{-1}|rs) - (\widetilde{pq}|r_{12}^{-1}|\widetilde{rs}) = (pq - \widetilde{pq}|r_{12}^{-1}|rs - \widetilde{rs}) + (\widetilde{pq}|r_{12}^{-1}|rs - \widetilde{rs}) \\ + (pq - \widetilde{pq}|r_{12}^{-1}|\widetilde{rs}).\end{aligned} \tag{2.209}$$

The second and third terms on the right vanish when the densities are fitted with the Coulomb criterion, since

$$\begin{aligned}(pq - \widetilde{pq}|r_{12}^{-1}|\widetilde{rs}) &= \sum_B D_B^{rs}\left[(pq|r_{12}^{-1}|B) - \sum_A D_A^{pq}(A|r_{12}^{-1}|B)\right] \\ (\widetilde{pq}|r_{12}^{-1}|rs - \widetilde{rs}) &= \sum_A D_A^{pq}\left[(A|r_{12}^{-1}|rs) - \sum_B (A|r_{12}^{-1}|B)D_B^{rs}\right]\end{aligned}$$

where, according to Eq. 2.205, the expressions inside the square brackets are zero. The error in the fitted integrals is therefore

$$(pq|r_{12}^{-1}|rs) - (\widetilde{pq}|r_{12}^{-1}|\widetilde{rs}) = (pq - \widetilde{pq}|r_{12}^{-1}|rs - \widetilde{rs}), \tag{2.210}$$

which is quadratic in the error in the fitted density. As a consequence, the error in the fitted integral converges rapidly with respect to the quality of the density-fit (i.e. the size of the fitting basis). Density fitting procedures which exhibit a quadratic dependence on the error in the fitted density are termed "robust" [148].

The poor performance of overlap-criterion-based density fitting of the ERIs described in Ref. 140 is likely to be mainly the result the non-robust nature of this fitting procedure. Using the overlap criterion to optimize the fit (Eq. 2.203), yields the linear equations (cf. Eq. 2.205),

$$0 = \sum_B (A|B)C_B^{pq} - (A|pq), \tag{2.211}$$

where $(A|B)$ and $(A|pq)$ are two- and three-index overlap integrals and C_B^{pq} are the fitting coefficients. In this case, the terms linear in the fitting residuals in Eq. 2.209 remain, and the error in the fitted ERIs is linear in the error in the fitted densities.

The concept of "robustness" is particularly relevant to the application of density fitting to explicitly correlated methods (section 2.2.4). A consequence of using resolutions of the identity to approximate the many-electron integrals that arise in the R12/F12 methods (Eq. 2.122) is that it becomes necessary to evaluate four-index integrals other than the ERIs, featuring the correlation factor f_{12}, e.g. $(pq|f_{12}|rs)$. If the density-fit is optimized using the Coulomb criterion (Eq. 2.202), then fitting these integrals using expressions analogous to Eq. 2.208 is non-robust. This is the case for any four-index integral over a general two-electron operator $\hat{\omega}_{12}$, where $\hat{\omega}_{12} \neq r_{12}^{-1}$, since the error in the fitted integral is

$$\begin{aligned}(pq|\hat{\omega}_{12}|rs) - (\widetilde{pq}|\hat{\omega}_{12}|\widetilde{rs}) = (pq - \widetilde{pq}|\hat{\omega}_{12}|rs - \widetilde{rs}) + (\widetilde{pq}|\hat{\omega}_{12}|rs - \widetilde{rs}) \\ + (pq - \widetilde{pq}|\hat{\omega}_{12}|\widetilde{rs}),\end{aligned} \tag{2.212}$$

and the terms linear in the fitting residuals do not vanish.

A robust density fitting expression for these integrals is [148, 149]

$$(pq|\hat{\omega}_{12}|rs)_{\text{robust}} = (\widetilde{pq}|\hat{\omega}_{12}|rs) + (pq|\hat{\omega}_{12}|\widetilde{rs}) - (\widetilde{pq}|\hat{\omega}_{12}|\widetilde{rs}), \tag{2.213}$$

for which the error in the fitted integral is quadratic in the fitted density, i.e.

$$(pq|\hat{\omega}_{12}|rs) - (pq|\hat{\omega}_{12}|rs)_{\text{robust}} = (pq - \widetilde{pq}|\hat{\omega}_{12}|rs - \widetilde{rs}). \tag{2.214}$$

In developing DF-MP2-R12 theory, Manby recognized this issue and used Eq. 2.213 to ensure the new four-index integrals featuring the correlation factor were robustly fitted [131]. Approximating the additional four-index integrals that arise in explicitly correlated MP2 in this way yields large gains in efficiency without any appreciable decrease in accuracy [70, 81, 131]. In chapter 3, the application of robust density fitting directly to many-electron integrals (without resolutions of the identity) is described (see also, Ref. 150).

In density fitting approaches, the auxiliary basis set is typically fixed and pre-optimized, with the fitting coefficients determined for a specific auxiliary basis set when the electronic structure calculation is run (using Eq. 2.207). To effectively design and optimize a density fitting basis, it is necessary to take account of the context in which the fitting is to occur, since, for example, a density fitting basis for use in DF-MP2 has different requirements to a fitting basis used in DF-HF. In addition, since the nature of the fitted density depends on the AO basis set used, it is often the case that density fitting basis sets are optimized for use in combination with a specific AO basis (and thus, a specific atom-type).

In this thesis, we use the common naming convention [AO basis]/[fit type] for density fitting basis sets (as used in Molpro [151,152]). For example, a density fitting basis set optimized for use with a cc-pVDZ AO basis in a DF-MP2 calculation would be designated "cc-pVDZ/MP2FIT". "MP2FIT" indicates that the density fitting basis is intended for use in DF-MP2 calculations and has been designed to minimize the error in the MP2 correlation energy arising from density fitting. This can be done by minimizing the quantity [145]

$$\delta_{\text{DF}} = \frac{1}{4}\sum_{ijab}\frac{[(ai||bj)_{\text{DF}} - (ai||bj)]^2}{\epsilon_a + \epsilon_b - \epsilon_i - \epsilon_j} \tag{2.215}$$

where

$$(ai||bj) \equiv (ai|r_{12}^{-1}|bj) - (aj|r_{12}^{-1}|bi).$$

In contrast,"JKFIT" fitting sets are intended for use in DF-HF calculations, and are optimized by minimizing the error in the exchange contribution to the energy [142]. A great variety of density fitting basis sets have been developed—for further details of the design and optimization of these, see Ref. 90, and references therein.

An important concern in the development of density fitting basis sets is the maximum angular momentum required in the basis. In a fit of products of GTOs on the same atomic centre, $|\text{ab}) \approx |\widetilde{\text{ab}})$ with $\mathbf{A} = \mathbf{B}$, this is easily determined, since the GPT (appendix B) indicates that the product of two Gaussians can be exactly expanded in a basis of Gaussian functions with angular momentum up to $\ell_a + \ell_b$. An approximate fitting basis for |ab) would therefore

require Gaussian functions centred on the atom at position **A** with angular momentum up to $\ell_a + \ell_b$. It is less clear what level of angular momentum is required to fit a charge density that is not atom-centred (using atom-centred density fitting basis functions), though the maximum angular momentum required in the atom-centred case can serve as a useful guide. In chapter 3 the angular momentum requirements of density fitting basis sets for fitting the many-electron integrals in MP2-F12 theory are evaluated.

CHAPTER 3

DENSITY-FITTED MANY-ELECTRON INTEGRALS IN EXPLICITLY CORRELATED METHODS

Note on previously published work

This chapter describes research that was previously published in Ref. 150, for which I was the main author. The account presented here is intended to extend and refine the account presented in the published work, but will necessarily share significant features with Ref. 150.

Notation and nomenclature

Implicit summation over repeated dummy indexes, e.g.

$$S_{ij} = \sum_k X_{ik} Y_{kj} \qquad \rightarrow \qquad S_{ij} = X_{ik} Y_{kj} \tag{3.1}$$

is used throughout this chapter, since this convention is used in much of the published work on explicitly correlated methods. Explicit summation notation will occasionally be used for emphasis.

There is some ambiguity in the notation and nomenclature used the published literature to describe explicitly correlated methods, particularly in earlier papers. In this chapter, the definitions and conventions used in Ref. 71 have been adopted for describing MP2-F12 theory and associated approximations.

The prefix "DF3-" is used to indicate that three-electron integrals are approximated by density fitting (as described in section 3.2.3), rather than resolution of the identity, e.g. "DF3-MP2-F12". Note that DF3-MP2-F12 is distinct from DF-MP2-F12 [70, 131]—in DF-MP2-F12, resolutions of the identity are used to approximate the many-electron integrals and density fitting is applied to the resulting two-electron integrals.

3.1 Introduction

The conventional post-Hartree-Fock *ab initio* methods (section 2.2) suffer from slow convergence of the error in the electron correlation energy with respect to AO basis set size, as a consequence

of the unsuitability of atom-centred, one-electron basis functions for describing the Coulomb hole (Eq. 2.56, Fig. 2.1). Explicitly correlated methods improve the representation of the Coulomb hole by introducing explicit dependence on the interelectronic distance, r_{12}, to the approximate molecular wavefunction (for details regarding the history of the explicitly correlated methods, and their essential features, see section 2.2.4).

Unfortunately, the use of explicitly correlated wavefunctions in electronic structure calculations results in the need to evaluate "many-electron" integrals, i.e. integrals over more than two electron coordinates. These complicated and numerous many-electron integrals present a significant challenge—their exact evaluation for molecular systems is, in general, computationally infeasible. Application of explicitly correlated approaches to systems of chemical interest requires the approximation of the many-electron integrals.

Many attempts have been made to tackle the problematic many-electron integrals arising from the use of explicitly correlated wavefunctions, for example:

- The derivation of closed-form expressions for integrals over Gaussian geminals by Boys [153] and Singer [154] in the 1960s.
- The introduction of the "weak orthogonality functional" by Szalewicz, Jeziorski, Monkhorst and Zabolitzky, to avoid the need to evaluate integrals over more than three electrons in explicitly correlated second-order MBPT [63, 155].
- The combination of analytic integration and numerical quadrature for the evaluation of three-electron integrals in Boys and Handy's transcorrelated method [62], and later in Ten-no's formulation of explicitly correlated MP2 [117].
- The derivation of Obara-Saika-type recurrence relations (section 2.3.3) for three-electron integrals by Ten-no [117, 156] and May [89].
- Numerical integration of three-electron integrals by discretization of the integral transform of r_{12}^{-1} (Eq. 2.154) [157].

See Ref. 56 for further details on some of these approaches to many-electron integral evaluation.

At present, the most successful method of approximating these integrals is the use of approximate resolutions of the identity (RIs), as first proposed by Kutzelnigg in the 1980s [66]. By insertion of an approximate identity operator, many-electron integrals can be factorized into sums over products of integrals over fewer electron coordinates, which are more computationally tractable—see section 3.2.4 for theoretical details. The R12 and F12 methods, initially developed by Kutzelnigg and Klopper [68], make use of RIs to approximate the many-electron integrals arising from introduction of the correlation factor, f_{12}. These methods are widely available in *ab initio* electronic structure software packages (including Molpro [151, 152] and TURBOMOLE [158]).

The success of the R12 and F12 approaches can perhaps be largely attributed to the use of RIs to approximate many-electron integrals. Using approximate RIs, integrals over more than two electron coordinates can be completely avoided, simplifying the derivation and implementation of integral evaluation schemes for these methods.

In the decades since Kutzelnigg and Klopper's original work, the R12/F12 approaches have seen significant developments and improvements, including:

- Nonlinear correlation factors, $f_{12} \neq r_{12}$ (F12 methods), offering dramatic improvements in accuracy for a given AO basis set [69, 70, 159–162].
- Efficient evaluation of two-electron integrals arising from RI approximation using density fitting (DF-MP2-R12/F12, section 2.4) [70, 81, 131].
- Reduced computational scaling with respect to system size using local methods [81, 163–168]
- Perturbative correction of the basis-set error in the Hartree-Fock energy ("CABS singles" correction) [80, 169, 170].
- The development of F12 variants of coupled-cluster methods (reviewed in Refs. 25, 171, 172).

The implementation of the approximate RIs has also seen notable advances. Early versions of the R12 approach (e.g. Ref. 68) used the MO basis set to form the approximate identity operator. To accurately approximate many-electron integrals, the RI basis should contain high-angular-momentum functions (see section 3.2.4)—using the MO basis for the RI passes this requirement on to the MO basis and thus places constraints on the AO basis set in which the MOs are represented. The auxiliary basis set (ABS) approach [173] allows a separate, auxiliary, basis set to be used for the approximate RI, avoiding angular-momentum constraints on the AO basis. The use of an auxiliary RI basis also has the advantages that the RI basis can be optimized specifically for the purpose of minimizing the RI error, and that this error can be reduced to a negligible size by employing a larger RI basis. The RI error can be further reduced by employing the complementary auxiliary basis set (CABS) approach [174], in which the auxiliary basis set approximates only the orthogonal complement to the MO basis. For further detail on the use of RIs in R12/F12 methods and the ABS and CABS approaches, see section 3.2.4.

The use of RIs to approximate many-electron integrals is not without problems. As has already been mentioned, the amount of angular momentum required in the RI basis to accurately represent the many-electron integrals is quite high. For the three-electron integrals arising in the MP2-F12 theory, it can be shown that, for a single-centre system (i.e. an atom), the angular momentum required in the RI basis is at least $3\ell_{\text{occ}}$ (where ℓ_{occ} is the maximum angular momentum for the occupied orbitals)—see section 3.2.4 for details. For example, a transition metal atom with occupied d-orbitals ($\ell_{\text{occ}} = 2$) would require a RI basis containing i-functions ($\ell = 6$). As a consequence, large RI basis sets with high angular momentum functions are often necessary. As mentioned in section 2.3, large basis sets are computationally problematic, leading to difficulties storing and accessing integrals as well as increased computational expense. The high angular momentum functions themselves can also be problematic—the angular momentum required to saturate the RI basis may exceed the maximum supported by some quantum chemistry codes.

The angular momentum required in density fitting the product of two occupied MOs, $|ij)$, is $2\ell_{\mathrm{occ}}$ (see sections 2.4 and 3.2.3). This raises a question—can density fitting be used to avoid the high angular momentum requirement involved in approximating many-electron integrals using RIs? In DF-MP2-R12/F12 theory [70, 81, 131], the application of density fitting to two-electron integrals has no effect on the angular momentum required in the RI basis, since the essential nature of the RI is unchanged. In contrast, in Ten-no and Manby's composite RI+DF method [175], the combination of a modified RI expression and density fitting reduces the angular momentum required in the RI basis from $3\ell_{\mathrm{occ}}$ to $2\ell_{\mathrm{occ}}$ for the single-centre case (see section 3.2.4). This indicates that density fitting can indeed be used in approaches which reduce the angular momentum required in the RI basis.

In both DF-MP2-R12/F12 theory and the RI+DF approach, density fitting is used in combination with RIs, introducing an additional auxiliary basis set on top of the orbital and RI sets. This raises another question—can density fitting alone be used to approximate many-electron integrals? In this case, it might be possible to use a single density fitting basis set to approximate two-electron integrals (as in DF-MP2 and DF-MP2-R12/F12) as well as many-electron integrals, avoiding the introduction of an RI basis completely. The problematic angular momentum requirements of using RIs to approximate many-electron integrals would also be avoided.

The work presented in Ref. 150 and in this chapter is the realisation of this idea of supplanting the RIs in R12/F12 methods with density fitting, as originally conceived by Manby [176]. In the following, the derivation and implementation of a RI-free MP2-F12 theory, where many-electron integrals are approximated using density fitting, is presented (sections 3.2 and 3.3). Subsequently, the behaviour of the correlation energy calculated by this method is examined with respect to the size and maximum angular-momentum of the density fitting basis set (section 3.4). As we shall see, the behaviour of this new density-fitting-based, RI-free, MP2-F12 theory is consistent with our expectations, demonstrating rapid convergence in the MP2-F12 correlation energy with respect to the size of the density fitting basis and lower angular momentum requirements than corresponding RI-based approaches.

3.2 Theory

3.2.1 MP2-F12 theory

In MP2-F12 theory, the normal MP2 pair function (Eq. 2.118) is augmented with explicitly correlated terms, i.e.

$$|u_{ij}\rangle = t_{ab}^{ij}|ab\rangle + \hat{Q}_{12}f_{12}t_{kl}^{ij}|kl\rangle, \tag{3.2}$$

where $\hat{Q}_{12}$ is the "strong orthogonality" projector, ensuring strong orthogonality of the pair function with the occupied MOs:

$$\langle u_{ij}|k\rangle = 0, \qquad \forall\, k. \tag{3.3}$$

Strong orthogonality is necessary for the formulation of pair theories—without it, the correlation energy would not be separable into independent contributions from pairs of orbitals (for early work regarding pair theories and strong orthogonality, see Refs. 49, 177–179).

In canonical MP2 theory, the pair function is always strongly orthogonal to the occupied orbitals, since the virtual MOs are orthogonal to the occupied MOs, but this is not the case for the MP2-F12 pair function and it is necessary to enforce strong orthogonality using $\hat{Q}_{12}$. The pair functions, and their correlation energy contributions to the total MP2-F12 correlation energy may be determined by variational minimization of the Hylleraas pair functional (see section 2.2.3),

$$e_{ij}[u_{ij}] = \langle u_{ij}|\hat{f}_1 + \hat{f}_2 - \epsilon_i - \epsilon_j|u_{ij}\rangle + 2\langle u_{ij}|r_{12}^{-1}|ij\rangle. \tag{3.4}$$

Inserting Eq. 3.2 into the Hylleraas pair functional yields

$$\begin{aligned} e_{ij}[u_{ij}] &= e_{ij}^{(\mathrm{MP2})} + e_{ij}^{(\mathrm{F12})} + 2t_{ab}^{ij}t_{kl}^{ij}\langle kl|f_{12}\hat{Q}_{12}(\hat{f}_1 + \hat{f}_2 - \epsilon_i - \epsilon_j)|ab\rangle \\ &= e_{ij}^{(\mathrm{MP2})} + e_{ij}^{(\mathrm{F12})} + 2t_{ab}^{ij}t_{kl}^{ij}(C_{ab}^{kl} - (\epsilon_i + \epsilon_j)\mathcal{F}_{ab}^{kl}) \end{aligned} \tag{3.5}$$

where $e_{ij}^{(\mathrm{MP2})}$ is the conventional MP2 pair energy (Eq. 2.116),

$$e_{ij}^{(\mathrm{F12})} = t_{kl}^{ij}B_{kl,mn}t_{mn}^{ij} - (\epsilon_i + \epsilon_j)t_{kl}^{ij}X_{kl,mn}t_{mn}^{ij} + 2t_{kl}^{ij}V_{kl}^{ij}. \tag{3.6}$$

and

$$C_{ab}^{kl} = \langle kl|f_{12}\hat{Q}_{12}(\hat{f}_1 + \hat{f}_2)|ab\rangle, \tag{3.7}$$

$$\mathcal{F}_{ab}^{kl} = \langle kl|f_{12}\hat{Q}_{12}|ab\rangle, \tag{3.8}$$

$$B_{kl,mn} = \langle kl|f_{12}\hat{Q}_{12}(\hat{f}_1 + \hat{f}_2)\hat{Q}_{12}f_{12}|mn\rangle, \tag{3.9}$$

$$X_{kl,mn} = \langle kl|f_{12}\hat{Q}_{12}f_{12}|mn\rangle, \tag{3.10}$$

$$V_{kl}^{ij} = \langle ij|r_{12}^{-1}\hat{Q}_{12}f_{12}|kl\rangle. \tag{3.11}$$

The matrix elements defined in Eqs. 3.7 to 3.11 are defined as in Ref. 71

In this work, we adopt a particular variant of MP2-F12 theory, which (using the nomenclature of Ref. 71) is termed "MP2-F12/3*A(FIX)". The suffix "3*A(FIX)" specifies a wavefunction ansatz and set of approximations—there are many other possible approximation schemes for MP2-F12 theory with different designations, as detailed in Ref. 71 (see, in particular Table II for a concise summary).

The 3*A(FIX) approximation scheme has the following characteristics:

Ansatz 3 strong orthogonality projector

Several ansatzes for the strong orthogonality projector, $\hat{Q}_{12}$, have been used in the R12/F12 methods. In this work, we use the ansatz 3 projector (as advocated in Refs. 70,71,174,180),

$$\hat{Q}_{12} = (1 - \hat{o}_1)(1 - \hat{o}_2)(1 - \hat{v}_1\hat{v}_2), \tag{3.12}$$

where $\hat{o} = |i\rangle\langle i|$ and $\hat{v} = |a\rangle\langle a|$ are projection operators onto the occupied and virtual MO spaces. The properties of alternative $\hat{Q}_{12}$ ansatzes are discussed in Ref. 56. A particular advantage of ansatz 3 is that, via the $(1-\hat{v}_1\hat{v}_2)$ term, it ensures that the explicitly correlated part of the MP2-F12 pair function (Eq. 3.2) is orthogonal to the standard MP2 pair function,

leading to simpler working equations.

Generalized and extended Brillouin conditions (GBC and EBC)

Under the generalized Brillouin condition (GBC) [68], the occupied orbital space is assumed to be closed under the Fock operator:

$$\hat{f}|i\rangle = \epsilon_i|i\rangle. \tag{3.13}$$

If the extended Brillouin condition (EBC) [68] is assumed, then the virtual orbital space is also assumed to be closed under the Fock operator:

$$\hat{f}|a\rangle = \epsilon_a|a\rangle. \tag{3.14}$$

For the exact Hartree-Fock orbitals, which are eigenvalues of the Fock operator, both these conditions are exactly satisfied. For the approximate Hartree-Fock orbitals that are the result of a Hartree-Fock SCF (section 2.1.1) calculation, the GBC is a reasonable approximation. The EBC is less well justified for approximate Hartree-Fock orbitals. Nevertheless, the combined impact of assuming the GBC and EBC has been shown to be relatively small (on the order of 1 millihartree in the analysis performed in Ref. 69).

Neglect of exchange commutator terms

The identity

$$f_{12}\hat{f}_{12} = \hat{f}_{12}f_{12} + [f_{12}, \hat{f}_{12}] \tag{3.15}$$

is used in the derivation of the "standard approximations" of Kutzelnigg and Klopper [68] ($\hat{f}_{12} = \hat{f}_1 + \hat{f}_2$). This simplifies certain matrix elements present in MP2-F12 theory, for example $A_{kl,mn}$ (which we will shortly consider in detail):

$$\begin{aligned}\langle kl|f_{12}\hat{f}_{12}\hat{Q}_{12}f_{12}|mn\rangle = \langle kl|\hat{f}_{12}f_{12}\hat{Q}_{12}f_{12}|mn\rangle \\ + \langle kl|[f_{12}, \hat{f}_{12}]\hat{Q}_{12}f_{12}|mn\rangle.\end{aligned} \tag{3.16}$$

The first term in Eq. 3.16 is simplified by application of the GBC, i.e.

$$\begin{aligned}\langle kl|f_{12}\hat{f}_{12}\hat{Q}_{12}f_{12}|mn\rangle &= (\epsilon_k + \epsilon_l)\langle kl|f_{12}\hat{Q}_{12}f_{12}|mn\rangle \\ &\quad + \langle kl|[f_{12}, \hat{f}_{12}]\hat{Q}_{12}f_{12}|mn\rangle \\ &= (\epsilon_k + \epsilon_l)X_{kl,mn} + \langle kl|[f_{12}, \hat{f}_{12}]\hat{Q}_{12}f_{12}|mn\rangle.\end{aligned} \tag{3.17}$$

Since the correlation factor, f_{12}, commutes with all components of the Fock operator, $\hat{f}$, except the non-local kinetic energy, $\hat{t} = -\frac{1}{2}\nabla^2$, and exchange, $\hat{k} = \sum_i \hat{K}_i$, operators (Eq. 2.34), the commutator of the second term in Eq. 3.17 may be expressed as

$$[f_{12}, \hat{f}_{12}] = [f_{12}, \hat{t}_{12}] - [f_{12}, \hat{k}_{12}] \tag{3.18}$$

with $\hat{t}_{12} = \hat{t}_1 + \hat{t}_2$ and $\hat{k}_{12} = \hat{k}_1 + \hat{k}_2$. In approximation "A", exchange commutator $[f_{12}, \hat{k}_{12}]$ terms arising from $A_{kl,mn}$ are neglected. In fact, under the "3*A" scheme, no matrix elements featuring the exchange operator need evaluating, since the EBC additionally results in the neglect of C_{ab}^{kl} [71].

Diagonal approximation with fixed amplitudes

In the diagonal approximation, the summation over occupied MOs in the explicitly correlated part of the MP2-F12 pair function (Eq. 3.2) is restricted to the diagonal orbital pairs ij and ji. This approximation simplifies the MP2-F12 energy equations by allowing further matrix elements to be neglected, and means that only diagonal elements of the **B** and **V** matrices need to be evaluated. These simplifications facilitate the development of efficient, lower scaling variants of MP2-F12, for example DF-LMP2-F12/3*A(DX) in Ref. 163 (where the method is denoted DF-LMP2-F12/2*A(loc), using a different naming convention). Unfortunately, determining the explicitly correlated amplitudes (t_{kl}^{ij} in Eq. 3.2) by minimization of Eq. 3.5 under the diagonal approximation produces results that are orbital-variant, i.e. not invariant with respect to unitary transformations of the occupied MOs [67].

Fixing the explicitly correlated amplitudes prior to minimization of Eq. 3.5 restores orbital invariance. This is the approach taken in the fixed-amplitude ansatz, denoted by "FIX", where the diagonal amplitudes are chosen to satisfy the singlet ($t_{ij}^{ij} = t_{ji}^{ij} = \frac{1}{2}$) and triplet ($t_{ij}^{ij} = t_{ji}^{ij} = \frac{1}{4}$) electronic cusp conditions [26–28] (section 2.2). See Refs. 117, 181, 182 for further details on the fixed-amplitude approximation.

Applying the 3*A(FIX) approximations, the MP2-F12 pair energy (Eq. 3.5) separates entirely into the standard MP2 contribution and the F12 correction, i.e.

$$e_{ij}[u_{ij}] = e_{ij}^{(\mathrm{MP2})} + e_{ij}^{(\mathrm{F12})}. \tag{3.19}$$

The coupling term in Eq. 3.5,

$$2t_{ab}^{ij} t_{kl}^{ij} (C_{ab}^{kl} - (\epsilon_i + \epsilon_j)\mathcal{F}_{ab}^{kl})$$

vanishes, since under the EBC,

$$C_{ab}^{kl} = \mathcal{F}_{ab}^{kl}(\epsilon_a + \epsilon_b) \tag{3.20}$$

and using the ansatz 3 projector $\mathcal{F}_{ab}^{kl} = 0$ (since $\hat{Q}_{12}|ab\rangle = 0$). The standard MP2 amplitudes (t_{ab}^{ij} in Eq. 3.2) are therefore completely independent of the explicitly correlated amplitudes (t_{kl}^{ij} in Eq. 3.2).

Further simplifications arise under the 3*A(FIX) scheme when the expanded expression for the ansatz 3 projector,

$$\hat{Q}_{12} = 1 + \hat{o}_1\hat{o}_2 - \hat{o}_1 - \hat{o}_2 - \hat{v}_1\hat{v}_2, \tag{3.21}$$

is inserted into $B_{kl,mn}$ (Eq. 3.9):

$$\begin{aligned} B_{kl,mn} = &\langle kl|f_{12}\hat{f}_{12}\hat{Q}_{12}f_{12}|mn\rangle \\ &+ \langle kl|f_{12}\hat{o}_1\hat{o}_2\hat{f}_{12}\hat{Q}_{12}f_{12}|mn\rangle \\ &- \langle kl|f_{12}[\hat{o}_1\hat{f}_{12} + \hat{o}_2\hat{f}_{12}]\hat{Q}_{12}f_{12}|mn\rangle \\ &- \langle kl|f_{12}\hat{v}_1\hat{v}_2\hat{f}_{12}\hat{Q}_{12}f_{12}|mn\rangle. \end{aligned} \tag{3.22}$$

Applying some of the one-electron projection operators and the GBC/EBC yields

$$\begin{aligned} B_{kl,mn} = &\langle kl|f_{12}\hat{f}_{12}\hat{Q}_{12}f_{12}|mn\rangle \\ &+ (\epsilon_i + \epsilon_j)\langle kl|f_{12}|ij\rangle\langle ij|\hat{Q}_{12}f_{12}|mn\rangle \\ &- \langle kl|f_{12}[\hat{o}_1\hat{f}_{12} + \hat{o}_2\hat{f}_{12}]\hat{Q}_{12}f_{12}|mn\rangle \\ &- (\epsilon_a + \epsilon_b)\langle kl|f_{12}|ab\rangle\langle ab|\hat{Q}_{12}f_{12}|mn\rangle. \end{aligned} \tag{3.23}$$

The second and fourth term of Eq. 3.23 vanish, since for the ansatz 3 projector, $\hat{Q}_{12}|ij\rangle = \hat{Q}_{12}|ab\rangle = 0$, while the third term is simplified by recognizing that $\hat{o}_1\hat{f}_2\hat{Q}_{12} = \hat{o}_2\hat{f}_1\hat{Q}_{12} = 0$:

$$\begin{aligned} B_{kl,mn} = &\langle kl|f_{12}\hat{f}_{12}\hat{Q}_{12}f_{12}|mn\rangle \\ &- \langle kl|f_{12}[\hat{o}_1\hat{f}_1 + \hat{o}_2\hat{f}_2]\hat{Q}_{12}f_{12}|mn\rangle. \end{aligned} \tag{3.24}$$

Under the GBC and EBC, $[\hat{f}_1 + \hat{f}_2, \hat{Q}_{12}] = 0$, so $\hat{o}_1\hat{f}_1\hat{Q}_{12} = \hat{f}_1\hat{o}_1\hat{Q}_{12} = 0$ and $\hat{o}_2\hat{f}_2\hat{Q}_{12} = \hat{f}_2\hat{o}_2\hat{Q}_{12} = 0$. Thus, the second term of Eq. 3.24 also vanishes under approximation 3*A. Inserting Eq. 3.15, applying the GBC, and neglecting the exchange part of the commutator (Eq. 3.18) then gives

$$B_{kl,mn} = \mathcal{A}_{kl,mn} + (\epsilon_k + \epsilon_l)X_{kl,mn}, \tag{3.25}$$

where

$$\mathcal{A}_{kl,mn} = \langle kl|[f_{12}, \hat{t}_1 + \hat{t}_2]\hat{Q}_{12}f_{12}|mn\rangle, \tag{3.26}$$

which differs from the more general $A_{kl,mn}$ matrix elements defined in Ref. 71.

To ensure hermiticity of the approximated **B** matrix, the matrix elements are symmetrized (see for example Refs. 70, 173), giving

$$B_{kl,mn} = \frac{1}{2}(\mathcal{A}_{kl,mn} + \mathcal{A}_{mn,kl}) + \frac{1}{2}(\epsilon_k + \epsilon_l + \epsilon_m + \epsilon_n)X_{kl,mn}. \tag{3.27}$$

Inserting the approximated $B_{kl,mn}$ (Eq. 3.27) into the F12 pair energy contribution (Eq. 3.6) yields

$$\begin{aligned} e_{ij}^{(\mathrm{F12})} = &\frac{1}{2}t_{kl}^{ij}(\mathcal{A}_{kl,mn} + \mathcal{A}_{mn,kl})t_{mn}^{ij} + 2t_{kl}^{ij}V_{kl}^{ij} \\ &+ \frac{1}{2}(\epsilon_k + \epsilon_l + \epsilon_m + \epsilon_n)t_{kl}^{ij}X_{kl,mn}t_{mn}^{ij} - (\epsilon_i + \epsilon_j)t_{kl}^{ij}X_{kl,mn}t_{mn}^{ij} \end{aligned} \tag{3.28}$$

where it is clear that the contribution from the **X** matrix cancels out under the diagonal ap-

proximation ($kl = ij$ or ji, $mn = ij$ or ji) and thus

$$e_{ij}^{(\mathrm{F12})} = \frac{1}{2} t_{kl}^{ij} (\mathcal{A}_{kl,mn} + \mathcal{A}_{mn,kl}) t_{mn}^{ij} + 2 t_{kl}^{ij} V_{kl}^{ij}. \tag{3.29}$$

As outlined above, MP2-F12/3*A(FIX) theory requires the evaluation of two new matrix elements, in addition to those required to evaluate the standard MP2 energy contribution: $\mathcal{A}_{kl,mn}$ (Eq. 3.26) and V_{kl}^{ij} (Eq. 3.11). Expanding the ansatz 3 projector (Eq. 3.21) in these matrix elements reveals the many-electron integral classes needed to evaluate the explicitly correlated part of the MP2-F12/3*A(FIX) pair energy (Eq. 3.29). For V_{kl}^{ij}, inserting Eq. 3.21 gives

$$\begin{aligned} V_{kl}^{ij} = {} & K_{ij,kl}^{F} + K_{mn}^{ij} F_{kl}^{mn} - K_{ab}^{ij} F_{kl}^{ab} \\ & - \langle ij | r_{12}^{-1} \hat{o}_1 f_{12} | kl \rangle - \langle ij | r_{12}^{-1} \hat{o}_2 f_{12} | kl \rangle \end{aligned} \tag{3.30}$$

with

$$K_{rs}^{ij} = \langle ij | r_{12}^{-1} | rs \rangle, \tag{3.31}$$

$$K_{ij,kl}^{F} = \langle ij | r_{12}^{-1} f_{12} | kl \rangle, \tag{3.32}$$

$$F_{kl}^{rs} = \langle rs | f_{12} | kl \rangle. \tag{3.33}$$

The fourth and fifth terms on the right of Eq. 3.30 are instances of the the three-electron integral type

$$v_{ij,kl} = \langle ijm | r_{12}^{-1} f_{23} | mlk \rangle \tag{3.34}$$

where summation over MO index m is implicit.

Expanding the ansatz 3 projector in $\mathcal{A}_{kl,mn}$ yields

$$\begin{aligned} \mathcal{A}_{kl,mn} = {} & U_{kl,mn}^{F} + U_{ij}^{kl} F_{mn}^{ij} - U_{ab}^{kl} F_{mn}^{ab} \\ & - \langle kl | [f_{12}, \hat{t}_1 + \hat{t}_2] \hat{o}_1 f_{12} | mn \rangle \\ & - \langle kl | [f_{12}, \hat{t}_1 + \hat{t}_2] \hat{o}_2 f_{12} | mn \rangle \end{aligned} \tag{3.35}$$

where

$$U_{kl,mn}^{F} = \langle kl | [f_{12}, \hat{t}_1 + \hat{t}_2] f_{12} | mn \rangle, \tag{3.36}$$

$$U_{rs}^{kl} = \langle kl | [f_{12}, \hat{t}_1 + \hat{t}_2] | rs \rangle, \tag{3.37}$$

and

$$\langle kl | [f_{12}, \hat{t}_1 + \hat{t}_2] \hat{o}_1 f_{12} | mn \rangle = \langle kli | [f_{12}, \hat{t}_1 + \hat{t}_2] f_{23} | inm \rangle, \tag{3.38}$$

$$\langle kl | [f_{12}, \hat{t}_1 + \hat{t}_2] \hat{o}_2 f_{12} | mn \rangle = \langle lki | [f_{12}, \hat{t}_1 + \hat{t}_2] f_{23} | imn \rangle. \tag{3.39}$$

The three-electron integrals arising from $\mathcal{A}_{kl,mn}$ (Eqs. 3.38 and 3.39) can be decomposed into

two distinct three-electron integral types:

$$a^{(1)}_{kl,mn} = \langle kli|[f_{12}, \hat{t}_1]f_{23}|inm\rangle, \tag{3.40}$$

$$a^{(2)}_{kl,mn} = \langle kli|[f_{12}, \hat{t}_2]f_{23}|inm\rangle. \tag{3.41}$$

The only many-electron integral classes required in MP2-F12 with the 3*A(FIX) approximation scheme theory are therefore the three-electron integrals $v_{ij,kl}$ (Eq. 3.34), $a^{(1)}_{kl,mn}$ and $a^{(2)}_{kl,mn}$ (Eqs. 3.40 and 3.41).

The relative simplicity of the 3*A(FIX) approximation scheme in terms of the number and complexity of many-electron integrals was the primary motivation for using MP2-F12/3*A(FIX) for our initial implementation of MP2-F12 theory with density-fitted many-electron integrals. Other MP2-F12 approximation schemes (outlined in Ref. 71) require the evaluation of additional many-electron integral types, including integrals over more than three electron coordinates, which would complicate the derivation of robust fitting expressions (section 3.2.2). Additionally, using a simpler MP2-F12 approximation scheme does not necessarily imply reduced accuracy—results published in Ref. 71 indicate that the MP2-F12/3*A(DX) method (where "DX" indicates the diagonal approximation with neglect of the $\mathbf{X}$ matrix) demonstrates surprising accuracy when compared to other approximation schemes with greater theoretical complexity, particularly in calculating energy differences (see in particular Table IX in Ref. 71).

3.2.2 Robust density-fitted three-electron integrals

To use density fitting (section 2.4) in place of resolutions of the identity in MP2-F12/3*A(FIX) theory, it is necessary to formulate density fitting expressions for three-electron MO integrals of the general form

$$\begin{aligned}\langle prt|\hat{v}_{12}\hat{w}_{23}|qsu\rangle &\equiv (pq|rs|tu) \\ &= \int \mathrm{d}\mathbf{r}_1\mathrm{d}\mathbf{r}_2\mathrm{d}\mathbf{r}_3\, \psi_p(\mathbf{r}_1)\psi_r(\mathbf{r}_2)\psi_t(\mathbf{r}_3)\hat{v}_{12}\hat{w}_{23}\psi_q(\mathbf{r}_1)\psi_s(\mathbf{r}_2)\psi_u(\mathbf{r}_3)\end{aligned} \tag{3.42}$$

where the $\hat{v}_{12}$ and $\hat{w}_{23}$ operators are assumed to commute with the MOs and are implicit in integral expressed in Mulliken notation [12, p.96], $(pq|rs|tu)$. It is tempting to fit all the orbital products simultaneously, in analogy to the four-index ERIs (Eq. 2.208), i.e.

$$(pq|rs|tu) \approx (\widetilde{pq}|\widetilde{rs}|\widetilde{tu}) = D^{pq}_A D^{rs}_B D^{tu}_C (A|B|C) \tag{3.43}$$

with the density fitting coefficients D^{pq}_A, D^{rs}_B and D^{tu}_C determined by minimization of the Coulomb energy of the fitting residual, as detailed in section 2.4. Unfortunately, this simple

fitting approach is non-robust, with the error in the fitted integral,

$$\begin{aligned}
&(pq|rs|tu) - (\widetilde{pq}|\widetilde{rs}|\widetilde{tu}) \\
&= (pq - \widetilde{pq}|rs - \widetilde{rs}|tu - \widetilde{tu}) + (pq|rs|\widetilde{tu}) + (pq|\widetilde{rs}|tu) \\
&\quad - (pq|\widetilde{rs}|\widetilde{tu}) + (\widetilde{pq}|rs|tu) - (\widetilde{pq}|rs|\widetilde{tu}) - (\widetilde{pq}|\widetilde{rs}|tu) \\
&= (pq - \widetilde{pq}|rs - \widetilde{rs}|tu - \widetilde{tu}) + (pq - \widetilde{pq}|\widetilde{rs}|tu) + (\widetilde{pq}|rs|tu - \widetilde{tu}) \\
&\quad + (pq|rs - \widetilde{rs}|\widetilde{tu})
\end{aligned} \tag{3.44}$$

containing terms that are linear in the error in the fitted orbital product. To ensure that the error in the fitted integral converges rapidly with the quality of the fit of the orbital products, it is important that there are no such linear terms in the error in the fitted integral, i.e. the fit is robust (section 2.4, see also Refs. 131, 148).

The terms linear in the error in the density-fit in Eq. 3.44 can be expressed using terms that are quadratic in the error in the density fit:

$$(pq|rs - \widetilde{rs}|\widetilde{tu}) = (pq - \widetilde{pq}|rs - \widetilde{rs}|\widetilde{tu}) + (\widetilde{pq}|rs|\widetilde{tu}) - (\widetilde{pq}|\widetilde{rs}|\widetilde{tu}), \tag{3.45}$$

$$(\widetilde{pq}|rs|tu - \widetilde{tu}) = (\widetilde{pq}|rs - \widetilde{rs}|tu - \widetilde{tu}) + (\widetilde{pq}|\widetilde{rs}|tu) - (\widetilde{pq}|\widetilde{rs}|\widetilde{tu}), \tag{3.46}$$

$$(pq - \widetilde{pq}|\widetilde{rs}|tu) = (pq - \widetilde{pq}|\widetilde{rs}|tu - \widetilde{tu}) + (pq|\widetilde{rs}|\widetilde{tu}) - (\widetilde{pq}|\widetilde{rs}|\widetilde{tu}). \tag{3.47}$$

Inserting Eqs. 3.45 to 3.47 into Eq. 3.44, and rearranging terms yields

$$\begin{aligned}
&(pq|rs|tu) - \{(pq|\widetilde{rs}|\widetilde{tu}) + (\widetilde{pq}|rs|\widetilde{tu}) + (\widetilde{pq}|\widetilde{rs}|tu) - 2(\widetilde{pq}|\widetilde{rs}|\widetilde{tu})\} \\
&= (pq - \widetilde{pq}|rs - \widetilde{rs}|tu - \widetilde{tu}) + (pq - \widetilde{pq}|rs - \widetilde{rs}|\widetilde{tu}) + (\widetilde{pq}|rs - \widetilde{rs}|tu - \widetilde{tu}) \\
&\quad + (pq - \widetilde{pq}|\widetilde{rs}|tu - \widetilde{tu}),
\end{aligned} \tag{3.48}$$

which contains no terms that are linear in the error in the density fit. Thus, a robust density fitting expression for $(pq|rs|tu)$ is (as suggested in Ref. 176):

$$\begin{aligned}
(pq|rs|tu)_{\text{robust}} &= (pq|\widetilde{rs}|\widetilde{tu}) + (\widetilde{pq}|rs|\widetilde{tu}) + (\widetilde{pq}|\widetilde{rs}|tu) - 2(\widetilde{pq}|\widetilde{rs}|\widetilde{tu}), \\
&= D_B^{rs} D_C^{tu} (pq|B|C) + D_A^{pq} D_C^{tu} (A|rs|C) \\
&\quad + D_A^{pq} D_B^{rs} (A|B|tu) - 2 D_A^{pq} D_B^{rs} D_C^{tu} (A|B|C).
\end{aligned} \tag{3.49}$$

This robust fitting expression has the significant advantage that only three- and four-index integrals are necessary to fit the six-index $(pq|rs|tu)$.

It is possible to envisage alternative robust density fitting expressions for $(pq|rs|tu)$, for example,

$$\begin{aligned}
(pq|rs|tu)_{\text{robust}} &= (pq|rs|\widetilde{tu}) + (pq|\widetilde{rs}|tu) - (pq|\widetilde{rs}|\widetilde{tu}) \\
&\quad + (\widetilde{pq}|rs|tu) - (\widetilde{pq}|rs|\widetilde{tu}) - (\widetilde{pq}|\widetilde{rs}|tu) + (\widetilde{pq}|\widetilde{rs}|\widetilde{tu})
\end{aligned} \tag{3.50}$$

which is obtained by simply subtracting the linear terms from the right side of Eq. 3.44 and has an error which is cubic in the error in the density fit. The error in Eq. 3.49 is quadratic in the error in the density fit, so the error in this alternative robust density fitting form would

be expected to converge even more rapidly with the quality of the density fit. However, fitting $(pq|rs|tu)$ using Eq. 3.50 would require five-index integrals like $(pq|rs|C)$, which would likely be very numerous. We therefore adopted Eq. 3.49, which requires only three- and four-index integrals, for robust density fitting the three-electron integrals that arise in MP2-F12/3*A(FIX) theory.

3.2.3 Density fitting in DF3-MP2-F12/3*A(FIX) theory

In DF3-MP2-F12/3*A(FIX) theory, there are three distinct types of three-electron integral that must approximated by density fitting: $v_{ij,kl}$, $a^{(1)}_{kl,mn}$ and $a^{(2)}_{kl,mn}$ (Eqs. 3.34, 3.40 and 3.41).

For the $v_{ij,kl}$ integrals (Eq. 3.34) the robust density fitting expression is trivially obtained by substitution of Eq. 3.49:

$$\begin{aligned} v^{\text{robust}}_{ij,kl} &= (im|r_{12}^{-1}|jl|f_{23}|mk)_{\text{robust}} \\ &= D^{jl}_B D^{mk}_C (im|r_{12}^{-1}|B|f_{23}|C) + D^{im}_A D^{mk}_C (A|r_{12}^{-1}|jl|f_{23}|C) \\ &\quad + D^{im}_A D^{jl}_B (A|r_{12}^{-1}|B|f_{23}|mk) - 2D^{im}_A D^{jl}_B D^{mk}_C (A|r_{12}^{-1}|B|f_{23}|C). \end{aligned} \tag{3.51}$$

The situation is more complicated for the $a^{(1)}_{kl,mn}$ and $a^{(2)}_{kl,mn}$ integrals, since the kinetic energy operators $\hat{t}_1$ and $\hat{t}_2$ do not commute with the MOs. Applying Eq. 3.49 to approximate these integral types requires further manipulation.

For the $a^{(1)}_{kl,mn}$ integrals (Eq. 3.40), expanding the commutator yields

$$\begin{aligned} a^{(1)}_{kl,mn} &= \langle kli|[f_{12}, \hat{t}_1] f_{23}|inm\rangle \\ &= \langle kli|f_{12}f_{23}|\{\hat{t}_1 i\}nm\rangle - \langle \{\hat{t}_1 k\}li|f_{12}f_{23}|inm\rangle \end{aligned} \tag{3.52}$$

which, in Mulliken notation is

$$a^{(1)}_{kl,mn} = -(\{\hat{t}_1 k\}i - k\{\hat{t}_1 i\}|f_{12}|ln|f_{23}|im) = -([ki]|f_{12}|ln|f_{23}|im). \tag{3.53}$$

with

$$|\{\hat{t}_1 p\}q - p\{\hat{t}_1 q\}) \equiv |[pq]). \tag{3.54}$$

As noted by Manby in his work on DF-MP2-R12 [131], $|[pq])$ can be treated like a standard orbital product, with density fitting coefficients $D_A^{[pq]}$ determined by minimizing the Coulomb criterion for the fitting residual $|[pq]) - \widetilde{[pq]})$:

$$\Delta_{[pq]} = ([pq] - \widetilde{[pq]}|r_{12}^{-1}|[pq] - \widetilde{[pq]}), \tag{3.55}$$

where

$$|\widetilde{[pq]}) = D_A^{[pq]}|A). \tag{3.56}$$

Minimizing $\Delta_{[pq]}$ using the procedure described in section 2.4 yields the following expression for the commutator product fitting coefficients:

$$D_A^{[pq]} = [\mathbf{J}^{-1}]_{AB} Y^B_{pq} \tag{3.57}$$

where $J_{AB} = (A|r_{12}^{-1}|B)$ and

$$\begin{aligned} Y_{pq}^{B} &= (B|r_{12}^{-1}|[pq]) = (B|[\hat{t}_2, r_{12}^{-1}]|pq) \\ &= \int \mathrm{d}\mathbf{r}_1 \mathrm{d}\mathbf{r}_2 \, \varphi_B^*(\mathbf{r}_1)\psi_p^*(\mathbf{r}_2)[\hat{t}_2, r_{12}^{-1}]\psi_q(\mathbf{r}_2). \end{aligned} \tag{3.58}$$

A method for evaluating the primitive integral corresponding to Y_{pq}^{B} using primitive three-index overlap and Coulomb integrals is described in Ref. 131.

Eq. 3.49 can now be applied in a straightforward manner to give the robust density fitting expression for $a_{kl,mn}^{(1)}$:

$$\begin{aligned} a_{kl,mn}^{(1),\mathrm{robust}} &= -([ki]|f_{12}|ln|f_{23}|im)_{\mathrm{robust}} \\ &= -D_B^{ln} D_C^{im}([ki]|f_{12}|B|f_{23}|C) - D_A^{[ki]} D_C^{im}(A|f_{12}|ln|f_{23}|C) \\ &\quad - D_A^{[ki]} D_B^{ln}(A|f_{12}|B|f_{23}|im) + 2D_A^{[ki]} D_B^{ln} D_C^{im}(A|f_{12}|B|f_{23}|C). \end{aligned} \tag{3.59}$$

where the commutator product fitting coefficients $D_A^{[ki]}$ are determined as described in Eqs. 3.55 to 3.57.

Unfortunately, the same approach cannot be applied to the $a_{kl,mn}^{(2)}$ integrals (Eq. 3.41), since expanding the commutator gives

$$\begin{aligned} a_{kl,mn}^{(2)} &= \langle kli|[f_{12}, \hat{t}_2]f_{23}|inm\rangle \\ &= \langle kli|f_{12}\hat{t}_2 f_{23}|inm\rangle - \langle kli|\hat{t}_2 f_{12} f_{23}|inm\rangle. \end{aligned} \tag{3.60}$$

This differs from the situation for the $a_{kl,mn}^{(1)}$ integrals (Eq. 3.40), where the $\hat{t}_1$ operator can act directly on the MOs (see Eq. 3.52)—in Eq. 3.60 the $\hat{t}_2$ operator is "trapped" between f_{12} and f_{23}. The strategy of "wrapping up" the kinetic energy operator in a fitted commutator product density (Eqs. 3.54 to 3.57), cannot therefore be applied here.

To convert the $a_{kl,mn}^{(2)}$ integrals into a form suitable for robust density fitting using Eq. 3.49, we take a different approach, which makes use the following identity:

$$\begin{aligned} \hat{t}_2 f_{23}\phi_n(\mathbf{r}_2) &= -\frac{1}{2}\nabla_2 \cdot \nabla_2 f_{23}\phi_n(\mathbf{r}_2) \\ &= -\frac{1}{2}\left(f_{23}\{\nabla_2^2\phi_n(\mathbf{r}_2)\} + 2(\nabla_2 f_{23}) \cdot (\nabla_2\phi_n(\mathbf{r}_2)) + \phi_n(\mathbf{r}_2)\{\nabla_2^2 f_{23}\}\right). \end{aligned} \tag{3.61}$$

Inserting this into Eq. 3.60 gives

$$\begin{aligned} a_{kl,mn}^{(2)} &= \langle kli|f_{12}f_{23}|i\{\hat{t}_2 n\}m\rangle - \langle kli|f_{12}(\nabla_2 f_{23})|i(\nabla_2 n)m\rangle \\ &\quad - \frac{1}{2}\langle kli|f_{12}\{\nabla_2^2 f_{23}\}|inm\rangle - \langle k\{\hat{t}_2 l\}i|f_{12}f_{23}|inm\rangle \end{aligned} \tag{3.62}$$

which, using the definition of the commutator product density (Eq. 3.54), becomes

$$\begin{aligned} a_{kl,mn}^{(2)} &= -(ki|f_{12}|[ln]|f_{23}|im) - (ki|f_{12}|l(\nabla_2 n)|(\nabla_2 f_{23})|im) \\ &\quad - \frac{1}{2}(ki|f_{12}|ln|\{\nabla_2^2 f_{23}\}|im). \end{aligned} \tag{3.63}$$

where we have used the following notation for the second term on the right of Eq. 3.63:

$$\begin{aligned}&\langle kli|f_{12}(\nabla_2 f_{23})|i(\nabla_2 n)m\rangle \equiv (ki|f_{12}|l(\nabla_2 n)|(\nabla_2 f_{23})|im)\\&= \int \mathrm{d}\mathbf{r}_1\mathrm{d}\mathbf{r}_2\mathrm{d}\mathbf{r}_3\ \{\phi_k^*(\mathbf{r}_1)\phi_l^*(\mathbf{r}_2)\phi_i^*(\mathbf{r}_3)f_{12}(\nabla_2 f_{23})\cdot(\nabla_2\phi_n(\mathbf{r}_2))\phi_i(\mathbf{r}_1)\phi_m(\mathbf{r}_3)\}\,.\end{aligned} \tag{3.64}$$

Each of the three integrals on the right of Eq. 3.63 can be approximated by robust density fitting using Eq. 3.49. Robust density fitting the first term gives

$$\begin{aligned}&(ki|f_{12}|[ln]|f_{23}|im)_{\text{robust}}\\&= D_B^{[ln]}D_C^{im}(ki|f_{12}|B|f_{23}|C) + D_A^{ki}D_C^{im}(A|f_{12}|[ln]|f_{23}|C)\\&\quad + D_A^{ki}D_B^{[ln]}(A|f_{12}|B|f_{23}|im) - 2D_A^{ki}D_B^{[ln]}D_C^{im}(A|f_{12}|B|f_{23}|C),\end{aligned} \tag{3.65}$$

where the commutator product fitting coefficients, $D_B^{[ln]}$ are determined using Eqs. 3.56 and 3.57.

For the second term, the dot product sums over Cartesian components, i.e.

$$(ki|f_{12}|l(\nabla_2 n)|(\nabla_2 f_{23})|im) = \sum_{\lambda=x,y,z}(ki|f_{12}|ln_\lambda|f_{2_\lambda 3}|im) \tag{3.66}$$

where we have introduced the following concise notation for partial derivatives:

$$(ki|f_{12}|ln_\lambda|f_{2_\lambda 3}|im) \equiv (ki|f_{12}|l(\nabla_{2\lambda} n)|(\nabla_{2\lambda} f_{23})|im) \tag{3.67}$$

with $\nabla_{2\lambda} \equiv \partial/\partial r_{2\lambda}$. The robust density fitting expression for each Cartesian component of Eq. 3.66 is

$$\begin{aligned}&(ki|f_{12}|ln_\lambda|f_{2_\lambda 3}|im)_{\text{robust}}\\&= D_B^{ln_\lambda}D_C^{im}(ki|f_{12}|B|f_{2_\lambda 3}|C) + D_A^{ki}D_C^{im}(A|f_{12}|ln_\lambda|f_{2_\lambda 3}|C)\\&\quad + D_A^{ki}D_B^{ln_\lambda}(A|f_{12}|B|f_{2_\lambda 3}|im) - 2D_A^{ki}D_B^{ln_\lambda}D_C^{im}(A|f_{12}|B|f_{2_\lambda 3}|C).\end{aligned} \tag{3.68}$$

To fit the gradient product density, $|ln_\lambda)$, in Eq. 3.68 another type of density fitting coefficient is needed, $D_B^{ln_\lambda}$. As for the standard density fitting coefficients (Eq. 2.199), and the commutator product fitting coefficients (Eq. 3.56), the gradient product fitting coefficients,

$$|\widetilde{pq_\lambda}) = D_A^{pq_\lambda}|A) \tag{3.69}$$

are determined by minimizing the Coulomb energy of the fitting residual,

$$\Delta_{pq_\lambda} = (pq_\lambda - \widetilde{pq_\lambda}|r_{12}^{-1}|pq_\lambda - \widetilde{pq_\lambda}), \tag{3.70}$$

which gives the following expression for the gradient product coefficients:

$$D_A^{pq_\lambda} = [\mathbf{J}^{-1}]_{AB}J_{pq_\lambda}^B, \tag{3.71}$$

where

$$\begin{aligned} J^B_{pq_\lambda} &= (B|r_{12}^{-1}|p(\nabla_{2\lambda}q)) \\ &= \int \mathrm{d}\mathbf{r}_1\mathrm{d}\mathbf{r}_2\, \varphi^*_B(\mathbf{r}_1)\psi^*_p(\mathbf{r}_2) r_{12}^{-1} \frac{\partial}{\partial r_{2\lambda}} \psi_q(\mathbf{r}_2). \end{aligned} \tag{3.72}$$

and $\lambda = x, y, z$. The $(\mathbf{a}|r_{12}^{-1}|\mathbf{bc}_\lambda)$ primitive integrals used to construct $J^B_{pq_\lambda}$ may be evaluated simply by applying the differential operator (Eq. 2.152), to yield an expression in terms of standard $(\mathbf{a}|r_{12}^{-1}|\mathbf{bc})$ integrals with incremented and decremented angular momentum in $|\mathbf{c})$.

Finally, the third term on the right of Eq. 3.63 can be approximated by robust density fitting using only the standard density fitting coefficients:

$$\begin{aligned} &(ki|f_{12}|ln|\{\nabla_2^2 f_{23}\}|im)_{\mathrm{robust}} \\ &= D^{ln}_B D^{im}_C (ki|f_{12}|B|\{\nabla_2^2 f_{23}\}|C) + D^{ki}_A D^{im}_C (A|f_{12}|ln|\{\nabla_2^2 f_{23}\}|C) \\ &\quad + D^{ki}_A D^{ln}_B (A|f_{12}|B|\{\nabla_2^2 f_{23}\}|im) - 2 D^{ki}_A D^{ln}_B D^{im}_C (A|f_{12}|B|\{\nabla_2^2 f_{23}\}|C). \end{aligned} \tag{3.73}$$

The robust density fitting expression for $a^{(2)}_{kl,mn}$ integrals can therefore be obtained by separately fitting each of the terms in Eq. 3.63 using Eqs. 3.65, 3.68 and 3.73, i.e.

$$\begin{aligned} a^{(2),\mathrm{robust}}_{kl,mn} = &-(ki|f_{12}|[ln]|f_{23}|im)_{\mathrm{robust}} - \sum_{\lambda=x,y,z} (ki|f_{12}|ln_\lambda|f_{2_\lambda 3}|im)_{\mathrm{robust}} \\ &- \frac{1}{2}(ki|f_{12}|ln|\{\nabla_2^2 f_{23}\}|im)_{\mathrm{robust}}. \end{aligned} \tag{3.74}$$

In this chapter, we will also consider non-robust density fitting of the three-electron integrals, where terms linear in the error in the fit are allowed to remain in the expression for the error in the fitted integral. The non-robust fitting forms used are based on the simple approach described in Eq. 3.43, and are as follows. For $v_{ij,kl}$ (Eq. 3.34):

$$\begin{aligned} v^{\mathrm{DF}}_{ij,kl} &= (\widetilde{im}|r_{12}^{-1}|\widetilde{jl}|f_{23}|\widetilde{mk}) \\ &= D^{im}_A D^{jl}_B D^{mk}_C (A|r_{12}^{-1}|B|f_{23}|C). \end{aligned} \tag{3.75}$$

For $a^{(1)}_{kl,mn}$ (Eq. 3.40):

$$\begin{aligned} a^{(1),\mathrm{DF}}_{kl,mn} &= -(\widetilde{[ki]}|f_{12}|\widetilde{ln}|f_{23}|\widetilde{im}) \\ &= D^{[ki]}_A D^{ln}_B D^{im}_C (A|f_{12}|B|f_{23}|C). \end{aligned} \tag{3.76}$$

For $a^{(2)}_{kl,mn}$ (Eq. 3.41):

$$\begin{aligned} a^{(2),\mathrm{DF}}_{kl,mn} &= -(\widetilde{ki}|f_{12}|\widetilde{[ln]}|f_{23}|\widetilde{im}) - \sum_{\lambda=x,y,z} (\widetilde{ki}|f_{12}|\widetilde{ln_\lambda}|f_{2_\lambda 3}|\widetilde{im}) \\ &\quad - \frac{1}{2}(\widetilde{ki}|f_{12}|\widetilde{ln}|\{\nabla^2_2 f_{23}\}|\widetilde{im}), \\ &= -D^{ki}_A D^{[ln]}_B D^{im}_C (A|f_{12}|B|f_{23}|C) \\ &\quad - \sum_{\lambda=x,y,z} D^{ki}_A D^{ln_\lambda}_B D^{im}_C (A|f_{12}|B|f_{2_\lambda 3}|C) \\ &\quad - D^{ki}_A D^{ln}_B D^{im}_C (A|f_{12}|B|\{\nabla^2_2 f_{23}\}|C). \end{aligned} \tag{3.77}$$

The relative simplicity of the preceding non-robust fitting expressions makes them worthy of consideration, despite the larger error in the fitted integrals that would be expected to result from their use.

In section 2.4, we considered the angular momentum required in a density fitting basis used to fit a product of one-electron functions. For a single-centre product of Gaussian-type orbitals |ab), this is $\ell_a + \ell_b$—for primitive Cartesian Gaussian functions, this requirement is obvious from the GPT (appendix B). For a general single-centre product of functions with well-defined angular momentum, the requirement is indicated by the rules of coupled angular momentum states. The permitted total angular momentum values for coupled angular momentum states are given by the Clebsch-Gordan series (see Ref. 5 (pp.112–120) for details), i.e.

$$\ell_{\mathrm{tot}} = \ell_a + \ell_b, \ell_a + \ell_b - 1, \ldots, |\ell_a - \ell_b|. \tag{3.78}$$

Consequently, a maximum of $\ell_a + \ell_b$ units of angular momentum are required in the auxiliary basis used to fit a such a product of functions.

In DF3-MP2-F12/3*A(FIX) theory, products of occupied molecular orbitals, $|ij)$ are density-fitted. In the case of an atom, where molecular orbitals have well-defined ℓ, and are constructed from atomic orbitals with solid harmonic angular components, the maximum angular momentum required in a fitting basis for the product $|ij)$ is $2\ell_{\mathrm{occ}}$ (ℓ_{occ} is the maximum angular momentum of the occupied orbitals). This is lower than the $3\ell_{\mathrm{occ}}$ required in the RI basis for approximating three-electron integrals using resolutions of the identity (see section 3.2.4), and is one of the main motivations behind pursuing density fitting as an alternative to RIs for many-electron integrals.

In order to approximate commutator and gradient orbital products, $|[ij])$ and $|ij_\lambda)$ (Eqs. 3.56 and 3.69) by density fitting, the angular momentum required in the fitting basis set may be higher—in the case of Cartesian Gaussian functions, taking the derivative of the function results in terms with total angular momentum increased by one unit (Eq. 2.152). The question of the angular momentum required fit these more unusual charge densities is addressed in section 3.4 in light of numerical results obtained for DF3-MP2-F12/3*A(FIX) calculations.

3.2.4 Resolution of the identity

To test the use of density fitting to approximate three-electron integrals in MP2-F12/3*A(FIX) theory, it was important to compare against the standard resolution of the identity approaches used in R12/F12 methods. In this section, we will consider the theoretical footing of the standard RI approach and outline the variants used in comparisons with the density fitting approach described in section 3.2.3.

A general resolution of the identity for a three-electron integral has the form

$$\begin{aligned}\langle ijk|\hat{u}_{12}\hat{v}_{23}|nml\rangle &\approx \langle ijk|\hat{u}_{12}\hat{X}_2\hat{v}_{23}|nml\rangle \\ &= \langle ij|\hat{u}_{12}|nx\rangle\langle kx|\hat{v}_{12}|lm\rangle\end{aligned} \tag{3.79}$$

where $\hat{u}_{12}$ and $\hat{v}_{23}$ are two-electron operators (e.g. r_{12}^{-1}, f_{12}) and $\hat{X}_2 = |x\rangle\langle x|$ is an approximation to the identity operator in the orthonormal RI basis $\{|x\rangle\}$. The approximate identity operator, $\hat{X}_2$, is a projection operator, acting on electron 2 and may be represented as follows:

$$\hat{X}_2 = \sum_x |x\rangle\langle x| = \sum_x \psi_x(\mathbf{r}_2)\int \mathrm{d}\mathbf{r}_k \psi_x^*(\mathbf{r}_k)\hat{\pi}_{2k} \tag{3.80}$$

where $\{\psi_x\}$ is the RI basis set and $\hat{\pi}_{2k}$ is an operator that permutes electron 2 and electron k. When the approximate identity operator is represented in this way, the nature of the operation occurring in Eq. 3.79 is revealed:

$$\begin{aligned}&\langle ijk|\hat{u}_{12}\hat{X}_2\hat{v}_{23}|nml\rangle \\ &\equiv \int \mathrm{d}\mathbf{r}_1\mathrm{d}\mathbf{r}_2\mathrm{d}\mathbf{r}_3\, \psi_i^*(\mathbf{r}_1)\psi_j^*(\mathbf{r}_2)\psi_k^*(\mathbf{r}_3)\hat{u}_{12}\hat{X}_2\hat{v}_{23}\psi_n(\mathbf{r}_1)\psi_m(\mathbf{r}_2)\psi_l(\mathbf{r}_3) \\ &= \sum_x \int \mathrm{d}\mathbf{r}_1\mathrm{d}\mathbf{r}_2\mathrm{d}\mathbf{r}_3\mathrm{d}\mathbf{r}_4\, \psi_i^*(\mathbf{r}_1)\psi_j^*(\mathbf{r}_2)\psi_k^*(\mathbf{r}_3)\hat{u}_{12}\psi_x(\mathbf{r}_2)\psi_x^*(\mathbf{r}_4)\hat{\pi}_{24}\hat{v}_{23} \\ &\qquad \times \psi_n(\mathbf{r}_1)\psi_m(\mathbf{r}_2)\psi_l(\mathbf{r}_3) \\ &= \sum_x \int \mathrm{d}\mathbf{r}_1\mathrm{d}\mathbf{r}_2\mathrm{d}\mathbf{r}_3\mathrm{d}\mathbf{r}_4\, \psi_i^*(\mathbf{r}_1)\psi_j^*(\mathbf{r}_2)\psi_k^*(\mathbf{r}_3)\psi_x^*(\mathbf{r}_4)\hat{u}_{12}\hat{v}_{34} \\ &\qquad \times \psi_n(\mathbf{r}_1)\psi_x(\mathbf{r}_2)\psi_l(\mathbf{r}_3)\psi_m(\mathbf{r}_4) \\ &= \sum_x \int \mathrm{d}\mathbf{r}_1\mathrm{d}\mathbf{r}_2\, \psi_i^*(\mathbf{r}_1)\psi_j^*(\mathbf{r}_2)\hat{u}_{12}\psi_n(\mathbf{r}_1)\psi_x(\mathbf{r}_2) \\ &\qquad \times \int \mathrm{d}\mathbf{r}_1\mathrm{d}\mathbf{r}_2\, \psi_k^*(\mathbf{r}_1)\psi_x^*(\mathbf{r}_2)\hat{v}_{12}\psi_l(\mathbf{r}_1)\psi_m(\mathbf{r}_2).\end{aligned} \tag{3.81}$$

In Ref. 131, the resolution of the identity used to approximate many-electron integrals in R12/F12 methods is considered in terms of a fitting procedure, where three-index objects are approximated by robust density fitting in an orthonormal RI basis. This analysis is reproduced and expanded upon in appendix E.

It is useful to consider the quantity of angular momentum needed in the RI basis to ensure an accurate fit, as for the density fitting of orbital products (section 3.2.3). To rigorously determine the angular momentum required in the RI basis for approximating integrals with arbitrary operators $\hat{u}_{12}$, $\hat{v}_{23}$, a partial wave expansion can be employed [68]. The partial wave

analysis is lengthy and complicated, so we will simply state the relevant result here (see the appendix of Ref. 68 for details): to approximate the three-electron integral

$$\langle pqt|\hat{u}_{12}\hat{v}_{23}|tsr\rangle \approx \langle pq|\hat{u}_{12}|tx\rangle\langle tx|\hat{v}_{12}|rs\rangle \tag{3.82}$$

where the operators are spherically symmetric (as is the case for r_{12}^{-1} and f_{12}) the angular momentum required in the RI basis, $\{|x\rangle\}$ is at most $3\ell_{\mathrm{max}}$ (for the single centre case, where ℓ_{max} is the maximum angular momentum in the basis $\{|p\rangle, |q\rangle, |r\rangle, |s\rangle, \ldots\}$, see also appendix A in Ref. 183). In MP2-F12/3*A(FIX) theory, the $v_{ij,kl}$ integrals (Eq. 3.34) can therefore be approximated using an RI basis with $3\ell_{\mathrm{occ}}$ units of angular momentum (in the single centre case, where ℓ_{occ} is the maximum angular momentum of the occupied MOs).

For other many-electron integral types, the angular momentum requirement may be higher, or the partial wave expansion may not truncate at any ℓ value, in which case the RI basis can never be formally "saturated" in terms of angular momentum (these integrals may still converge quickly with angular momentum, however—see the appendix of Ref. 68). The situation is further complicated by the introduction of multiple centres, particularly because the RI basis is, for convenience, typically constructed from atom-centred functions. The $3\ell_{\mathrm{occ}}$ requirement for the RI basis should therefore be treated as a guide, rather than a rigorous bound. Nevertheless, this result illustrates the problematic angular momentum requirements imposed on the RI basis in explicitly correlated R12/F12 calculations.

As mentioned in the introduction to this chapter, the RI+DF method of Ten-no and Manby [175] employs density fitting in an approach which reduces the angular momentum required in the RI basis set. The key to this is the realization that, for a local operator (which commutes with the orbitals, e.g. r_{12}^{-1}), an orbital can be moved from the ket to the bra of a three-electron integral, e.g. for $v_{ij,kl}$ (Eq. 3.34):

$$\langle ijm|r_{12}^{-1}f_{23}|mlk\rangle = \langle i(jl)m|r_{12}^{-1}f_{23}|m1k\rangle. \tag{3.83}$$

Applying an approximate resolution of the identity gives

$$\langle ijm|r_{12}^{-1}f_{23}|mlk\rangle \approx \sum_{x}\langle i(jl)|r_{12}^{-1}|mx\rangle F_k^{mx}, \tag{3.84}$$

and density fitting the orbital product $|jl)$ to avoid the five-index integral yields

$$\langle ijm|r_{12}^{-1}f_{23}|mlk\rangle \approx \sum_{x}\sum_{A} D_A^{jl}\langle iA|r_{12}^{-1}|mx\rangle F_k^{mx} \tag{3.85}$$

where

$$F_k^{mx} = \langle mx|f_{12}|k1\rangle \equiv (mk|f_{12}|x).$$

Since F_k^{mx} only contains two occupied orbital indexes, $2\ell_{\mathrm{occ}}$ units of angular momentum are required in the RI basis rather than the $3\ell_{\mathrm{occ}}$ required in the standard RI approach (for the single-centre case).[1] For integrals containing the non-local kinetic energy commutator, $[f_{12}, \hat{t}_1 + \hat{t}_2]$,

[1] In terms of the fitting procedure described in appendix E, the RI+DF method is equivalent to using the RI basis to fit a two-index object, rather than a three-index object.

further manipulation is required—see Refs. 56, 175 for details.

In early implementations of the R12/F12 methods, the MO basis set was also used as an RI basis (see for example Ref. 68), placing problematic angular momentum requirements on the MO basis set (and by extension, the AO basis set in which the MOs are expanded). The development of the auxiliary basis set (ABS) approach [173], where the RI basis is separate to the MO basis, was a significant development, isolating the MO basis (and AO basis) from the angular momentum requirements for the RIs. Using the ABS RI approach, the RI basis can be constructed to minimize the error from the RI approximation.

A key requirement for the RI basis is that it spans the MO basis set, i.e.

$$\hat{X}_1 \hat{p}_1 = \hat{p}_1 \tag{3.86}$$

where $\hat{X}_1 = |x\rangle\langle x|$ is the approximate identity operator expanded in the RI basis and $\hat{p}_1 = |p\rangle\langle p|$ is a projection operator onto the MO space. This can be understood in terms of the strong orthogonality projector (e.g. Eq. 3.12), which may be expressed as

$$\hat{Q}_{12} = 1 - \hat{o}_1(1 - \hat{p}_2) - (1 - \hat{p}_1)\hat{o}_2 - \hat{p}_1\hat{p}_2, \tag{3.87}$$

where $\hat{p}_1 = \hat{o}_1 + \hat{v}_1$ and $(1 - \hat{p}_1)$ is the "orthogonal complement" to the MO basis set, such that

$$\hat{p}_1(1 - \hat{p}_1) = 0. \tag{3.88}$$

Approximating the three-electron integrals in MP2-F12/3*A(FIX) theory (Eqs. 3.34, 3.40 and 3.41) using RIs is equivalent to using the modified projector

$$\hat{Q}_{12}^{\mathrm{RI}} = 1 - \hat{o}_1(\hat{X}_2 - \hat{p}_2) - (\hat{X}_1 - \hat{p}_1)\hat{o}_2 - \hat{p}_1\hat{p}_2, \tag{3.89}$$

which demonstrates that the insertion of approximate identity operators $\hat{X}_1$, $\hat{X}_2$ approximates the orthogonal complement to the MO basis (Eq. 3.88). In order for Eq. 3.88 to hold for the approximated orthogonal complement operator, $(\hat{X}_1 - \hat{p}_1)$, the RI basis should span the MO basis [56]—for the ABS RI approach, unlike the earlier approach of using the MO basis for the RIs, this is not guaranteed.

The formulation of the strong orthogonality projector in Eq. 3.87 suggests an alternative RI approach—approximating the orthogonal complement operator, $(1 - \hat{p}_1)$, directly. This is the complementary auxiliary basis set (CABS) approach [174] in which the identity is approximated as

$$\hat{p}_1 + \hat{X}'_1 \approx 1 \tag{3.90}$$

where

$$\hat{X}' = |x_\perp\rangle\langle x_\perp| \approx (1 - \hat{p}_1) \tag{3.91}$$

with orthogonal complement (CABS RI) basis set $\{|x_\perp\rangle\}$. The CABS RI basis is constructed by taking a standard ABS basis set and orthogonalizing it with respect to the MO basis, while ensuring that the basis functions remain mutually orthonormal.

For a given auxiliary basis set, the CABS RI approach yields smaller errors than the ABS

RI approach, and is therefore generally the preferred approach [174]. The CABS approach has the additional advantage that the CABS RI basis set can also be used to perturbatively correct the Hartree-Fock basis set error in the "CABS singles" correction [80, 169, 170].

In section 3.4, we compare the ABS and CABS RI approaches for approximating many-electron integrals to the robust and non-robust density fitting approaches described in section 3.2.3. The application of the ABS and CABS RI approaches in MP2-F12/3*A(FIX) theory involves the approximation of the three-electron integrals arising from V_{kl}^{ij} (Eq. 3.30) and $\mathcal{A}_{kl,mn}$ (Eq. 3.35). As mentioned above, this is equivalent to using an approximated strong orthogonality projector (Eq. 3.87), i.e. for the ansatz 3 projector and ABS RI

$$\hat{Q}_{12}^{\mathrm{ABS}} = 1 + \hat{o}_1\hat{o}_2 - \hat{o}_1\hat{X}_2 - \hat{X}_1\hat{o}_2 - \hat{v}_1\hat{v}_2. \tag{3.92}$$

The CABS RI projector is obtained by setting $\hat{X}_1 = \hat{X'}_1 + \hat{p}_1$, i.e.

$$\hat{Q}_{12}^{\mathrm{CABS}} = 1 + \hat{o}_1\hat{o}_2 - \hat{o}_1\hat{X'}_2 - \hat{o}_2\hat{X'}_1 + \hat{o}_1\hat{p}_2 + \hat{o}\hat{p}_1 - \hat{v}_1\hat{v}_2 \tag{3.93}$$

which, using $\hat{p} = \hat{o} + \hat{v}$, simplifies to

$$\hat{Q}_{12}^{\mathrm{CABS}} = 1 - o_1\hat{X'}_2 - \hat{X'}_1\hat{o}_2 - \hat{p}_1\hat{p}_2. \tag{3.94}$$

Inserting the approximate strong orthogonality projectors into the definition for V_{kl}^{ij} (Eq. 3.11) we obtain

$$V_{kl}^{ij} = K_{ij,kl}^{F} + K_{mn}^{ij}F_{kl}^{mn} - K_{mx}^{ij}F_{kl}^{mx} - K_{xm}^{ij}F_{kl}^{xm} - K_{ab}^{ij}F_{kl}^{ab} \tag{3.95}$$

for the ABS projector (Eq. 3.92), and

$$V_{kl}^{ij} = K_{ij,kl}^{F} - K_{mx'}^{ij}F_{kl}^{mx'} - K_{x'm}^{ij}F_{kl}^{x'm} - K_{rs}^{ij}F_{kl}^{rs} \tag{3.96}$$

for the CABS projector (Eq. 3.94), where the various component integrals are defined in Eqs. 3.31 to 3.33, $\{x\}$ is the ABS RI basis set, and $\{x'\}$ is the orthogonal complement (CABS RI) basis set ($\{|x_\perp\rangle\}$ in Eq. 3.91). Similarly, for $\mathcal{A}_{kl,mn}$ (Eq. 3.26), inserting the ABS projector (Eq. 3.92) yields

$$\mathcal{A}_{kl,mn} = U_{kl,mn}^{F} + U_{ij}^{kl}F_{mn}^{ij} - U_{ix}^{kl}F_{mn}^{ix} - U_{xi}^{kl}F_{mn}^{xi} - U_{ab}^{kl}F_{mn}^{ab} \tag{3.97}$$

while inserting the CABS projector (Eq. 3.94) gives

$$\mathcal{A}_{kl,mn} = U_{kl,mn}^{F} - U_{ix'}^{kl}F_{mn}^{ix'} - U_{x'i}^{kl}F_{mn}^{x'i} - U_{rs}^{kl}F_{mn}^{rs} \tag{3.98}$$

where the component two-electron integrals are defined in Eqs. 3.36 and 3.37.

3.3 Implementation

3.3.1 Computational details

The DF3-MP2-F12/3*A(FIX) method (as described in section 3.2) was implemented in Molpro [151], a quantum chemistry software package with established implementations of explicitly correlated methods, including a number of MP2-F12 variants (see Ref. 152 and the Molpro user manual[2] for details).

In the course of this implementation, new code for the evaluation of the three-electron integrals required in DF3-MP2-F12/3*A(FIX) theory was added to Molpro. In particular, code was written for the evaluation of several new types of integrals over primitive Cartesian Gaussians (section 3.3.2) and the contraction of these integrals with density fitting coefficients and MO coefficients (section 3.3.3). The existing MP2-F12 code [70, 71, 81, 131, 163] in Molpro was modified to allow use of density fitting in the place of RIs for the approximation of the three-electron integrals arising from the V_{kl}^{ij} and $\mathcal{A}_{kl,mn}$ matrix elements (Eqs. 3.11 and 3.26).

The work described in the following sections represents a "proof-of-concept" implementation of the DF3 density fitting approach in MP2-F12 theory, intended to test the numerical behaviour of the method, rather than demonstrate computational efficiency. To expedite the development process, some choices were made that favoured simplicity of implementation over efficiency. For example, to avoid the complicated and time-consuming process of implementing Obara-Saika-type recurrence relations (section 2.3.3) for the three-electron integrals over primitive Cartesian Gaussians (as derived in Ref. 89 and reproduced in appendix C), a simpler, but relatively inefficient method combining the Gaussian product theorem and a Gaussian transform was employed (see section 3.3.2 for a description of this technique, and chapter 4 for details of a more efficient implementation using automatic code generation). In the following, we will consider only the theoretical cost of the method, since real-world measurements of the computational performance would likely not be representative of the behaviour of the method in a performance-oriented implementation.

All the code added to Molpro for this implementation was written for serial execution, and all results quoted in this chapter are for Molpro (development version) running in serial on a single x86 CPU core.

3.3.2 Primitive integrals

In section 2.3 the evaluation of integrals over Gaussian-type orbitals (GTOs, section 2.3.1) starting from integrals over primitive Cartesian Gaussians—"primitive integrals"—was described. In this section, we will consider the various types of primitive integral that needed implementing in Molpro [151, 152] in order that the density-fitted forms of $v_{ij,kl}$, $a_{kl,mn}^{(1)}$ and $a_{kl,mn}^{(2)}$ integrals (Eqs. 3.34, 3.40 and 3.41) could be evaluated.

The robust density fitting expressions for the three-electron integrals (Eqs. 3.51, 3.59 and 3.74) contain integrals over MO and density fitting (DF) functions, which may be obtained by contraction of primitive integrals with the correct coefficients. For example, to evaluate $v_{ij,kl}^{\mathrm{robust}}$

[2]The online user manual can be found at `http://www.molpro.net`.

Table 3.1: Integrals over primitive Cartesian Gaussian functions required to evaluate $v_{ij,kl}^{\text{robust}}$, $a_{kl,mn}^{(1),\text{robust}}$ and the components of $a_{kl,mn}^{(2),\text{robust}}$, where $\lambda = x, y, z$.

Label	Definition
J-F	$(\mathbf{a}\|r_{12}^{-1}\|\mathbf{b}\|f_{23}\|\mathbf{c})$
	$(\mathbf{ab}\|r_{12}^{-1}\|\mathbf{c}\|f_{23}\|\mathbf{d})$
	$(\mathbf{a}\|r_{12}^{-1}\|\mathbf{bc}\|f_{23}\|\mathbf{d})$
	$(\mathbf{a}\|r_{12}^{-1}\|\mathbf{b}\|f_{23}\|\mathbf{cd})$
F-F	$(\mathbf{a}\|f_{12}\|\mathbf{b}\|f_{23}\|\mathbf{c})$
	$(\mathbf{ab}\|f_{12}\|\mathbf{c}\|f_{23}\|\mathbf{d})$
	$(\mathbf{a}\|f_{12}\|\mathbf{bc}\|f_{23}\|\mathbf{d})$
	$(\mathbf{a}\|f_{12}\|\mathbf{b}\|f_{23}\|\mathbf{cd})$
F-FX	$(\mathbf{a}\|f_{12}\|\mathbf{b}\|\{\nabla_2^2 f_{23}\}\|\mathbf{c})$
	$(\mathbf{ab}\|f_{12}\|\mathbf{c}\|\{\nabla_2^2 f_{23}\}\|\mathbf{d})$
	$(\mathbf{a}\|f_{12}\|\mathbf{bc}\|\{\nabla_2^2 f_{23}\}\|\mathbf{d})$
	$(\mathbf{a}\|f_{12}\|\mathbf{b}\|\{\nabla_2^2 f_{23}\}\|\mathbf{cd})$
F-FD	$(\mathbf{a}\|f_{12}\|\mathbf{b}\|f_{2_\lambda 3}\|\mathbf{c})$
	$(\mathbf{ab}\|f_{12}\|\mathbf{c}\|f_{2_\lambda 3}\|\mathbf{d})$
	$(\mathbf{a}\|f_{12}\|\mathbf{bc}\|f_{2_\lambda 3}\|\mathbf{d})$
	$(\mathbf{a}\|f_{12}\|\mathbf{b}\|f_{2_\lambda 3}\|\mathbf{cd})$
FT1-F	$(\mathbf{ab}\|[\hat{t}_1, f_{12}]\|\mathbf{c}\|f_{23}\|\mathbf{d})$
FT2-F	$(\mathbf{a}\|[\hat{t}_2, f_{12}]\|\mathbf{bc}\|f_{23}\|\mathbf{d})$

(Eq. 3.51), the three-electron, four-index integral $(A|r_{12}^{-1}|B|f_{23}|mk)$ is required, which may be obtained by contraction of the AO/DF integral $(A|r_{12}^{-1}|B|f_{23}|\text{cd})$ with the relevant MO coefficients (see section 2.3.2 for details of the notation used for molecular integrals). This AO/DF integral is itself obtained by contraction of primitive integrals with AO coefficients,

$$(A|r_{12}^{-1}|B|f_{23}|\text{cd}) = d_k^A d_l^B d_m^{\text{c}} d_n^{\text{d}} (\mathbf{a}_k|r_{12}^{-1}|\mathbf{b}_l|f_{23}|\mathbf{c}_m\mathbf{d}_n) \tag{3.99}$$

followed by spherical transformation (if spherical-harmonic GTOs are required). The other three- and four-index components of the robust density fitting expressions for $v_{ij,kl}$, $a_{kl,mn}^{(1)}$ and $a_{kl,mn}^{(2)}$ (Eqs. 3.51, 3.59 and 3.74) can be evaluated in the same way—the necessary primitive integrals are listed in Table 3.1.

As mentioned previously, for this initial implementation of the DF3 approach for MP2-F12 theory, the complicated and time-consuming process of coding and debugging new Obara-Saika-type recurrence relations (section 2.3.3) was avoided by use of an alternative method of evaluating three-electron primitive integrals. This alternative method allows the evaluation of primitive three-electron integrals of the general form

$$(\mathbf{a}|\hat{u}_{12}|\mathbf{b}|f_{23}|\mathbf{c}) = c_i(\mathbf{a}|\hat{u}_{12}|\mathbf{b}|g_{23}^{\mu_i}|\mathbf{c}), \tag{3.100}$$

starting from the two-electron integral $(\mathbf{a}|\hat{u}_{12}|\mathbf{r})$, where $\hat{u}_{12}$ is a general two-electron operator, and the correlation factor f_{23} is some linear combination of Gaussian geminals $g_{23}^{\mu_i} \equiv g(\mathbf{r}_2; \mu_i, \mathbf{0}, \mathbf{r}_3)$.

The method starts by evaluating the required $(\mathbf{a}|\hat{u}_{12}|\mathbf{r})$ integrals, typically using existing,

well-tested code in Molpro. These are then transformed to three-index, two-electron integrals using the GPT (Eq. 2.138, appendix B), i.e.

$$(\mathbf{a}|\hat{u}_{12}|\mathbf{bc}') = T_{\mathbf{r}}^{\mathbf{bc}'}(\mathbf{a}|\hat{u}_{12}|\mathbf{r}), \tag{3.101}$$

where the transformation coefficients, $T_{\mathbf{r}}^{\mathbf{bc}'}$, are as defined in appendix B, with the summation over $\mathbf{r}$ running from $(0,0,0)$ to $(b_x + c'_x, b_y + c'_y, b_z + c'_z)$,

$$|\mathbf{r}) \equiv g(\mathbf{r}; \zeta_b + \zeta_{c'}, \mathbf{r}, \mathbf{R}), \qquad |\mathbf{c}') \equiv g(\mathbf{r}; \zeta_{c'}, \mathbf{c}', \mathbf{C}), \tag{3.102}$$

with

$$\zeta_{c'} = \frac{\zeta_c \mu}{\zeta_c + \mu}, \qquad \mathbf{R} = \frac{\zeta_b \mathbf{B} + \zeta_{c'} \mathbf{C}}{\zeta_b + \zeta_{c'}}, \tag{3.103}$$

and μ being the exponent of the Gaussian geminal $g_{23}^{\mu_i}$ from Eq. 3.100.

Following the application of the GPT, a Gaussian transform[3] is used to transform the two-electron, three-index $(\mathbf{a}|\hat{u}_{12}|\mathbf{bc}')$ integrals into three-electron, three-index integrals:

$$(\mathbf{a}|\hat{u}_{12}|\mathbf{b}|g_{23}^{\mu}|\mathbf{c}) = C_{\mathbf{cc}'}^{\mu}(\mathbf{a}|\hat{u}_{12}|\mathbf{bc}'), \tag{3.104}$$

where the summation over $\mathbf{c}'$ runs from $(0,0,0)$ to (c_x, c_y, c_z) and the Gaussian transform coefficients are defined in appendix F. To obtain three-electron, four-index integrals, the GPT can be applied again, i.e.

$$(\mathbf{ab}|\hat{u}_{12}|\mathbf{c}|f_{23}|\mathbf{d}) = T_{\mathbf{p}}^{\mathbf{ab}}(\mathbf{p}|\hat{u}_{12}|\mathbf{c}|f_{23}|\mathbf{d}), \tag{3.105}$$

$$(\mathbf{a}|\hat{u}_{12}|\mathbf{bc}|f_{23}|\mathbf{d}) = T_{\mathbf{p}}^{\mathbf{bc}}(\mathbf{a}|\hat{u}_{12}|\mathbf{p}|f_{23}|\mathbf{d}), \tag{3.106}$$

$$(\mathbf{a}|\hat{u}_{12}|\mathbf{b}|f_{23}|\mathbf{cd}) = T_{\mathbf{p}}^{\mathbf{cd}}(\mathbf{a}|\hat{u}_{12}|\mathbf{b}|f_{23}|\mathbf{p}). \tag{3.107}$$

The technique just described can be applied to evaluate all the three-electron primitive integral types in Table 3.1. The J-F and F-F integrals may be evaluated starting with the two-electron "J" $(\mathbf{a}|r_{12}^{-1}|\mathbf{b})$ and "F" $(\mathbf{a}|f_{12}|\mathbf{b})$ integrals, both of which are implemented in Molpro (evaluation of the F-type integrals is described in Ref. 70). Evaluation of the F-FX integrals starts with the two-electron "FX" $(\mathbf{a}|\{\nabla_2^2 f_{12}\}|\mathbf{b})$ integrals, which are also available in Molpro.

For the F-FD, FT1-F and FT2-F integral types, some further manipulation is necessary in order that the method described above can be applied.

F-FD integrals

The implementation of the F-FD integrals using the method outlined in the preceding section requires the evaluation of the two-electron "FD" integrals:

$$\begin{aligned}(\mathbf{a}|f_{1_\lambda 2}|\mathbf{b}) &= -(\mathbf{a}|f_{12_\lambda}|\mathbf{b}) \\ &= \int \mathrm{d}\mathbf{r}_1 \mathrm{d}\mathbf{r}_2 \, g_a(\mathbf{r}_1)(\nabla_{1\lambda} f_{12}) g_b(\mathbf{r}_2),\end{aligned} \tag{3.108}$$

[3]Unpublished work by F.R. Manby and A. J. May (2004). Later reproduced in Ref. 150. Further details can be found in appendix F.

where $\lambda = x, y, z$, and where we have used the shorthand notation $g_a(\mathbf{r}_1) \equiv g(\mathbf{r}_1; \zeta_a, \mathbf{a}, \mathbf{A})$. Unlike the two-electron J, F and FX integrals, the FD integrals were not available in Molpro at the start of this project, and it was therefore necessary to write code for their evaluation.

When the correlation factor, f_{12}, represents a linear combination of Gaussian geminals, $g_{12}^{\mu_i}$ (as in Eq. 3.100) the $\nabla_{1\lambda} f_{12} \equiv f_{1_\lambda 2}$ operator may be expressed in terms of a linear combination of Gaussian geminals with a new set of coefficients:

$$\begin{aligned} \nabla_{1\lambda} f_{12} &= \frac{\partial}{\partial r_{1\lambda}} \sum_j c_j g_{12}^{\mu_j} \\ &= (r_{1\lambda} - r_{2\lambda}) \sum_j c'_j g_{12}^{\mu_j} \end{aligned} \tag{3.109}$$

where $c'_j = -2c_j\mu_j$. Defining a new correlation factor,

$$G_{12} = \sum_j c'_j g_{12}^{\mu_j}, \tag{3.110}$$

and using the identity

$$(r_{1\lambda} - r_{2\lambda}) = (r_{1\lambda} - A_\lambda) - (r_{2\lambda} - B_\lambda) + (A_\lambda - B_\lambda) \tag{3.111}$$

the $(\mathbf{a}|f_{1_\lambda 2}|\mathbf{b})$ integrals can be expressed in terms of integrals over the new correlation factor $(\mathbf{a}|G_{12}|\mathbf{b})$:

$$\begin{aligned} (\mathbf{a}|f_{1_\lambda 2}|\mathbf{b}) = (\mathbf{a} + \mathbf{1}_\lambda|G_{12}|\mathbf{b}) - (\mathbf{a}|G_{12}|\mathbf{b} + \mathbf{1}_\lambda) \\ + (A_\lambda - B_\lambda)(\mathbf{a}|G_{12}|\mathbf{b}). \end{aligned} \tag{3.112}$$

where $(\mathbf{a}|G_{12}|\mathbf{b})$ is an F-type integral with the new correlation factor (Eq. 3.110), i.e.

$$(\mathbf{a}|G_{12}|\mathbf{b}) = \sum_j c'_j (\mathbf{a}|g_{12}^{\mu_j}|\mathbf{b}). \tag{3.113}$$

Recognizing that the $(\mathbf{a}|g_{12}^{\mu}|\mathbf{b})$ integrals are translationally invariant (Eq. 2.153), the corresponding translational recurrence relation (TRR, e.g. Eq. 2.175, referred to as the "transfer equation" in Ref. 70),

$$\begin{aligned} (\mathbf{a}|g_{12}^{\mu}|\mathbf{b} + \mathbf{1}_\lambda) = \frac{a_\lambda}{2\zeta_b}(\mathbf{a} - \mathbf{1}_\lambda|g_{12}^{\mu}|\mathbf{b}) + \frac{b_\lambda}{2\zeta_b}(\mathbf{a}|g_{12}^{\mu}|\mathbf{b} - \mathbf{1}_\lambda) \\ - \frac{\zeta_a}{\zeta_b}(\mathbf{a} + \mathbf{1}_\lambda|g_{12}^{\mu}|\mathbf{b}), \end{aligned} \tag{3.114}$$

can be inserted into Eq. 3.112 to avoid the need to evaluate both $(\mathbf{a}|G_{12}|\mathbf{b})$ terms with angular

momentum incremented, i.e.

$$\begin{aligned}(\mathbf{a}|f_{1_\lambda 2}|\mathbf{b}) = &\left(1+\frac{\zeta_a}{\zeta_b}\right)(\mathbf{a}+\mathbf{1}_\lambda|G_{12}|\mathbf{b}) \\ &-\frac{b_\lambda}{2\zeta_b}(\mathbf{a}|G_{12}|\mathbf{b}-\mathbf{1}_\lambda)-\frac{a_\lambda}{2\zeta_b}(\mathbf{a}-\mathbf{1}_\lambda|G_{12}|\mathbf{b}) \\ &+(A_\lambda-B_\lambda)(\mathbf{a}|G_{12}|\mathbf{b}).\end{aligned} \tag{3.115}$$

The FD integrals can therefore be obtained by evaluating the required $(\mathbf{a}|G_{12}|\mathbf{b})$ integrals (using the existing code for F-type integrals) and assembling Eq. 3.115 using these—this was the approach taken for the implementation used in this work. It is then simply a case of following the general method for evaluating three-electron integrals described in Eqs. 3.100 to 3.107 to obtain the three- and four-index F-FD integrals from Table 3.1.

FT1-F integrals

The FT1-F integrals are defined as follows:

$$\begin{aligned}&(\mathbf{ab}|[\hat{t}_1,f_{12}]|\mathbf{c}|f_{23}|\mathbf{d}) \equiv ([\mathbf{ab}]||f_{12}|\mathbf{c}|f_{23}|\mathbf{d}) \\ &= \int \mathrm{d}\mathbf{r}_1\mathrm{d}\mathbf{r}_2\mathrm{d}\mathbf{r}_3 g_a(\mathbf{r}_1)[\hat{t}_1,f_{12}]g_b(\mathbf{r}_1)g_c(\mathbf{r}_2)f_{23}g_d(\mathbf{r}_3)\end{aligned} \tag{3.116}$$

with kinetic energy operator $\hat{t}_1 = -\frac{1}{2}\nabla_1^2$. These integrals arise in the evaluation of $a^{(1),\mathrm{robust}}_{kl,mn}$ (Eq. 3.59), since this requires the evaluation of $([ki]||f_{12}|B|f_{23}|C)$ integrals. The corresponding primitive integral is an FT1-F integral, i.e.

$$([\mathbf{ab}]||f_{12}|\mathbf{c}|f_{23}|\mathbf{d}) \equiv (\mathbf{ab}|[\hat{t}_1,f_{12}]|\mathbf{c}|f_{23}|\mathbf{d}). \tag{3.117}$$

The two-electron "FT" integrals,

$$(\mathbf{ab}|[\hat{t}_1,f_{12}]|\mathbf{c}) = \int \mathrm{d}\mathbf{r}_1\mathrm{d}\mathbf{r}_2 g_a(\mathbf{r}_1)[\hat{t}_1,f_{12}]g_b(\mathbf{r}_1)g_c(\mathbf{r}_2) \tag{3.118}$$

can be expressed as [4]

$$(\mathbf{ab}|[\hat{t}_1,f_{12}]|\mathbf{c}) = -\frac{1}{2}\frac{\zeta_a-\zeta_b}{\zeta_a+\zeta_b}(\mathbf{ab}|\{\nabla_1^2 f_{12}\}|\mathbf{c}) - \nabla_P\cdot\nabla_R(\mathbf{ab}|f_{12}|\mathbf{c}), \tag{3.119}$$

where

$$\mathbf{P} = \frac{\zeta_a\mathbf{A}+\zeta_b\mathbf{B}}{\zeta_a+\zeta_b}, \qquad \mathbf{R} = \mathbf{A}-\mathbf{B}. \tag{3.120}$$

The same relationship can be applied to the three-electron FT1-F integrals, since the derivation (appendix G) applies only to the parts of the integrand that depend on $\mathbf{r}_1$ which is the same in both the FT and FT1-F integral types:

$$g_a(\mathbf{r}_1)[\hat{t}_1,f_{12}]g_b(\mathbf{r}_1)\,f(\mathbf{r}_2,\cdots) \tag{3.121}$$

[4] Klopper and Röhse [184] published the derivation of a very similar relationship for the specific case where $f_{12} = r_{12}$ and May and Manby [70] later published the result where f_{12} is a linear combination of Gaussian geminals, as in Eq. 3.119. The derivation of May and Manby's result is presented in appendix G.

where for the FT integrals, $f(\mathbf{r}_2, \cdots) = g_c(\mathbf{r}_2)$ and for the FT1-F integrals, $f(\mathbf{r}_2, \cdots) = g_c(\mathbf{r}_2) f_{23} g_d(\mathbf{r}_3)$. Thus, in analogy to Eq. 3.119, the FT1-F integrals can be expressed as, in terms of F-FX, and F-F integrals:

$$\begin{aligned}(\mathbf{ab}|[\hat{t}_1, f_{12}]|\mathbf{c}|f_{23}|\mathbf{d}) = &-\frac{1}{2}\frac{\zeta_a - \zeta_b}{\zeta_a + \zeta_b}(\mathbf{ab}|\{\nabla_1^2 f_{12}\}|\mathbf{c}|f_{23}|\mathbf{d}) \\ &- \nabla_P \cdot \nabla_R (\mathbf{ab}|f_{12}|\mathbf{c}|f_{23}|\mathbf{d}).\end{aligned} \tag{3.122}$$

The action of $\nabla_P \cdot \nabla_R$ on the product of primitive Gaussians $|\mathbf{ab})$ can be represented by a linear combination of primitive Gaussians centred at $\mathbf{P}$ (as defined in Eq. 3.120),

$$\nabla_P \cdot \nabla_R |\mathbf{ab}) = Q_{\mathbf{p}}^{\mathbf{ab}} |\mathbf{p}), \tag{3.123}$$

where the summation over $\mathbf{p}$ contains all Cartesian components with total angular momentum from 0 to $|\mathbf{a}| + |\mathbf{b}| + 1$. In Ref. 131, the $Q_{\mathbf{p}}^{\mathbf{ab}}$ coefficients were used to evaluate the $(\mathbf{ab}|[\hat{t}_1, r_{12}]|\mathbf{c})$ integrals, but were not defined. The definition of the coefficients in terms of GPT coefficients (appendix B) was later published in Ref. 150 and is as follows

$$\begin{aligned}Q_{\mathbf{p}}^{\mathbf{ab}} = &-2\zeta_b(2\xi R_\lambda^2 + a_\lambda) T_{\mathbf{p}}^{\mathbf{ab}} - 4\zeta_a \zeta_b R_\lambda T_{\mathbf{p}}^{\mathbf{a}+\mathbf{1}_\lambda, \mathbf{b}} \\ &+ 2\zeta_a b_\lambda T_{\mathbf{p}}^{\mathbf{a}+\mathbf{1}_\lambda, \mathbf{b}-\mathbf{1}_\lambda} + 2\zeta_b \eta a_\lambda R_\lambda T_{\mathbf{p}}^{\mathbf{a}-\mathbf{1}_\lambda, \mathbf{b}} \\ &+ 4\xi b_\lambda R_\lambda T_{\mathbf{p}}^{\mathbf{a}, \mathbf{b}-\mathbf{1}_\lambda} - \eta a_\lambda b_\lambda T_{\mathbf{p}}^{\mathbf{a}-\mathbf{1}_\lambda, \mathbf{b}-\mathbf{1}_\lambda} \\ &+ \frac{\zeta_b}{\zeta} a_\lambda (a_\lambda - 1) T_{\mathbf{p}}^{\mathbf{a}-\mathbf{2}_\lambda, \mathbf{b}} - \frac{\zeta_a}{\zeta} b_\lambda (b_\lambda - 1) T_{\mathbf{p}}^{\mathbf{a}, \mathbf{b}-\mathbf{2}_\lambda}\end{aligned} \tag{3.124}$$

where $R_\lambda = A_\lambda - B_\lambda$, there is a summation over $\lambda = x, y, z$, and

$$\zeta = \zeta_a + \zeta_b, \qquad \xi = \frac{\zeta_a \zeta_b}{\zeta}, \qquad \eta = \frac{\zeta_a - \zeta_b}{\zeta}. \tag{3.125}$$

The full derivation of Eq. 3.124 is presented in appendix H.

In this work, the FT1-F integrals were therefore implemented using the following approach: evaluate the necessary three-electron F-FX and F-F integrals using the method described in Eqs. 3.100 to 3.107, and then combine these with the appropriate $Q_{\mathbf{p}}^{\mathbf{ab}}$ coefficients to evaluate the right side of Eq. 3.122.

FT2-F integrals

The FT2-F integrals are defined as follows:

$$(\mathbf{a}|[\hat{t}_2, f_{12}]|\mathbf{bc}|f_{23}|\mathbf{d}) = \int \mathrm{d}\mathbf{r}_1 \mathrm{d}\mathbf{r}_2 \mathrm{d}\mathbf{r}_3 g_a(\mathbf{r}_1) g_b(\mathbf{r}_2) [\hat{t}_2, f_{12}] g_c(\mathbf{r}_2) f_{23} g_d(\mathbf{r}_3) \tag{3.126}$$

with kinetic energy operator $\hat{t}_2 = -\frac{1}{2}\nabla_2^2$. These integrals arise in the evaluation of $a_{kl,mn}^{(2),\mathrm{robust}}$ (Eq. 3.74), since the $(ki|f_{12}|[ln]|f_{23}|im)_{\mathrm{robust}}$ term requires the evaluation of $(A|f_{12}|[ln]|f_{23}|C)$ integrals (Eq. 3.65). The corresponding primitive integral may be decomposed into FT2-F,

F-FD and F-FX integrals:

$$\begin{aligned}(\mathbf{a}|f_{12}|[\mathbf{bc}]|f_{23}|\mathbf{d}) = (\mathbf{a}|[\hat{t}_2, f_{12}]|\mathbf{bc}|f_{23}|\mathbf{d}) \\ - (\mathbf{a}|f_{12}|\mathbf{b}(\nabla_2\mathbf{c})|(\nabla_2 f_{23})|\mathbf{d}) \\ - \frac{1}{2}(\mathbf{a}|f_{12}|\mathbf{bc}|\{\nabla_2^2 f_{23}\}|\mathbf{d}).\end{aligned} \tag{3.127}$$

The method used to decompose the FT1-F integrals cannot be applied to the FT2-F integrals, since the part of the integrand that is common to the FT and FT1-F integrals (highlighted in Eq. 3.121) is not present in the FT2-F integrals. For the FT2-F integrals, the integrand has the form

$$g_a(\mathbf{r}_1)\, g_b(\mathbf{r}_2)[\hat{t}_2, f_{12}]g_c(\mathbf{r}_2)f_{23}\, g_d(\mathbf{r}_3) \tag{3.128}$$

where we have highlighted all the terms which depend on $\mathbf{r}_2$—it is clear that the highlighted region differs in structure from that in Eq. 3.121 by inclusion of the f_{23} operator.

The derivation of Eqs. 3.119 and 3.122 requires that the following relationship holds:

$$\nabla_P(\mathbf{ab}|(\nabla_1 f_{12})|\mathbf{c}\cdots) = (\mathbf{ab}|\{\nabla_1^2 f_{12}\}|\mathbf{c}\cdots), \tag{3.129}$$

One reason why the same method used for the FT1-F integrals cannot be applied to the FT2-F integrals is that the corresponding relation for the FT2-F integrals does not hold, i.e.

$$\nabla_S(\mathbf{a}|(\nabla_2 f_{12})|\mathbf{bc}|f_{12}|\mathbf{d}) \neq (\mathbf{a}|\{\nabla_2^2 f_{12}\}|\mathbf{bc}|f_{12}|\mathbf{d}), \tag{3.130}$$

where

$$\mathbf{S} = \frac{\zeta_b\mathbf{B} + \zeta_c\mathbf{C}}{\zeta_b + \zeta_c}. \tag{3.131}$$

For the FT1-F integrals, the second term on the right of

$$\begin{aligned}&\nabla_P(\mathbf{ab}|(\nabla_1 f_{12})|\mathbf{c}|f_{23}|\mathbf{d}) \\ &= \int \mathrm{d}\mathbf{r}_1\mathrm{d}\mathbf{r}_2\mathrm{d}\mathbf{r}_3 g_a(\mathbf{r}_1)g_b(\mathbf{r}_1)\{\nabla_1^2 f_{12}\}g_c(\mathbf{r}_2)f_{23}g_d(\mathbf{r}_3) \\ &\quad - \int \mathrm{d}\mathbf{r}_1\mathrm{d}\mathbf{r}_2\mathrm{d}\mathbf{r}_3 \nabla_1 \cdot (g_a(\mathbf{r}_1)g_b(\mathbf{r}_1)\nabla_1 f_{12})g_c(\mathbf{r}_2)f_{23}g_d(\mathbf{r}_3)\end{aligned} \tag{3.132}$$

vanishes (via integration-by-parts, see Eq. G.26 in appendix G) such that Eq. 3.129 holds.

Following the derivation in appendix G for the FT2-F integrals, we arrive at

$$\begin{aligned}&\nabla_S(\mathbf{a}|(\nabla_2 f_{12})|\mathbf{bc}|f_{23}|\mathbf{d}) \\ &= \int \mathrm{d}\mathbf{r}_1\mathrm{d}\mathbf{r}_2\mathrm{d}\mathbf{r}_3 g_a(\mathbf{r}_1)g_b(\mathbf{r}_2)\{\nabla_2^2 f_{12}\}g_c(\mathbf{r}_2)f_{23}g_d(\mathbf{r}_3) \\ &\quad - \int \mathrm{d}\mathbf{r}_1\mathrm{d}\mathbf{r}_2\mathrm{d}\mathbf{r}_3 g_a(\mathbf{r}_1)\{\nabla_2 \cdot (\nabla_2 f_{12}g_b(\mathbf{r}_2)g_c(\mathbf{r}_2))\}f_{23}g_d(\mathbf{r}_3)\end{aligned} \tag{3.133}$$

and the second term on the right does not necessarily vanish under integration-by-parts (cf. Eq. G.26), since the integration over $\mathbf{r}_2$ now also includes the f_{23} correlation factor.

In this work, we therefore used an alternative method to evaluate the FT2-F integrals, using

the following expression obtained by expanding the commutator:

$$\begin{aligned}(\mathbf{a}|[\hat{t}_2, f_{12}]|\mathbf{bc}|f_{23}|\mathbf{d}) = \frac{1}{2}(\mathbf{a}|\{\nabla_2^2 f_{12}\}|\mathbf{bc}|f_{23}|\mathbf{d}) \\ + (\mathbf{a}|(\nabla_2 f_{12})|(\nabla_2 \mathbf{b})\mathbf{c}|f_{23}|\mathbf{d}).\end{aligned} \tag{3.134}$$

Using Eq. 3.134, the FT2-F integral is expressed in terms of an F-FX integral and three F-FD integrals featuring the derivative of a primitive Cartesian Gaussian, one for each component of the dot product

$$(\nabla_2 f_{12}) \cdot (\nabla_2 g_b(\mathbf{r}_2))$$

i.e.

$$(\mathbf{a}|(\nabla_2 f_{12})|(\nabla_2 \mathbf{b})\mathbf{c}|f_{23}|\mathbf{d}) = \sum_{\lambda=x,y,z} (\mathbf{a}|f_{12_\lambda}|\mathbf{b}_\lambda \mathbf{c}|f_{23}|\mathbf{d}). \tag{3.135}$$

The F-FD integrals in Eq. 3.135 can be evaluated using standard F-FD integrals with angular momentum incremented/decremented by applying the differential operator to $|\mathbf{b})$ (Eq. 2.152):

$$\begin{aligned}(\mathbf{a}|f_{12_\lambda}|\mathbf{b}_\lambda \mathbf{c}|f_{23}|\mathbf{d}) = -2\zeta_b(\mathbf{a}|f_{12_\lambda}|(\mathbf{b}+\mathbf{1}_\lambda)\mathbf{c}|f_{23}|\mathbf{d}) \\ + b_\lambda(\mathbf{a}|f_{12_\lambda}|(\mathbf{b}-\mathbf{1}_\lambda)\mathbf{c}|f_{23}|\mathbf{d}).\end{aligned} \tag{3.136}$$

3.3.3 Contractions

To obtain the robust (and non-robust) density-fitted three-electron integrals required in DF3-MP2-F12/3*A(FIX) theory (Eqs. 3.51, 3.59 and 3.74 to 3.77), the various corresponding primitive integral types (Table 3.1) must be contracted with the correct AO, MO and DF coefficients.

The AO contraction phase is straightforward since each summation over AO coefficients for an integral index is independent of the summation for the other indexes (see, for example, Eq. 3.99). The spherical transformation of each index is similarly independent of the other indexes. Although DF basis sets are often uncontracted, contraction with "AO" coefficients for the DF integral indexes was still necessary in this implementation, since the radial normalization factor ($N_{\zeta l}$ in Eq. 2.133) was folded into the coefficients.

In theory, the ordering of the various AO contraction and spherical transformation steps could be optimized to minimize the overall computational expense of the sequence. However, for simplicity, no optimization of this type was done in this "proof-of-concept" implementation.

To obtain the density-fitted integrals over MOs, contraction of the AO/DF integrals with MO and DF coefficients is necessary. This process is more complicated than the preceding contraction with AO coefficients, since the DF and MO coefficients share indexes and may be contracted in different combinations. For example, consider the three-index component of $v_{ij,kl}^{\text{robust}}$ (Eq. 3.51):

$$D_A^{im} D_B^{jl} D_C^{mk} (A|r_{12}^{-1}|B|f_{23}|C) \equiv (\widetilde{im}|r_{12}^{-1}|\widetilde{jl}|f_{23}|\widetilde{mk}) \,. \tag{3.137}$$

One possible first step is to contract two DF coefficients together, i.e.

$$X_{AC}^{ik} = D_A^{im} D_C^{mk}, \tag{3.138}$$

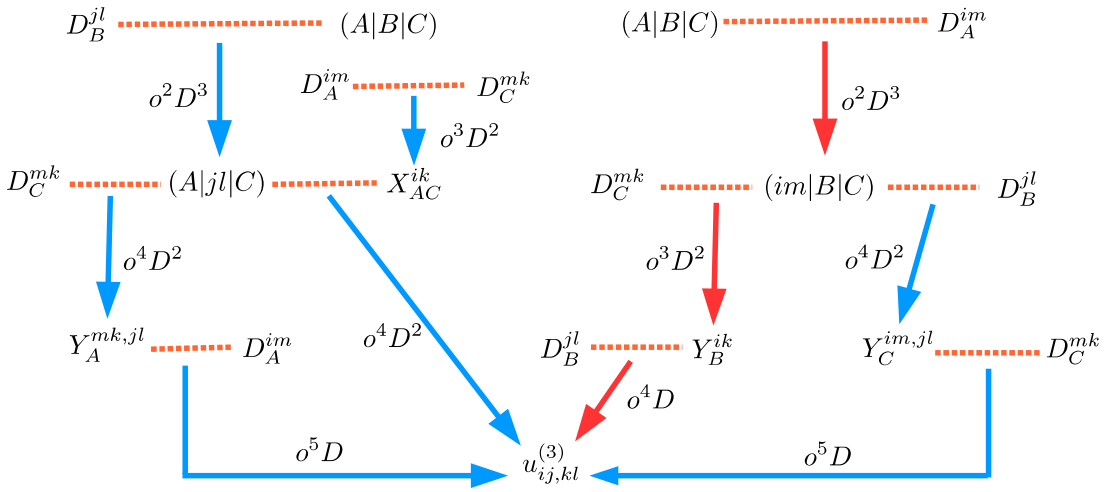

Figure 3.1: Diagram of some possible contraction routes to obtain the three-index, three-electron component of $u_{ij,kl}$ (Eq. 3.140, operators omitted for clarity). The orange dashed lines represent contractions between objects, with the arrows descending from them leading to the result of the contraction. The route used in this work, which has a maximum $\mathcal{O}(N^5)$ formal scaling (N is a measure of system size), is represented by the red arrows. Next to the arrows, the scaling of each step is provided, with o and D being the number of occupied MOs and DF basis functions, respectively.

where occupied MO index m is summed over. This step has a formal scaling of $\mathcal{O}(o^3D^2)$, where o is the number of occupied MOs, and D is the number of functions in the DF basis set. An alternative start point is to contract a DF coefficient with the integral

$$X_{BC}^{im} = D_A^{im}(A|\hat{u}_{12}|B|f_{23}|C) \tag{3.139}$$

which has a formal scaling of $\mathcal{O}(o^2D^3)$. In each case, the set of possible subsequent contractions is different—to determine an efficient sequence, all these operations must be considered together.

For the sequence of contractions with MO and DF coefficients, the polynomial order of the formal scaling expression of the contraction sequence can change depending on the sequence of operations—this is illustrated in Fig. 3.1, where some routes to the final fully-contracted object have a higher formal scaling with respect to system size than others.

The robust density fitting expression for all the three-electron integral types required in DF3-MP2-F12/3*A(FIX) has the general form:

$$\begin{aligned} u_{ij,kl} &= (im|\hat{u}_{12}|jl|f_{23}|mk)_{\text{robust}} \\ &= u_{ij,kl}^{(4,1)} + u_{ij,kl}^{(4,2)} + u_{ij,kl}^{(4,3)} - 2u_{ij,kl}^{(3)} \end{aligned} \tag{3.140}$$

where $\hat{u}_{12}$ is a general two-electron operator and there is an implicit summation over occupied

MO index m. The components of Eq. 3.140 are

$$\begin{aligned} u^{(3)}_{ij,kl} &= D^{im}_A D^{jl}_B D^{mk}_C (A|\hat{u}_{12}|B|f_{23}|C), \\ u^{(4,1)}_{ij,kl} &= D^{jl}_B D^{mk}_C c^i_{\mathrm{a}} c^m_{\mathrm{b}} (\mathrm{ab}|\hat{u}_{12}|B|f_{23}|C), \\ u^{(4,2)}_{ij,kl} &= D^{im}_A D^{mk}_C c^j_{\mathrm{c}} c^l_{\mathrm{d}} (A|\hat{u}_{12}|\mathrm{cd}|f_{23}|C), \\ u^{(4,3)}_{ij,kl} &= D^{im}_A D^{jl}_B c^m_{\mathrm{e}} c^k_{\mathrm{f}} (A|\hat{u}_{12}|B|f_{23}|\mathrm{ef}), \end{aligned} \tag{3.141}$$

with MO coefficients c^i_{a} and where the density fitting coefficients may be for fitting a standard orbital product (Eq. 2.207), commutator orbital product (Eq. 3.57), or gradient orbital product (Eq. 3.71). To implement a general scheme for robust density fitting $u_{ij,kl}$, it was necessary to devise three contraction schemes: one for each of $u^{(3)}_{ij,kl}$ and $u^{(4,2)}_{ij,kl}$ and another for both $u^{(4,1)}_{ij,kl}$ and $u^{(4,3)}_{ij,kl}$.

It is clear from the example in Fig. 3.1 that there are many possible ways in which the fully-contracted components of Eq. 3.140 could be obtained—implementing and testing each possibility would have been very time-consuming. We therefore devised the contraction schemes for each of the components in Eq. 3.141 based on theoretical arguments. Our primary concern in devising efficient contraction sequences was to minimize the computational scaling with respect to system size N—sequences with overall lower polynomial order scaling expressions were preferred. Selecting only sequences with the lowest possible overall computational scaling with respect to N greatly reduced the number of possibilities. For example, in Fig. 3.1, many of the possible routes have steps which scale as N^6, but there is at least one route with a maximum scaling of N^5, meaning that the N^6 routes can be disregarded. Where multiple sequences with equivalent order polynomial scaling expressions in N existed, our selections were informed by other factors, including:

- The relative sizes of individual terms in the scaling expression for the highest scaling step(s).
- The scaling of other steps in the sequence.
- The size of partially-contracted intermediates.

As mentioned in sections 2.2.6 and 2.3.5, this kind of theoretical cost analysis uses an idealized model of computation, and does not account for many of the factors affecting real-world computation. The sequences selected using the approach described above may therefore not be the most efficient possibilities when implemented and run on a real computer, particularly where multiple possibilities with the same overall computational scaling exist. However, by minimizing the computational cost with respect to N, we can at least be sure that the worst-case computational scaling of the sequences with system size is as low as possible. A rigorous analysis would likely require some degree of real-world testing—the prohibitive amount of work associated with implementing many possible sequences in software for comparison is a strong argument in favour of code generation (see chapter 4).

The contraction schemes selected for use in this work based on the approach just outlined are as follows. For the $u^{(3)}_{ij,kl}$ component, the following contraction sequence, with $\mathcal{O}(N^5)$ formal

scaling, was adopted (see also Fig. 3.1):

$$\begin{aligned} X^{im}_{BC} &= D^{im}_{A}(A|\hat{u}_{12}|B|f_{23}|C) && \mathcal{O}(o^2D^3) \\ Y^{ik}_{B} &= X^{im}_{BC}D^{mk}_{C} && \mathcal{O}(o^3D^2) \\ u^{(3)}_{ij,kl} &= D^{jl}_{B}Y^{ik}_{B} && \mathcal{O}(o^4D) \end{aligned} \tag{3.142}$$

where the computational scaling of each step in terms of o, the number of occupied MOs, and D, the number of DF basis functions, is shown next to each step.

For the $u^{(4,1)}_{ij,kl}$ component, the contraction sequence,

$$\begin{aligned} (i\mathrm{b}|\hat{u}_{12}|B|f_{23}|C) &= c^{i}_{\mathrm{a}}(\mathrm{ab}|\hat{u}_{12}|B|f_{23}|C) && \mathcal{O}(a^2oD^2) \\ (im|\hat{u}_{12}|B|f_{23}|C) &= c^{m}_{\mathrm{b}}(i\mathrm{b}|\hat{u}_{12}|B|f_{23}|C) && \mathcal{O}(ao^2D^2) \\ X^{ik}_{B} &= D^{mk}_{C}(im|\hat{u}_{12}|B|f_{23}|C) && \mathcal{O}(o^3D^2) \\ u^{(4,1)}_{ij,kl} &= X^{ik}_{B}D^{jl}_{B} && \mathcal{O}(o^4D) \end{aligned} \tag{3.143}$$

with a formal scaling of $\mathcal{O}(N^5)$ was used, with a representing the number of functions in the AO basis in the scaling expressions. The contraction scheme adopted for $u^{(4,3)}_{ij,kl}$ has the same structure and scaling as Eq. 3.143, but with indexes and operators permuted, i.e.

$$\begin{aligned} (A|\hat{u}_{12}|B|f_{23}|m\mathrm{f}) &= c^{m}_{\mathrm{e}}(A|\hat{u}_{12}|B|f_{23}|\mathrm{ef}) && \mathcal{O}(a^2oD^2) \\ (A|\hat{u}_{12}|B|f_{23}|mk) &= c^{k}_{\mathrm{f}}(A|\hat{u}_{12}|B|f_{23}|m\mathrm{f}) && \mathcal{O}(ao^2D^2) \\ X^{ik}_{B} &= D^{im}_{A}(A|\hat{u}_{12}|B|f_{23}|mk) && \mathcal{O}(o^3D^2) \\ u^{(4,3)}_{ij,kl} &= X^{ik}_{B}D^{jl}_{B} && \mathcal{O}(o^4D). \end{aligned} \tag{3.144}$$

Finally, for the $u^{(4,2)}_{ij,kl}$ component, the following contraction scheme, which scales as $\mathcal{O}(N^6)$, was implemented:

$$\begin{aligned} (A|\hat{u}_{12}|j\mathrm{d}|f_{23}|C) &= c^{j}_{\mathrm{c}}(A|\hat{u}_{12}|\mathrm{cd}|f_{23}|C) && \mathcal{O}(a^2oD^2) \\ (A|\hat{u}_{12}|jl|f_{23}|C) &= c^{l}_{\mathrm{d}}(A|\hat{u}_{12}|j\mathrm{d}|f_{23}|C) && \mathcal{O}(ao^2D^2) \\ X^{ik}_{AC} &= D^{im}_{A}D^{mk}_{C} && \mathcal{O}(o^3D^2) \\ u^{(4,2)}_{ij,kl} &= X^{ik}_{AC}(A|\hat{u}_{12}|jl|f_{23}|C) && \mathcal{O}(o^4D^2). \end{aligned} \tag{3.145}$$

The presence of an $\mathcal{O}(N^6)$ scaling step in the contraction scheme for $u^{(4,2)}_{ij,kl}$ is unavoidable as a consequence of the structure of the objects involved. Regardless of the route taken, a contraction between a four-index object and another object with at least two additional unique indexes must occur, e.g. the final step in Eq. 3.145

$$u^{(4,2)}_{ij,kl} = X^{ik}_{AC}(A|\hat{u}_{12}|jl|f_{23}|C) \tag{3.146}$$

and a step from an alternative scheme

$$(im|\hat{u}_{12}|\mathrm{cd}|f_{23}|C) = D^{im}_{A}(A|\hat{u}_{12}|\mathrm{cd}|f_{23}|C) \tag{3.147}$$

both of which have a formal scaling of $\mathcal{O}(N^6)$.

The process of forming three-electron integral $u_{ij,kl}$ (with robust density fitting expression Eq. 3.140) using the contraction schemes described in Eqs. 3.142 to 3.145 has an overall computational scaling of $\mathcal{O}(N^6)$.[5] In the non-robust case, $u_{ij,kl} \equiv u^{(3)}_{ij,kl}$, and only the contraction scheme described in Eq. 3.142 is required. As a consequence, the non-robust density fitting $u_{ij,kl}$ has the advantage of a lower formal scaling of $\mathcal{O}(N^5)$.

Approximation of $u_{ij,kl}$ by resolution of the identity scales as $\mathcal{O}(N^6)$ and takes the following form:

$$\begin{aligned} u^{\mathrm{RI}}_{ij,kl} &= \langle ijm|\hat{u}_{12}\hat{X}_2 f_{23}|mlk\rangle \\ &= \langle ij|\hat{u}_{12}|mx\rangle\langle mx|f_{12}|kl\rangle \end{aligned} \tag{3.148}$$

with approximate identity operator $\hat{X}_2 = |x\rangle\langle x|$ and orthonormal RI basis $\{|x\rangle\}$. Resolution of the identity and robust density fitting therefore have the same overall $\mathcal{O}(N^6)$ computational scaling, while non-robust density fitting has a lower formal scaling than either approach: $\mathcal{O}(N^5)$.

The relative computational costs of the $\mathcal{O}(N^6)$ steps in the robust density fitting procedure for $u_{ij,kl}$ (Eq. 3.145) and the resolution of the identity in Eq. 3.148 can be analysed by comparing the structure of the scaling expressions. The aforementioned caveats regarding theoretical cost analysis (sections 2.2.6 and 2.3.5) notwithstanding, this is a reasonable comparison to make, since both steps are matrix multiplications which are likely to have comparable per-operation costs.

The scaling expression for the approximate RI in Eq. 3.148 is o^5x, where x represents the size of the RI basis set, while the expression for the $\mathcal{O}(N^6)$ step in Eq. 3.145 is o^4D^2. The ratio between these two scaling expressions is

$$R = \frac{o^4D^2}{o^5x} = \frac{D^2}{xo}. \tag{3.149}$$

Since typically, $x \approx D >> o$ (i.e. the size of the auxiliary basis sets is comparable, but much larger than the number of occupied MOs) we would expect that $R > 1$ and that the $\mathcal{O}(N^6)$ step in robust density fitting $u_{ij,kl}$ would be more costly than the $\mathcal{O}(N^6)$ step in approximating $u_{ij,kl}$ by resolution of the identity. However, given the rapid convergence of the error in the correlation energy with DF basis set size and angular momentum described in section 3.4, it is plausible that this may not be a problem in practice.

Unfortunately, at present we are limited to the above theoretical analysis for comparing the computational efficiency of the RI and DF3 approaches for three-electron integral approximation. A rigorous real-world performance comparison is not possible with the "proof-of-concept" implementation of DF3-MP2-F12/3*A(FIX) described in this section. Such a comparison would require a more computationally efficient implementation, in particular for the evaluation of the primitive three-electron integral types described in section 3.3.2—the code generation work described in chapter 4 offers a possible means by which this could be achieved.

[5] In Ref. 176, it is suggested that the scaling of robust density fitting three-electron integrals of this type was $\mathcal{O}(N^5)$, but this is incorrect since there are no possible contraction schemes for $u^{(4,2)}_{ij,kl}$ with a computational scaling lower than $\mathcal{O}(N^6)$.

Although all the results described in this chapter use the diagonal approximation (see section 3.2.1), we have presented contraction schemes for the non-diagonal case. The preceding analysis of the computational scaling of these schemes refers to the general case. The diagonal approximation was not assumed for the derivation and implementation of the DF3 contraction schemes in order that these may be applied to non-diagonal MP2-F12 approximation schemes in the future. It is, however, interesting to consider how the diagonal approximation affects the computational scaling of the contractions involved in robust density fitting $u_{ij,kl}$ (Eq. 3.140) and the corresponding RI (Eq. 3.148). Using the same contraction schemes but for the diagonal three-electron integrals $u_{ij,ij}$ and $u_{ij,ji}$, the overall computational scaling for the robust DF3 contraction schemes (Eqs. 3.142 to 3.145) is reduced to $\mathcal{O}(N^5)$.

The schemes presented here may not be optimal when the diagonal approximation is assumed, though, an overall $\mathcal{O}(N^5)$ scaling is the minimum possible for any robust density fitting contraction scheme where the diagonal approximation is applied. In robust density fitting $u_{ij,ij}$ and $u_{ij,ji}$, there will always be steps involving the contraction of a four-index object with another object with at least one additional index, e.g.

$$(A|\hat{u}_{12}|B|f_{23}|\mathrm{mf}) = c_{\mathrm{e}}^{m}(A|\hat{u}_{12}|B|f_{23}|\mathrm{ef}), \tag{3.150}$$

which will scale as at least $\mathcal{O}(N^5)$.

At first glance, it might appear that, under the diagonal approximation, approximating $u_{ij,ij}$ and $u_{ij,ji}$ using RI, scales as $\mathcal{O}(N^4)$, since when kl is replaced with ij or ji in Eq. 3.148, the scaling expression for that contraction step is o^3x. The overall scaling of approximating these integrals using RI is, however, $\mathcal{O}(N^5)$, since to form the four-index MO integrals in Eq. 3.148 requires that four-index AO integrals are contracted with MO coefficients,

$$\langle ij|\hat{u}_{12}|mx\rangle = c_{\mathrm{a}}^{i}\langle \mathrm{a}j|\hat{u}_{12}|mx\rangle \tag{3.151}$$

which has the scaling expression ao^3x, i.e. $\mathcal{O}(N^5)$.

3.4 Results and Discussion

To determine the utility of density fitting for the three-electron integrals in MP2-F12/3*A(FIX) theory (i.e. "the DF3 approach"), the robust and non-robust DF3 integral approximation approaches (section 3.2.3) were compared to the ABS RI and CABS RI approaches (section 3.2.4) in calculations on small molecules (H_2O and HF, Figs. 3.2 and 3.3) and a transition metal atom/cation (Zn, Zn^{2+}, Figs. 3.4 and 3.5, and Table 3.2). The behaviour of these various integral approximation methods with respect to the size and maximum angular momentum available in the auxiliary basis set was examined, with clear differences apparent. Crucially, the DF3 approach exhibited much improved convergence with respect to both the size and angular momentum available in the auxiliary basis set.

In the following, all MP2-F12/3*A(FIX) calculations involving RIs use density-fitting to approximate the two-electron integrals arising in MP2-F12 theory, i.e. the DF-MP2-F12 approach is used [70, 131] (MP2-F12 theory without density fitting is not available in Molpro). The DF3

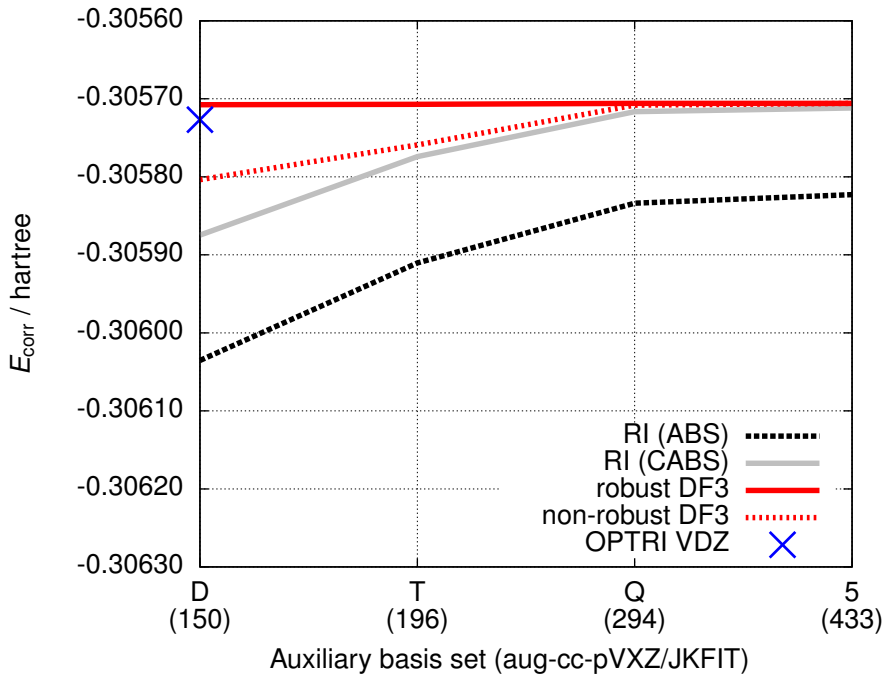

Figure 3.2: Plot of DF-MP2-F12/3*A(FIX) valence correlation energies of the H_2O molecule (for geometry data, see appendix I) with increasing auxiliary basis set size, using the ABS/-CABS RI and non-robust/robust DF3 approaches to approximate three-electron integrals. The aug-cc-pVXZ/JKFIT auxiliary basis sets used for all approximation schemes were constructed by augmenting the cc-pVXZ/JKFIT sets from Ref. 142 with a single even-tempered function at each angular momentum level. The aug-cc-pVDZ/JKFIT sets were created by removing the highest angular momentum functions from the aug-cc-pVTZ/JKFIT set. The number of functions in each auxiliary set is provided in parentheses under the basis set name. The aug-cc-pVDZ AO basis set was used in all calculations [51, 72]. The blue cross represents the valence correlation energy calculated using CABS RI with the OPTRI basis set corresponding to the AO basis set used [186].

Note: The aug-cc-pVXZ/JKFIT auxiliary basis sets do not appear to have been published in the literature, but are available publicly via the Molpro basis set library at `http://www.molpro.net`.

approaches were implemented by modifying the existing DF-MP2-F12 code in Molpro, and thus density fitting was applied to both two- and three-electron integrals in DF3-MP2-F12/3*A(FIX) calculations. In addition, density fitting was applied to the calculation of the standard MP2 component of the correlation energy, i.e. DF-MP2 [57, 144, 145] was used. In both cases, the error from density fitting two-electron integrals was minimized by using a very large DF basis set (cc-pV5Z/MP2FIT for H_2O, HF and def2-AQZVPP/MP2FIT for Zn, Zn^{2+} [185], see note associated with Fig. 3.4).

3.4.1 H_2O and HF

The convergence of the MP2-F12/3*A(FIX) valence correlation energy of H_2O and HF with respect to auxiliary basis set size for the RI and DF3 integral approximations is plotted in Figs. 3.2 and 3.3. The following general trends are observed for both small molecules:

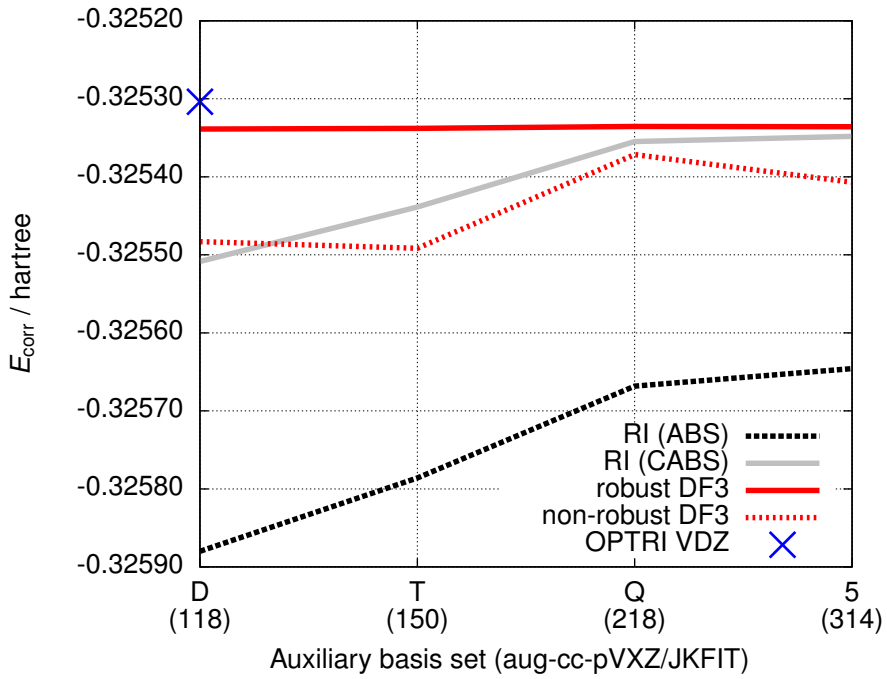

Figure 3.3: Plot of DF-MP2-F12/3*A(FIX) valence correlation energies of the HF molecule (for geometry data, see appendix I) with increasing auxiliary basis set size, using the ABS/CABS RI and non-robust/robust DF3 approaches to approximate three-electron integrals. The AO and auxiliary basis sets used are as described for Fig. 3.2.

- The robust DF3 energies converge very rapidly, with the correlation energy calculated using a double-ζ (DZ) auxiliary basis set being within $\sim 10^{-6}$ hartree of the quintuple-ζ (5Z) energy.
- The other approaches converge more slowly, with the difference between the DZ and 5Z energies being $\sim 10^{-4}$ hartree for non-robust DF3, ABS RI and CABS RI.
- As expected, the valence correlation energy converges more rapidly with auxiliary basis set size for CABS RI than for ABS RI.

Both plots feature a single point CABS RI calculation using an OPTRI auxiliary basis set [186,187]. Since the OPTRI basis sets are optimized to represent the orthogonal complement to a specific AO basis, only the OPTRI auxiliary set specifically optimized for the AO basis set (aug-cc-pVDZ [51, 72]) was used [186]. The calculated energies for CABS RI with the aug-cc-pVDZ/OPTRI basis set improve upon the CABS RI energy obtained using the aug-cc-pVDZ/JKFIT auxiliary basis, being converged to within $\sim 10^{-5}$ hartree of the 5Z result (for either CABS RI or robust DF3 approximation methods).

When developing the OPTRI basis sets, Yousaf and Peterson observed that the inclusion of g-type basis functions significantly reduced the RI error for the aug-cc-pVDZ/OPTRI basis set. Though the aug-cc-pVDZ/JKFIT basis set has more functions compared to the aug-cc-pVDZ/OPTRI set (for H_2O and HF the OPTRI sets have 113 and 91 functions, respectively) it also lacks g-functions—this may help explain the larger error (with respect to the 5Z energy) seen in the CABS RI energies for the DZ JKFIT set compared to the OPTRI set.

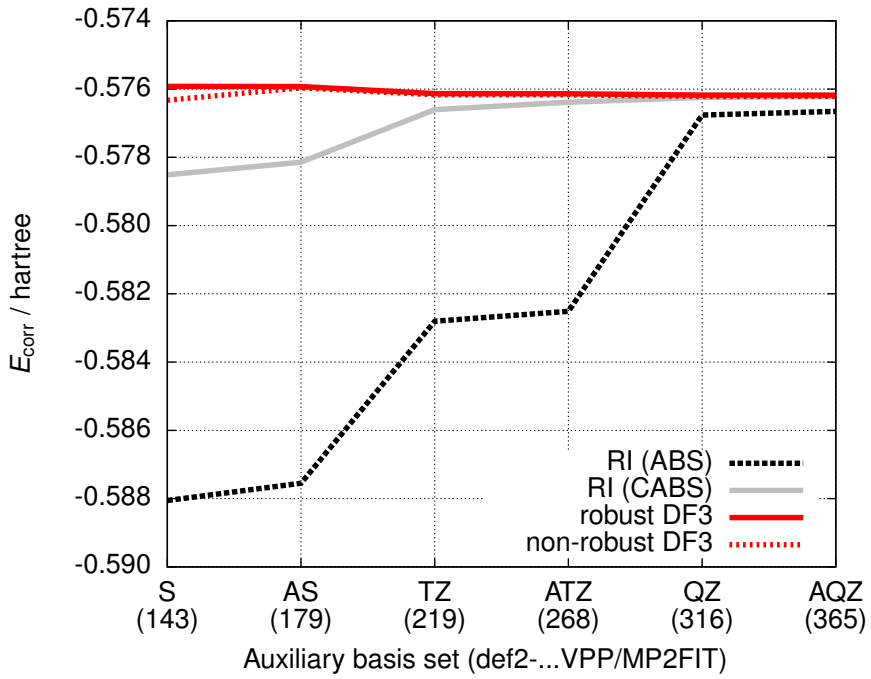

Figure 3.4: DF-MP2-F12/3*A(FIX) valence correlation energies of the Zn atom ($4s^2 3d^{10}$) plotted for increasing auxiliary basis set size, using the ABS/CABS RI and non-robust/robust DF3 approaches to approximate three-electron integrals. MP2FIT auxiliary basis sets optimized for the def2 AO basis set family were used [185, 188], with the ASVP, ATZVPP and AQZVPP sets formed by addition of a set of even-tempered diffuse functions to the corresponding SVP, TZVPP and QZVPP MP2FIT sets. The number of functions in each auxiliary set is provided in parentheses under the basis set name. All calculations used the def2-TZVPP AO basis set [189].

Note: The def2-ASVP/MP2FIT, def2-ATZVPP/MP2FIT and def2-AQZVPP/MP2FIT auxiliary basis sets do not appear to have been published in the literature, but are available publicly via the Molpro basis set library at `http://www.molpro.net`.

3.4.2 Zn and Zn^{2+}

Figs. 3.4 and 3.5 demonstrate the behaviour of the MP2-F12/3*A(FIX) valence correlation energy for the Zn atom for increasing size and angular momentum available in the auxiliary basis set used for the RI and DF3 integral approximations. The behaviour of the double ionization energy of Zn with respect to angular momentum available in the auxiliary basis is examined in Table 3.2.

The transition metal atoms present a challenge to the standard RI approach in terms of saturating the RI basis set. As described in section 3.2.4, to approximate the three-electron integral $v_{ij,kl}$ (in the single-centre case, Eq. 3.34) formally requires $3\ell_{\mathrm{occ}}$ units of angular momentum in the RI basis set. Since the transition metal atoms have occupied d-orbitals ($\ell = 2$), the RI basis to fit three-electron integrals like $v_{ij,kl}$ over the occupied orbitals of these atoms should contain i-functions ($\ell = 6$). Using the DF3 approach (section 3.2.3) to approximate $v_{ij,kl}$ (Eq. 3.34), the DF basis formally requires functions with up to only $2\ell_{\mathrm{occ}}$. For a transition metal atom with occupied d-orbitals, the DF basis set should therefore contain up to g-functions ($\ell = 4$).

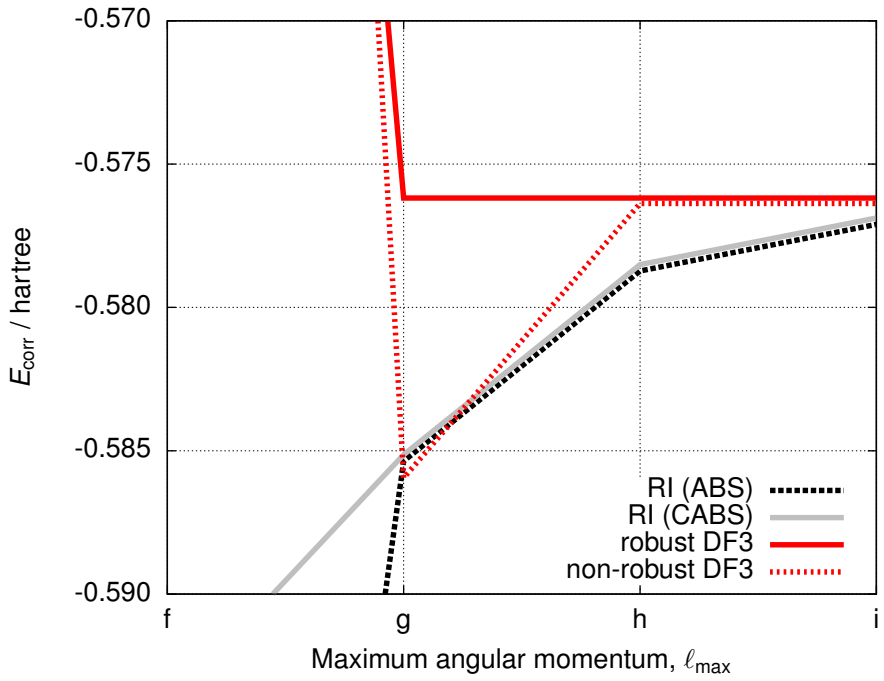

Figure 3.5: Plot of DF-MP2-F12/3*A(FIX) valence correlation energies of the Zn atom ($4s^2 3d^{10}$) with increasing angular momentum in the auxiliary basis set, calculated with three-electron integrals approximated using the ABS/CABS RI and non-robust/robust DF3 approaches. The auxiliary basis set used in all the three-electron integral approximation schemes was constructed from functions from the def2-QZVPP/JKFIT auxiliary basis set (the "Contracted, standard" basis set in Ref. 190) with total angular momentum up to and including ℓ_{max}. The def2-TZVPP AO basis set was used in all calculations [189].

The different angular momentum requirements of the DF and RI basis sets are borne out in the results obtained for the Zn atom and Zn^{2+} ion. Fig. 3.4 exhibits the same trends observed for H_2O and HF in Figs. 3.2 and 3.3, though over a larger range of energies. The def2-(A)SVP/MP2FIT auxiliary basis sets have only up to h-functions ($\ell = 5$), while the (A)TZ and (A)QZ sets have up to i-functions ($\ell = 6$). For both RI variants the improvement in the correlation energies is significant, when moving from (A)S to (A)TZ sets, indicating the importance of i-functions for these methods. In particular, for the CABS RI approach, sub-millihartree differences (with respect to the AQZ result) are only attained when the TZ set, or larger is used.

The ABS RI approach is significantly slower to converge, only attaining valence correlation energies within a millihartree of the converged value (for robust DF3 or CABS RI) when the (A)QZ auxiliary sets are used. In contrast, both the non-robust and robust DF3 approaches yield energies that are within a millihartree of the AQZ result using only a def2-SVP/MP2FIT DF basis. As has been mentioned, the def2-(A)SVP/MP2FIT are not saturated to $3\ell_{occ}$, so these results indicate that the DF3 approach does indeed place a lower angular momentum requirement on the auxiliary basis than the RI approach.

Fig. 3.5 clearly demonstrates the lower angular momentum requirements of the DF3 approach, compared to RI. In this case, the valence correlation energy of Zn is plotted against the

Table 3.2: Double ionization energies of the Zn atom (Zn, $4s^2 3d^{10} \rightarrow Zn^{2+}$, $3d^{10}$) in millihartree, calculated using DF-MP2-F12/3*A(FIX) theory with three-electron integrals approximated using the ABS/CABS RI or non-robust/robust DF3 approaches. The auxiliary basis set for all three-electron integral approximation schemes was constructed from functions from the def2-QZVPP/JKFIT auxiliary basis set (the "Contracted, standard" basis set in Ref. 190) with total angular momentum up to and including $\ell_{\max}$. The def2-TZVPP AO basis set was used in all calculations [189].

$\ell_{\max}$	ABS RI	CABS RI	Non-robust DF3	Robust DF3
3	974.483160	976.135720	978.263309	977.759893
4	974.778962	974.774913	974.718356	974.969957
5	974.917008	974.912959	974.966144	974.970005
6	974.963739	974.959690	974.966144	974.970005

maximum angular momentum available in the auxiliary basis set. It is immediately clear from the plot that, as soon as the auxiliary basis contains g-functions ($\ell = 4$), the robust DF3 correlation energy is converged to a value very close to the value for the full def2-QZVPP/JKFIT auxiliary basis set. In fact, once g-functions are introduced, the robust DF3 energy comes within $\sim 10^{-6}$ hartree of the energy calculated for the full auxiliary basis set (with h- and i- functions). The convergence of the ABS and CABS RI energies with respect to angular momentum in the auxiliary basis set is significantly slower, with the difference between the energies for $\ell_{\max} = 5$ (h-functions) and $\ell_{\max} = 6$ (i-functions) being several millihartree.

When the g-functions are removed, the robust DF3 energy diverges away from the converged value very rapidly—where $\ell_{\max} = 3$ (f-functions), the calculated valence correlation energy is 92 millihartree away from the converged value. The same rapid divergence for auxiliary basis sets truncated below $\ell_{\max} = 4$ is observed for all the other integral approximation methods, with the exception of CABS RI, where the divergence is somewhat slower, though still significant. This is likely because in the CABS RI approach, the identity operator is approximated by the union of the MO basis and the approximated orthogonal complement (see section 3.2.4), so higher angular momentum functions from the AO basis (def2-TZVPP in this case, which has up to g-functions) contribute to the RI, even where they are not available in the auxiliary basis set.

It appears that, for the robust DF3 approach, the DF basis should contain functions with up to $2\ell_{\text{occ}}$ to accurately approximate the $v_{ij,kl}$, $a^{(1)}_{kl,mn}$ and $a^{(2)}_{kl,mn}$ (Eqs. 3.34, 3.40 and 3.41) integrals required in MP2-F12/3*A(FIX) theory. This is interesting observation considering that robust density fitting the $a^{(1)}_{kl,mn}$ and $a^{(2)}_{kl,mn}$ integrals involves fitting the commutator and gradient product densities, $|[ij]\rangle$ and $|ij_\lambda\rangle$ (Eqs. 3.56 and 3.69). Given that the derivative of a Gaussian function includes terms with incremented angular momentum (Eq. 2.152), one might predict that functions with more than $2\ell_{\text{occ}}$ units of angular momentum would be required to accurately approximate $a^{(1)}_{kl,mn}$ and $a^{(2)}_{kl,mn}$ using density fitting. In fact, the non-robust DF3 energies presented in Fig. 3.5 do not converge to within less than a millihartree of the energy for the full auxiliary basis until h-functions ($\ell = 5$) are added—this may be a consequence of fitting these unusual product densities. This suggests that perhaps the reduced error from robust density fitting can compensate for the effect of using an auxiliary basis set that is not

formally saturated in terms of angular momentum.

To gauge the effect of the angular momentum available in the auxiliary basis set on energy differences, the double ionization energy of Zn (i.e. Zn, $4s^2 3d^{10} \rightarrow Zn^{2+}$, $3d^{10}$) for increasing auxiliary basis set ℓ_{max} was calculated (double ionization was used, rather than single ionization to ensure that all calculations were closed-shell). Table 3.2 presents Zn double ionization energies, calculated using MP2-F12/3*A(FIX) theory using RI and DF3 integral approximation approaches, and with increasing angular momentum available in the auxiliary basis set.

As with the valence correlation energy calculations (Fig. 3.5), the double ionization energies calculated using the robust DF3 approach converge rapidly—when g-functions are added, the robust DF3 energy is converged to within $\sim 10^{-8}$ hartree of the full auxiliary basis set ($\ell_{max} = 6$) value. For the non-robust DF3 and RI approaches, the difference between the energy calculated for the auxiliary basis set truncated at $\ell_{max} = 4$ and the full auxiliary basis is several orders of magnitude larger.

It is interesting to make comparisons between the Villani and Klopper's results for very large basis set MP2-R12 calculations on Zn and Zn^{2+} [191] and the data presented in Table 3.2. Using the Zn and Zn^{2+} Hartree-Fock and correlation energies[6] from Tables 6 and 7 in Ref. 191, the near-complete-basis-set MP2 double ionization energy for Zn is 981.13 millihartree. The converged energies in Table 3.2 are calculated using a def2-TZVPP AO basis, and differ from this very large basis number by several millihartree. Our def2-TZVPP energies sit closer to the double ionization energies for TZVPP (970.77 millihartree) and QZVPP (977.07 millihartree) AO basis sets[7] (using the same very large RI basis) calculated using the data from Ref. 191. The $\sim 10^{-8}$ hartree error from truncating the auxiliary basis to $\ell_{max} = 4$ for robust DF3 is therefore $\sim 10^6$ times smaller than the error from AO basis set incompleteness seen in Villani and Klopper's results.

3.5 Conclusions

This chapter describes the derivation and computational implementation of DF3-MP2-F12/3*A(FIX) theory, in which three-electron integrals are approximated using density fitting, rather than resolutions of the identity (RIs). The DF3-MP2-F12/3*A(FIX) method avoids the problematic angular momentum constraints associated with the use of RIs to approximate many-electron integrals by employing density fitting, which imposes lower angular momentum requirements on the fitting basis set. The lower angular momentum requirement associated with density fitting the three-electron integrals is clearly demonstrated by the rapid convergence of DF3-MP2-F12/3*A(FIX) correlation and double ionization energies with the size (Figs. 3.2, 3.3 and 3.4) and maximum angular momentum (Fig. 3.5 and Table 3.2) in the auxiliary basis set.

It appears that relatively small auxiliary basis sets can be used in DF3-MP2-F12/3*A(FIX) theory, since the error from robust fitting three-electron integrals using small aug-cc-pVDZ/JKFIT and def2-SVP/MP2FIT auxiliary sets is significantly smaller than the

[6] MP2-R12 theory with approximation diag-A and a very large basis set with $\ell_{max} = 7$ used as AO and RI basis.

[7] The TZVPP and QZVPP used in Ref. 191 are from Refs. 145, 192.

typical error from AO basis incompleteness in an MP2-F12 calculation. In addition, provided the auxiliary basis set contains functions with up to $2\ell_{\mathrm{occ}}$, the DF3-MP2-F12/3*A(FIX) approach seems to be relatively insensitive to the auxiliary basis set used. It therefore seems likely that the DF basis used to fit two electron integrals in DF-MP2-F12 theory [70, 81, 131] could also be used to fit the three-electron integrals in DF3-MP2-F12/3*A(FIX), avoiding the need to introduce an additional auxiliary basis set.

The "proof-of-concept" implementation of DF3-MP2-F12/3*A(FIX) described in this chapter uses inefficient techniques to evaluate the density-fitted three-electron integrals, most notably in the evaluation of the primitive integrals (section 3.3.2, Eqs. 3.100 to 3.107). Results relating to the real-world computational performance of the implementation have not been reported since these would be dominated by the inefficient techniques used and would not be representative of the computational properties of the DF3 method.

The theoretical analysis of computational scaling presented in section 3.3.3 indicates that the robust DF3 approach has the same formal scaling with respect to system size, N, as a standard RI ($\mathcal{O}(N^6)$ in the general case and $\mathcal{O}(N^5)$ for the diagonal approximation). Consideration of the scaling expressions for the highest scaling steps in the RI and robust DF3 approaches suggests that, where the RI and DF basis sets are similar in size, the robust DF3 approach would be more costly, since the most costly step in RI scales linearly with respect to the auxiliary basis set size while the most cost step in robust DF3 scales quadratically (for the general, non-diagonal case).

The high accuracy observed in energies calculated using the robust DF3 approach using relatively small auxiliary basis sets may reduce or eliminate this disadvantage. Of course, the real-world performance of a given implementation is affected by many factors other than the formal scaling of a method (see sections 2.2.6 and 2.3.5), so the preceding remarks on the theoretical cost of the DF3 and RI approaches should not be considered to be definitive.

The non-robust DF3 approach, in which $v_{ij,kl}$, $a^{(1)}_{kl,mn}$ and $a^{(2)}_{kl,mn}$ are approximated using the non-robust fitting expressions Eqs. 3.75 to 3.77, scales as $\mathcal{O}(N^5)$ (for the general, non-diagonal case), but exhibits slower convergence with respect to the size and angular momentum in the auxiliary basis set compared to robust DF3. Nevertheless, the non-robust DF3 approach may be worth investigating further, since in all tests presented in section 3.4, it exhibits similar or better convergence than the CABS RI approach. It may be that the non-robust DF3 approach offers acceptable accuracy for approximating three-electron integrals in MP2-F12 theory with a simpler, lower scaling approach than robust DF3.

An obvious direction for future work on the DF3-MP2-F12/3*A(FIX) method would be to develop a more computationally efficient implementation of the theory. The automatic code generation work described in chapter 4 was a response to the difficulty of writing efficient code to evaluate the three-electron integrals necessary in the DF3 approach—the evaluation of three-electron primitive integrals (section 3.3.2) could be significantly accelerated by using automatically generated code, as demonstrated in section 4.4.3.

A more efficient implementation of DF3-MP2-F12/3*A(FIX) theory could take better advantage of the diagonal approximation, for example by using contraction schemes specific to the diagonal approximation. Under the diagonal approximation, some cancellations of terms

in the robust density fitting expressions (section 3.2.3) are also possible. For example, the $(ki|f_{12}|[ln]|f_{23}|im)$ term in Eq. 3.63 vanishes if $kl = mn$, since

$$(ki|f_{12}|[ll]|f_{23}|ik) = 0 \tag{3.152}$$

because $|[ll]) = 0$. Such cancellations were not investigated in this work, but could improve efficiency in a future implementation of DF3-MP2-F12/3*A(FIX) theory.

Another interesting avenue for expanding upon the work presented here would be a rigorous investigation of the auxiliary basis set requirements of the robust and non-robust DF3 approaches. For example, it would be interesting to know whether a single MP2FIT or JKFIT auxiliary basis could be used for density fitting the ERIs in DF-MP2, as well as the two- and three-electron integrals in MP2-F12/3*A(FIX) theory. It would also be useful to investigate whether the auxiliary basis set insensitivity exhibited by the DF3 approach (particularly the robust DF3 approach) for H_2O, HF and Zn (section 3.4) generalizes over a larger set of molecules. In addition, it may be worth examining the optimization of auxiliary basis sets specifically for use in the DF3 approximation. DF3-specific sets could be optimized in the same way as the OPTRI basis sets, minimizing a functional based on the diagonal elements of the $\mathbf{V}$ (Eq. 3.11) and $\mathbf{B}$ (Eq. 3.9) matrices [186, 187]. Given the observed insensitivity of robust DF3 to the auxiliary basis set used, it seems unlikely that robust DF3 would significantly benefit from optimized auxiliary basis sets. It is possible, however, that non-robust DF3 energies may be further improved by using DF3-specific auxiliary sets.

The diagonal 3*A MP2-F12 approximation scheme was selected for this work because it features no integrals over more than three electron coordinates. To apply the density fitting approach to integrals over more than three electrons would have required considerable additional work, involving new robust fitting expressions and contraction schemes. There are other explicitly correlated methods that require no more than three-electron integrals, to which variants of the DF3 approach could be applied. For example, the transcorrelated method [62], which features three-electron integrals of the form $\langle pqr|(\nabla_1 f_{12}) \cdot (\nabla_2 f_{13})|stu\rangle$ [56, 156].

To apply the DF3 approach to other MP2-F12 variants, four-electron integrals involving the exchange operator could be avoided by use of intermediate orbitals [173] and neglect of terms where intermediate orbitals cannot be applied, similar to the approach used in the "hybrid" schemes [71, 193]. The coupled cluster F12 methods present a further challenge for density fitting of many-electron integrals, since these introduce further integral intermediates on top of those required for MP2-F12 theory [171]. A good starting point for applying the DF3 approach to explicitly correlated coupled cluster might be the CCSD-F12a and CCSD-F12b methods, which require only a single additional integral intermediate, $V_{pq}^{kl} = \langle pq|f_{12}\hat{Q}_{12}r_{12}^{-1}|kl\rangle$ (where p, q may be virtual MOs) [80, 169, 171].

In summary, the "proof-of-concept" DF3-MP2-F12/3*A(FIX) method presented here (particularly the robust variant) exhibits some attractive features, namely:

- Rapid convergence with respect to size of the auxiliary basis set.
- Lower maximum angular momentum required in the auxiliary basis set compared to resolution of the identity.

To rigorously determine the numerical and computational properties of the DF3 approach to integral approximation, it would be necessary to create a more efficient implementation and test the method on a wider range of chemical systems. Nevertheless, the DF3 approach presents a promising alternative to the use of resolutions of the identity for the approximation of many-electron integrals in explicitly correlated methods, particularly where high angular momentum occupied MOs are present.

CHAPTER 4

CODE GENERATION FOR MOLECULAR-INTEGRAL EVALUATION

Note on object-oriented programming

This chapter assumes a basic familiarity with object-oriented programming. In particular, the description of the software developed for this work (section 4.3) requires some knowledge of classes, objects and inheritance. There are many widely available computer science and programming books, as well as online resources (including the Python language documentation [194]) which introduce these concepts.

4.1 Introduction

The evaluation of molecular integrals is a vital component of electronic structure methods. In computational implementations of these methods, the evaluation of molecular integrals is often a costly step, if not the most expensive part of the procedure. Efficient molecular integral evaluation is therefore a key concern in the creation of software for performing electronic structure calculations.

In section 2.3, the basic theory associated with the evaluation of integrals over Gaussian-type orbitals (GTOs) was established. GTOs have become the *de facto* standard AO basis functions used in electronic structure calculations because their functional form facilitates efficient molecular integral evaluation (Eqs. 2.137 and 2.138), while satisfying key theoretical requirements (see the list on p. 32). Many theoretical schemes for efficient evaluation of molecular integrals over GTOs have been developed, as outlined in section 2.3.3 (see also Refs. 10 (ch.9), 98 and 99).

In this chapter, we are concerned solely with the Obara-Saika scheme [1] for molecular integral evaluation, where recurrence relations are used to increment angular momentum in integrals over primitive Cartesian Gaussians. The essential notation, terminology and theoretical details of this scheme, and basic information relating to its implementation in software are covered in sections 2.3.3 to 2.3.5.

Obara and Saika published their scheme in 1986 [1]. Since then, there have been numerous theoretical developments which enhance the efficiency of integral evaluation using the scheme, including:

- The use of horizontal recurrence relations (HRRs) to make some angular momentum incrementing work independent of the degree of contraction [101] (sections 2.3.3 and 2.3.5).
- The "PRISM" algorithm for ERI evaluation [98,108–110], which allows the point at which contraction occurs in integral evaluation to be dynamically selected.
- The "J-matrix engine algorithm", which involves early summation over density matrix elements in the formation of the Coulomb contribution to the Fock matrix (Eq. 2.47) [111].
- The development of new HRRs which transfer multiple units of angular momentum between integral indexes [102].
- The development of TRR-based recurrence relations for ERIs which allow angular momentum to be transferred between indexes on different electrons [195].
- Optimization of the VRRs and HRR used to evaluate ERIs by analysis of the associated tree-search problem [196, 197].
- The recognition that certain terms in the VRR for the evaluation of three-index Coulomb integrals vanish upon spherical transformation, leading to a simplified VRR [198] (see section 4.2.3 for details).

In general, these advances have focused upon the implementation of four-index ERIs and three-index Coulomb integrals, since these are typically the most numerous, and most costly, integrals involved in conventional electronic structure calculations.

The efficient evaluation of three-index Coulomb integrals has become more important with the advent of density-fitted methods, such as DF-HF [141–143] and DF-MP2 [57, 144, 145], where the four-index ERIs are approximated in terms of two- and three-index Coulomb integrals (see section 2.4). Density fitting, and the closely related Cholesky decomposition, Eq. 2.196, are approximation methods, which complement the evaluation of integrals using the Obara-Saika scheme, rather than modifying the scheme itself. Other complementary techniques for improving integral evaluation efficiency include direct integral evaluation and integral screening, as described in section 2.3.6.

In addition to the theoretical advances just outlined, improvements in computer hardware have yielded significant increases in the speed of molecular integral evaluation. Since the 1980s, the speed of processors and quantity of memory available in high-performance computers has increased by orders of magnitude, and the software running on this hardware has largely been able to benefit from these vast improvements for "free", i.e. without significant modification. Nevertheless, to take full advantage of the capabilities of computer hardware, software must be written in a manner which accounts for the strengths and limitations of the hardware. For code which evaluates molecular integrals, general methods for creating efficient numerical software apply, e.g. ensuring efficient use of CPU registers and caches, efficient memory access, and utilization of instruction-level and data-level parallelism. See Refs. 121 and 122 for details regarding the concepts and techniques associated with writing numerical software.

In the past decade, parallel computing hardware has proliferated to the point where it is now ubiquitous—it is now unusual for even the least powerful of personal computers to have less than

two CPU cores. To fully utilize modern computing hardware, software must now be written in a way which takes advantage of parallelism. For the kind of multi-node supercomputer used in research, this often means utilizing multiple levels of parallelism—at the level of multiple nodes (e.g. using MPI [199]) and within a node (e.g. using OpenMP [200]). In addition, the use of graphics processing units (GPUs) for numerical computation has become widespread in the past few years. These are highly parallel processors, with many processing units executing the same instructions on different data (i.e. single-instruction, multiple data, SIMD). If software is written in a way which takes advantage of this architecture, significant improvements in performance can be achieved. However, writing software for GPUs presents unique challenges, particularly in terms of managing data transfer between the CPU and GPU, and typically requires the use of GPU-specific programming frameworks, such as NVIDIA's CUDA [201] or OpenCL [202]. The use of GPUs for molecular integral evaluation is a area of active research, which has yielded some impressive performance improvements over conventional CPU-based integral evaluation [112, 115, 127, 128, 203, 204].

In chapter 3, a new formulation of MP2-F12 theory, was described, in which density fitting was used to approximate many-electron integrals instead of resolutions of the identity. The most significant challenge in the development and testing of this new theory was the implementation of new three-electron integral types in software. The theoretical and computational methods outlined earlier for improving the efficiency of molecular integral code were not helpful in this respect, since the issue was not in the performance of the implementation, but in the human cost of implementing the integrals in software. As described in section 3.3.2, this challenge was partially mitigated by the use of less efficient methods that made use of existing code for the evaluation of two-electron integral types. Despite this, writing and debugging source code for the new integral types was still very time-consuming. If a reasonably efficient implementation of the three-electron integral types had been already available, a large amount of work could have been saved, and perhaps a more detailed investigation of the new theory could have been performed.

The time-consuming nature of numerical software development extends beyond the realm of molecular integral evaluation, and is a challenge for computational scientists writing software in many fields. The problem is compounded by the continuing development of computer technology. As has already been mentioned, to fully utilize new hardware architectures often requires the adoption of new algorithms and programming frameworks, necessitating some degree of modification or rewriting of existing code. In addition, numerical procedures are often common to multiple software packages—if these are implemented independently, then the effort of writing the software is duplicated.

One way to address the time cost of numerical software development is the creation of standard libraries and application programming interfaces (APIs), which allow code to be reused in different contexts. These provide a standard set of routines that can be called from external software packages. Examples of this approach, as relevant to computational chemistry, include:

- BLAS (Basic Linear Algebra Subprograms), which specifies standardized interfaces for routines for performing linear algebra operations, such as matrix multiplication [205] (many implementations of the standard exist, such as ATLAS [206] and the Intel Math

Kernel Library [207]).

- Libxc, a library of routines for the evaluation of exchange-correlation functionals in DFT [208].
- OpenMM, a software library (and also a fully-featured application) for performing molecular mechanics simulations [209].

A conceptually similar approach is "component-based software engineering" (CBSE), where applications are constructed from modular software "components" which interact through clearly-defined interfaces. The Common Component Architecture (CCA) [210] specifies a component programming model for high-performance scientific computing, which has been used to develop components for quantum chemistry, including for molecular integral evaluation [211–213].

Libraries, APIs and CBSE are very useful in situations where software is required to perform well-defined tasks, and where those tasks can easily be isolated from other tasks. In this case, code to perform these tasks can be developed once and then reused across multiple software packages, and for different purposes, avoiding duplication of programming effort. Molecular integral evaluation might appear to be a good candidate for this approach, as most electronic structure calculations require a well-defined set of integral types. However, as evidenced by the work presented in chapter 3, this is not always the case. A researcher seeking to implement a new theory requiring the evaluation of non-standard integral types would still need to write integral evaluation code, even if a library of standard routines was available.

Another approach for reducing the burden of scientific software development is code generation, in which a computer program is used to automatically create software to a particular specification. While libraries, APIs and similar approaches to code reuse are limited to predefined behaviours, a code generator can create software which behaves in novel ways. This is an attractive prospect for the implementation of new theory, which may require entirely new behaviours.

Arguably, the first "code generators" were compilers, developed in the 1950s to reduce the burden of programming for early computers by allowing programs to be written in high-level languages, such as Fortran, rather than machine code (see Ref. 214, and references therein for more details). In this work, however, we will use the phrase "code generation" to describe the use of a computer program to automatically create source code in a high-level language. In this sense, a code generator is intended to automate the work of a modern-day human programmer, who would otherwise need to write high-level source code by hand.

The idea of automating the implementation of theory in quantum chemistry has a long history. Some early examples from the 1950s and 1960s include the works of Boys, Cook, Reeves and Shavitt [215], Gray, Pritchard and Sumner [216], and of Wactlar and Bartnett [217], relating to the automation of the "tedious" algebra and mathematical manipulations involved in the implementation of quantum chemical theories. The automation in these early works focused upon using computers to perform mathematical analysis which would otherwise require significant human time and effort.

More recently, the idea of reducing the cost of implementing theory has driven the use of code generation in the field of quantum chemistry. Often, these code generators have combined

the automation of mathematical derivation and software implementation, as with the Tensor Contraction Engine (TCE) [218–220], which automatically derives working equations for many-body methods and generates source code for their evaluation. Other works featuring some degree of code generation in relation to the implementation many-body methods include Refs. 221–224 (see also Ref. 225 for a detailed overview of automatic derivation and implementation of many-body theories). The implementation of exchange-correlation functionals in DFT has also been a target for code generation approaches [226, 227]. In both cases, code generation is intended to facilitate the development of new theory, by automation of systematic elements of the process of method development, such as the derivation of working equations and the creation of source code.

In this work, we describe the development and application of a code generator which automatically implements code for the evaluation of molecular integrals. The creation of this code generator was motivated by the significant time and effort required to implement code for the evaluation of new integral types, as demonstrated by the difficulties encountered in chapter 3. The code generating software, "Intception", uses the Obara-Saika scheme for molecular integral evaluation to flexibly generate source code for a wide range of integral types. As described in section 4.2, the systematic, modular, nature of the Obara-Saika scheme makes it a good candidate for code generation.

Intception is one of a number of projects which apply code generation techniques to molecular integral evaluation. In recent years, several works have been published which apply code generation for the implementation of GPU-specific molecular integral evaluation source code [112, 113, 115, 204, 228]. These could be considered "integral-specific" code generators, using code generation as a means to create optimized GPU code specifically for the evaluation of ERIs. Intception differs from these code generation projects, in that it is designed to be capable of generating code for a broad range of integral types and thus is a "general" code generator.

There are other general code generators in development for molecular integral code. "Libcint" [229] is a recently released general code generator and symbolic algebra system which uses the Dupuis-Rys-King integral evaluation scheme [94–96] to automatically implement code for a wide range of integral types. "Libint" [230] is a well-established open-source library of efficient molecular integral evaluation routines, created by means of a code generator and developed by the Valeev group. Like Intception, Libint's code generator uses Obara-Saika-type recurrence relations [1] (section 2.3.3) and both Libint and Intception employ a domain-specific language (DSL) to express these recurrence relations. The use of a DSL is a key component of the code generation approach used in Intception, where it serves as both an input language and an internal representation of domain-specific elements (see section 4.3.2).

A DSL is a language designed to express a particular problem domain, usually in a highly abstracted fashion (see section 4.3.2 for details). DSLs are particularly useful for concisely expressing complicated abstract concepts, and have found use in areas other than code generation. For example, in the integrated tensor framework [152] (and the local integrated tensor framework (LITF) [231] based on this), a DSL is used to express tensor contraction algorithms, which are read and executed by a program at run-time. This is an alternative route to automatic method implementation which does not involve code generation, but instead uses an

interpreter to implement algorithms expressed in abstract form. Another example is the super instruction assembly language (SIAL), used in the electronic structure package ACES III [232]. SIAL is used to abstractly express parallel operations on large blocks of data, and is executed by a run-time environment (the super instruction processor, SIP) which deals with the details of efficiently implementing these operations on specific hardware [233].

In the remainder of this chapter, the underlying theory (section 4.2), implementation (section 4.3) and application (section 4.4) of Intception is described. The following key topics are covered:

- A theoretical framework for integral evaluation (section 4.2.2) which can be used to implement a general code generator (section 4.3.5).
- The use of a DSL to allow flexible, yet simple, specification of integral types (sections 4.3.2 and 4.3.3).
- The structure and behaviour of generated source code (section 4.3.4).
- The practical application of Inception for source code generation (section 4.3.6).

In addition, results are presented (section 4.4) that demonstrate the numerical and computational performance of the generated code and indicate where the development of Intception should be focused in the future.

4.2 Theory

The aim of this work has been to construct a general automatic code generator for molecular integrals over Gaussian-type orbitals (GTOs), where the mathematical definition of an integral is directly translated into source code for the evaluation of that integral. A general code generator should be capable of automatically generating code for a broad range of integral types—this is distinct from an integral-specific code generator, which may only generate code for a single type of integral. For an integral-specific code generator, the definition of an integral type (i.e. the information required to evaluate that integral) and algorithm for evaluation of that integral may be implicit in the source code of the generator. In contrast, for a general code generator, the integral definition necessarily forms part of the input, and the code generator must use some theoretical framework by which an abstract integral definition is translated into an algorithm for evaluating that integral.

The framework for molecular integral evaluation using Obara-Saika-type recurrence relations [1] described in section 2.3 is well-suited to general code generation, since it can be decomposed into basic computational steps, that can be generally applied to evaluate a great diversity of integral types. For example, the basic VRR-only and VRR+HRR integral evaluation algorithms described in section 2.3.5 (presented in Figs. 2.7 and 2.8 for the two-index overlap integrals) can be used to evaluate a variety of molecular integral types by customizing the individual components of the algorithm (e.g. VRR, HRR, contraction). In principle, source code for the evaluation of a general integral type could be automatically generated by assembling a program from these algorithm components, provided that a valid evaluation algorithm can be

expressed using the theoretical framework. This is the core premise of the work presented in this chapter: *code for evaluating specific molecular integral types can be generated by assembling and customizing the components of a general theoretical framework for molecular integral evaluation.*

In the code generating software described in the following sections we use a framework based upon Obara-Saika-type recurrence relations, though the premise should apply to other general frameworks which use other techniques, such as the McMurchie-Davidson [97] and Dupuis-Rys-King [94–96] algorithms. Indeed, Sun's recently published Libcint package is a general molecular integral code generator which uses the Dupuis-Rys-King framework [229].

When hand-coding an integral evaluation routine, it is typically not necessary to keep in mind rigorous formal definitions of the various components of the theoretical framework used—it is often intuitively obvious how to customize and assemble the components. For this assembly and customization to be done programatically, however, clear formal definitions are vital. Consider the example of identifying different types of recurrence relation, a necessary step in implementing integral evaluation code featuring recurrence relations. A human with relevant expertise would be able to immediately determine whether a given recurrence relation is a VRR or HRR by visual inspection, even if they found it difficult to articulate formal definitions of these. In contrast, a computer program could only distinguish between the two recurrence relation types if it was able check for certain conditions unambiguously associated with VRRs and HRRs. In the software described in this chapter, recurrence relations are defined in a DSL (see section 4.3.2), so the code generator must distinguish between different recurrence relation types by checking formal definitions against the representations of recurrence relations in the DSL.

While an expert human programmer can use inference and experience to write correct integral evaluation code for a given integral type, a general code generator must be provided with explicit definitions and rules for translation of the abstract integral definition into source code. Rigorous definitions for the theoretical components of the integral evaluation framework used by Intception are presented in section 4.2.2.

4.2.1 Terminology

To provide unambiguous definitions of the components of the integral evaluation framework used by Intception, it is useful to first establish some precise terminology for expressing the definitions:

Integral index

An index associated with an integral type, typically a primitive Cartesian Gaussian, contracted GTO or molecular orbital, labelled by a letter using the notation outlined in section 2.3.2. These indexes may have associated with them various properties, e.g. a centre, angular momentum or exponent, though not all integral indexes we consider have all these properties. An integral type is defined by the number and type of integral indexes associated with it, along with any operators (e.g. r_{12}^{-1}, f_{12}).

Auxiliary index

An additional index introduced to assist integral evaluation, typically an integer index, e.g.

m in Eq. 2.169. Auxiliary indexes are not used to define an integral type outside of the context of integral evaluation, i.e. they are not relevant after integral evaluation has taken place.

Increment/decrement

An increase/decrease in a value which characterizes an integral or auxiliary index. For integral indexes, this is an increase/decrease in angular momentum, typically in a general Cartesian direction i. For auxiliary indexes, this is generally an integer increase or decrease in the value of that index.

Assigned integral

An instance of an integral type which is assigned in a recurrence relation, and which depends on other instances of that integral type, e.g. the integrals on the left side of Eqs. 2.157 and 2.173.

Intermediate integral

An instance of an integral type which is required to evaluate the assigned integral using a recurrence relation, e.g. the integrals on the right side of Eqs. 2.157 and 2.173. These typically feature incremented or decremented indexes (integral or auxiliary).

4.2.2 Integral evaluation framework

Component definitions

The theoretical framework for integral evaluation used in this work consists of several distinct components, which can be assembled and customized to create algorithms for the evaluation of many types of molecular integral. These components were described informally in section 2.3.5, with examples of how they can be assembled into integral evaluation algorithms presented in Figs. 2.7 and 2.8. The formal definitions of the components of the theoretical framework, necessary for programmatic assembly and customization of integral evaluation algorithms, are as follows:

Base expressions

The "base expression" is the zero-angular-momentum expression of an integral, used as a starting point for a vertical recurrence relation. For the generic integral type $(\mathbf{abc}\cdots)^{(m,n,\cdots)}$, the base expression is some function, f, which corresponds to evaluating the zero-angular-momentum case $(\mathbf{0}_A\mathbf{0}_B\mathbf{0}_C\cdots)^{(m,n,\cdots)}$. The function f may contain simple arithmetic operations, standard mathematical functions and "special" functions, such as the Boys function (section 2.3.4). For example, the base expression for a two-index overlap integral $(\mathbf{a}|\mathbf{b})$ (Eq. 2.155) is

$$\begin{aligned} f(\zeta_a, \zeta_b, \mathbf{A}, \mathbf{B}) &= \left(\frac{\pi}{\zeta}\right)^{3/2} \exp(-\xi|\mathbf{A}-\mathbf{B}|^2) \\ &= (\mathbf{0}_A|\mathbf{0}_B) \end{aligned} \tag{4.1}$$

where $\zeta = \zeta_a + \zeta_b$ and $\xi = \zeta_a\zeta_b/\zeta$.

Base expressions should be numerically computable, but do not need to be in analytic form—any base expression featuring the Boys function is not analytic because it contains an integration (however, this can be efficiently calculated using numerical approximations—see appendix D).

For integral types featuring auxiliary indexes, it may be necessary to evaluate the base expression for a range of values of these indexes to provide a starting point for a recurrence relation.

Vertical recurrence relations (VRRs)

VRRs increment angular momentum in an integral index of the assigned integral, using intermediate integrals with integral indexes that are either unchanged or decremented with respect to the indexes of the assigned integral. Intermediate integrals may also feature auxiliary indexes, that can be incremented or decremented. These have the general form

$$((\mathbf{a}+\mathbf{W}_i)\mathbf{bc}\cdots)^{(m)} = \sum_k C_k((\mathbf{a}-\mathbf{X}_{ki})(\mathbf{b}-\mathbf{Y}_{ki})(\mathbf{c}-\mathbf{Z}_{ki})\cdots)^{(m\pm M_k)} \tag{4.2}$$

where $(\mathbf{abc}\cdots)^{(m)}$ is a generic integral type, C_k is a coefficient for integral intermediate k, which may depend on the exponents and centres of the integral indexes (and other parameters associated with the integral type) and $i = x, y$ or z. $\mathbf{W}_i$, $\mathbf{X}_{ki}$, $\mathbf{Y}_{ki}$ and $\mathbf{Z}_{ki}$ are angular momentum vectors defining the increments and decrements in angular momentum for integral indexes, e.g. $\mathbf{X}_{kx} \equiv (X_k, 0, 0)$ (a generalized form of the notation used in Ref. 1, where $\mathbf{1}_i \equiv (1, 0, 0)$). The allowed values for these angular momentum vectors are

$$|\mathbf{W}_i| > 0 \qquad M_k \geq 0 \qquad |\mathbf{X}_{ki}|, |\mathbf{Y}_{ki}|, |\mathbf{Z}_{ki}| \geq 0.$$

For clarity, only one auxiliary index has been included in Eq. 4.2, though more than one is possible.

Horizontal recurrence relations (HRRs)

This type of recurrence relation shifts angular momentum from one integral index to another, both of which must represent functions of the same electron coordinate. This takes the general form

$$(\mathbf{a}(\mathbf{b}+\mathbf{1}_i)|\mathbf{c}\cdots) = ((\mathbf{a}+\mathbf{1}_i)\mathbf{b}|\mathbf{c}\cdots) + \mathbf{AB}_i(\mathbf{ab}|\mathbf{c}\cdots) \tag{4.3}$$

where angular momentum has been incremented in index $|\mathbf{a})$ of the general assigned integral $(\mathbf{ab}|\mathbf{c}\cdots)$ and the vertical bar, $|$, is used to emphasize that indexes $|\mathbf{a})$ and $|\mathbf{b})$ represent functions of the same electron coordinate.

Translational recurrence relations (TRRs)

This type of recurrence relation requires that the integral type satisfies the translational invariance relation (Eq. 2.153) and arises directly from the application of the differential operators in this relation. For an integral with standard primitive Cartesian Gaussian indexes

and no other dependence on the centres of these Gaussians, the TRR has the form

$$\begin{aligned}((\mathbf{a}+\mathbf{1}_i)\mathbf{bc}\cdots) = \frac{a_i}{2\zeta_a}((\mathbf{a}-\mathbf{1}_i)\mathbf{bc}\cdots) \\ - \frac{\zeta_b}{\zeta_a}(\mathbf{a}(\mathbf{b}+\mathbf{1}_i)\mathbf{c}\cdots) + \frac{b_i}{2\zeta_a}(\mathbf{a}(\mathbf{b}-\mathbf{1}_i)\mathbf{c}\cdots) \\ - \frac{\zeta_c}{\zeta_a}(\mathbf{ab}(\mathbf{c}+\mathbf{1}_i)\cdots) + \frac{c_i}{2\zeta_a}(\mathbf{ab}(\mathbf{c}-\mathbf{1}_i)\cdots) \\ + \cdots\end{aligned} \tag{4.4}$$

In general, the form of the TRR depends on the result of applying the differential operators in Eq. 2.153, which may vary between integral types if the dependence on the centres of integrals indexes is not straightforward.

This definition is provided for completeness, since support for TRRs has not yet been implemented in Intception.

Contraction

"Contraction" is the construction of contracted GTO integral indexes from linear combinations of primitive Gaussian integral indexes, with primitive exponents and contraction coefficients defined by some basis set. This takes the general form

$$|\mathrm{a}) = \sum_{k=1}^{K_a} d_k |\mathbf{a}_k) \tag{4.5}$$

where $|\mathrm{a})$ is a contracted GTO index with degree of contraction K_a, $|\mathbf{a}_k)$ is a primitive Gaussian index with exponent ζ_k, and d_k are contraction coefficients. To produce fully contracted molecular integrals from primitive integrals, all integral indexes of a given integral type must undergo contraction.

Though the notation of Eq. 4.5 indicates that the indexes use a Cartesian coordinate system (see section 2.3.2), this definition of contraction is the same for Cartesian and spherical coordinate systems.

The radial normalization factor $N_{\zeta\ell}$ of Eq. 2.133 may be folded into the contraction coefficients such that $d'_k = d_k N_{\zeta_k \ell}$, but could equally be applied separately, prior to contraction, so that $|\mathbf{a}_k)$ is a normalized primitive Cartesian Gaussian function.

Spherical transformation

This is the transformation of Cartesian GTO integral indexes into spherical GTO integral indexes, i.e. the transformation between Cartesian and solid harmonic representations of the angular part of the basis function (see section 2.3.1). This takes the general form

$$\varphi_a(\mathbf{r}; \ell_a, m_a, \mathbf{A}) = \sum_{\mathbf{a}} C_{\mathbf{a}}^{\ell_a m_a} \phi_a(\mathbf{r}; \mathbf{a}, \mathbf{A}) \tag{4.6}$$

where $\varphi_a(\mathbf{r}; \ell_a, m_a, \mathbf{A})$ is a spherical GTO and $\phi_a(\mathbf{r}; \mathbf{a}, \mathbf{A})$ is a Cartesian GTO (these may be primitive or contracted). $C_{\mathbf{a}}^{\ell_a m_a}$ is a spherical transformation coefficient, which transforms the angular part of the GTO from a Cartesian polynomial to a solid harmonic (see Eq. 2.131

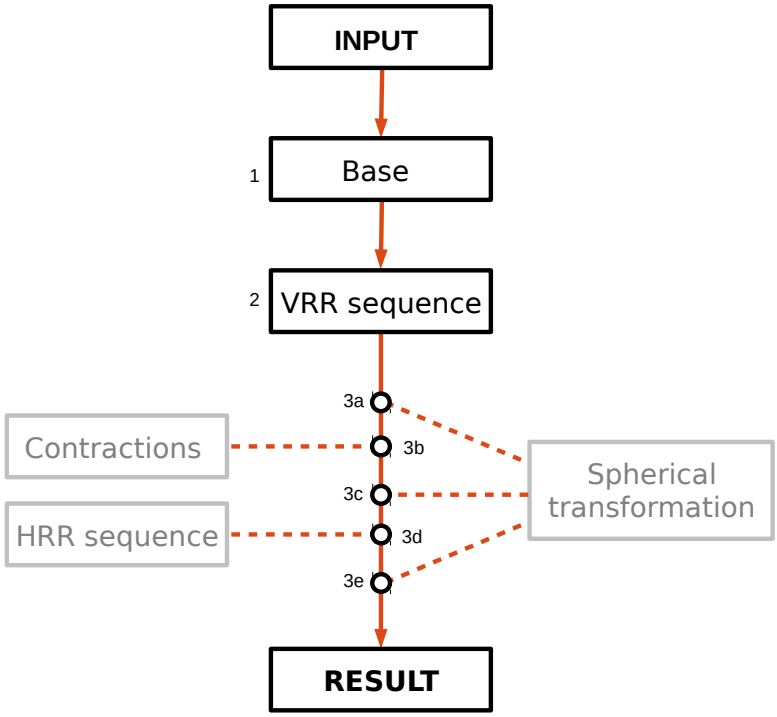

Figure 4.1: A schematic of a generic integral "assembly line" resulting from the combination of the different components of the integral evaluation framework described in section 4.2.2. Black-outlined boxes are always used, while grey-outlined boxes are optional processes which are not needed in every case. The orange arrows represent progress along the assembly line, and dashed lines connect to this line where optional processes can be applied. The integral "assembly line" is described in the main text (referring to the numbered steps in the figure).

and appendix A). As with the preceding definition for contraction, this definition makes no assertions regarding normalization. The angular normalization factor $N^S_{\ell m}$ of Eq. 2.131 (defined in Eq. A.13) could be incorporated into the transformation coefficients such that $C^{\prime\ell_a m_a}_{\mathbf{a}} = C^{\ell_a m_a}_{\mathbf{a}} N^S_{\ell_a m_a}$, though this is not required in this definition.

General algorithm

The various components of the integral evaluation framework can be combined to form a generic integral "assembly line", as pictured in Fig. 4.1. This generic sequence of operations can be used to construct molecular integrals over GTOs of any type for which at least a base expression and VRR expressions are known.

To obtain integrals with angular momentum greater than zero using the assembly line always requires the evaluation of a base expression and application of VRRs (steps 1 and 2 in Fig 4.1). It is then possible to apply several optional operations (steps 3a to 3e), depending on the nature of the integrals being evaluated. For contracted integrals, contraction (3b) is required, while for integrals with spherical indexes, spherical transformation (3a, 3c, 3e) is necessary. For integral types with products of indexes on the same electron coordinate, a HRR may be applied after contraction (3d). The sequence of operations depicted in Fig. 4.1 can therefore be used to evaluate primitive and contracted integrals, with Cartesian or spherical indexes.

The "assembly line" depicted in Fig. 4.1 is not the only possible sequence that can be constructed from the various algorithm components defined earlier. For example, it is possible that a sequence of HRRs could be interspersed with contraction and spherical transformation operations. It is also possible that a contraction step could be added after any HRRs have been applied. It is unlikely that this would be useful, however, since the key advantage of the HRR is that it can be applied after contraction [101] to allow some angular momentum incrementing work to be independent of the degree of contraction (see section 2.3.5). As mentioned in section 2.3.5, contraction of integral indexes may be done even before VRRs are applied, as in the "PRISM" algorithm [98, 108–110], though this requires VRRs with additional auxiliary indexes.

The sequence of operations shown in Fig. 4.1 is the basis of the general integral evaluation framework used in our code generator. It was selected for reasons of simplicity and generality—with only knowledge of the base expression and VRRs for a given integral, it is possible to use this framework to evaluate the primitive or contracted, Cartesian or spherical variants of that integral. It is also possible to implement multiple algorithm variants for a given integral type, e.g. with or without HRR(s).

To illustrate how this framework is applied in practice, consider the specific example of the three-index Coulomb integrals. For the primitive, Cartesian integrals, $(\mathbf{ab}|r_{12}^{-1}|\mathbf{c})$, the process of evaluating a shell-block of integrals of arbitrary angular momentum could proceed as follows (the steps are numbered to correspond with the Fig. 4.1):

1 Base: Evaluate $(\mathbf{0}_A\mathbf{0}_B|\mathbf{0}_C)^{(m)}$ for $m = 0 \ldots \ell_a + \ell_b + \ell_c$.

2 VRRs: Increment $|\mathbf{a})$, $|\mathbf{b})$, $|\mathbf{c})$, to yield $(\mathbf{ab}|\mathbf{c})^{(0)} \equiv (\mathbf{ab}|r_{12}^{-1}|\mathbf{c})$.

Alternatively, a HRR could be used to transfer angular momentum between $|\mathbf{a})$ and $|\mathbf{b})$, resulting in a VRR+HRR integral evaluation algorithm:

1 Base: Evaluate $(\mathbf{0}_A\mathbf{0}_B|\mathbf{0}_C)^{(m)}$ for $m = 0 \ldots \ell_a + \ell_b + \ell_c$.

2 VRRs: Increment $|\mathbf{a})$, $|\mathbf{c})$, to yield $(\mathbf{a'0}_B|\mathbf{c})^{(0)}$ $(\ell_{a'} = \ell_a + \ell_b)$.

3d HRR: Shift angular momentum from $|\mathbf{a'})$ to $|\mathbf{0}_B)$, yielding $(\mathbf{ab}|r_{12}^{-1}|\mathbf{c})$.

If contracted spherical integrals are required, additional steps are necessary. For the spherical, contracted 3-index Coulomb integrals $(\mathsf{ab}|r_{12}^{-1}|\mathsf{c})$, the following algorithm could be used to evaluate shell-blocks:

for all primitive exponent indexes i, j, k:

1 Base: Evaluate $(\mathbf{0}_{A,i}\mathbf{0}_{B,j}|\mathbf{0}_{C,k})^{(m)}$ for $m = 0 \ldots \ell_a + \ell_b + \ell_c$.

2 VRRs: Increment $|\mathbf{a}_i)$, $|\mathbf{c}_k)$, to yield $(\mathbf{a}'_i\mathbf{0}_{B,j}|\mathbf{c}_k)^{(0)}$ $(\ell_{a'} = \ell_a + \ell_b)$.

3a Spherical transform: Transform $|\mathbf{c}_k)$ to yield $(\mathbf{a}_i\mathbf{0}_{B,j}|r_{12}^{-1}|\mathbf{c}_k)$.

3b Contraction: Contract $|\mathbf{a})$, $|\mathbf{0}_B)$, $|\mathbf{c})$, to yield $(\mathrm{a'}0_B|r_{12}^{-1}|\mathsf{c})$.

3d HRR: Shift angular momentum from $|\mathrm{a'})$ to $|\mathrm{b})$, yielding $(\mathrm{ab}|r_{12}^{-1}|\mathsf{c})$.

3e Spherical transform: Transform |a), |b) to yield $(\mathsf{ab}|r_{12}^{-1}|\mathsf{c})$.

Using this general framework for integral evaluation, the problem of constructing an algorithm to evaluate a molecular integral type is converted into the simpler problem of determining how to customize the framework for that integral type. This greatly simplifies the development of a general code generator, since a single system for assembling and customizing components of the framework can be applied to all integral types. The modularity of the framework is also useful for code generation, since the generated code can be conceptually separated into units corresponding to components of the framework.

4.2.3 Three-index Coulomb integrals

The three-index Coulomb integrals, $(\mathsf{ab}|r_{12}^{-1}|\mathsf{c})$, play an important role in testing the code generator described in the following sections. In particular, the three-index Coulomb integrals are useful for determining the performance of generated integral evaluation code. The evaluation of these integrals typically represents one of the more costly steps in density-fitted methods (section 2.4), and thus efficient implementations exist with which the generated code can be compared. In section 4.4.5, generated routines for three-index Coulomb integrals are benchmarked against the efficient implementation in Molpro [151, 152].

The three-index integrals are also useful for testing the flexibility of the code generator, since they offer considerable scope for algorithmic variation. In addition to the VRR-only and VRR+HRR algorithms described in section 4.2.2, a simplified VRR expression may be applied where the integrals are spherical, because certain terms vanish in the transformation [198]. The relative performance of different variants is also considered in section 4.4.5.

The code generated for the results reported in section 4.4, uses the following base expression:

$$\begin{aligned}(\mathbf{0}_A\mathbf{0}_B|\mathbf{0}_C)^{(m)} = {} & 2\pi^{5/2}(\zeta_a+\zeta_b+\zeta_c)^{-1/2}((\zeta_a+\zeta_b)\zeta_c))^{-1} \\ & \times \exp\left(-\frac{\zeta_a\zeta_b}{\zeta_a+\zeta_b}|\mathbf{A}-\mathbf{B}|^2\right) F_m(T)\end{aligned} \tag{4.7}$$

where

$$T = \frac{(\zeta_a+\zeta_b)\zeta_c}{\zeta_a+\zeta_b+\zeta_c}|\mathbf{A}-\mathbf{B}|^2. \tag{4.8}$$

This was derived derived from the base expression for the four-index ERIs (as described in Ref. 1 and appendix C) by setting $\zeta_d = 0$.[1]

For all algorithm variants, the VRR for incrementing angular momentum in |**a**) was

$$\begin{aligned}((\mathbf{a}+\mathbf{1}_i)\mathbf{b}|\mathbf{c})^{(m)} = {} & \mathbf{PA}_i(\mathbf{ab}|\mathbf{c})^{(m)} - \frac{\zeta_c}{\zeta_a+\zeta_b+\zeta_c}\mathbf{PC}_i(\mathbf{ab}|\mathbf{c})^{(m+1)} \\ & + \frac{a_i}{2(\zeta_a+\zeta_b)}\left(((\mathbf{a}-\mathbf{1}_i)\mathbf{b}|\mathbf{c})^{(m)} - \frac{\zeta_c}{\zeta_a+\zeta_b+\zeta_c}((\mathbf{a}-\mathbf{1}_i)\mathbf{b}|\mathbf{c})^{(m+1)}\right) \\ & + \frac{b_i}{2(\zeta_a+\zeta_b)}\left((\mathbf{a}(\mathbf{b}-\mathbf{1}_i)|\mathbf{c})^{(m)} - \frac{\zeta_c}{\zeta_a+\zeta_b+\zeta_c}(\mathbf{a}(\mathbf{b}-\mathbf{1}_i)|\mathbf{c})^{(m+1)}\right) \\ & + \frac{c_i}{2(\zeta_a+\zeta_b+\zeta_c)}(\mathbf{ab}|(\mathbf{c}-\mathbf{1}_i))^{(m+1)}.\end{aligned} \tag{4.9}$$

[1] This differs slightly from the base expression published in Ref. 198, which was found to contain a small error while implementing these integrals using the code generator.

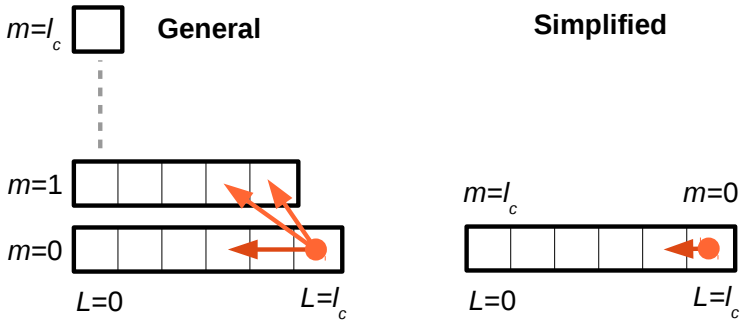

Figure 4.2: Diagram representing the implementation of the general and simplified VRRs for the three-index Coulomb integrals (Eqs. 4.10 and 4.11). Each box represents a shell-block of integrals, with angular momentum L in index $|\mathbf{c})$ and auxiliary index value m. The intermediate integrals required to evaluate a shell-block of assigned integrals with $L = \ell_c$ and $m = 0$ are indicated by orange arrows. For simplicity, only the angular momentum in $|\mathbf{c})$ is shown. In a full implementation, the $|\mathbf{a})$ and $|\mathbf{b})$ indexes would also need to be considered.

where $\mathbf{P}$ is as defined in Eq. 2.139 and $\mathbf{PA}_i \equiv P_i - A_i$. This may also be simply derived from the VRR for the four-index ERIs (Ref. 1 and appendix C) by setting $\zeta_d = 0$ and $\mathbf{d} = \mathbf{0}$. The VRR for incrementing angular momentum in $|\mathbf{b})$ may be obtained by permuting the $|\mathbf{a})$ and $|\mathbf{b})$ indexes in Eq. 4.9.

Two VRRs were used for incrementing angular momentum in $|\mathbf{c})$, depending on the algorithm variant used and whether the integrals were Cartesian or spherical. For all primitive (Cartesian) integrals, and contracted spherical integrals evaluated using a "general" algorithm, the following VRR was used:

$$\begin{aligned}(\mathbf{ab}|\mathbf{c}+\mathbf{1}_i)^{(m)} = &\frac{\zeta_a+\zeta_b}{\zeta_a+\zeta_b+\zeta_c}\mathbf{PC}_i(\mathbf{ab}|\mathbf{c})^{(m+1)} \\ &+ \frac{c_i}{2\zeta_c}\left((\mathbf{ab}|\mathbf{c}-\mathbf{1}_i)^{(m)} - \frac{\zeta_a+\zeta_b}{\zeta_a+\zeta_b+\zeta_c}(\mathbf{ab}|\mathbf{c}-\mathbf{1}_i)^{(m+1)}\right) \\ &+ \frac{a_i}{2(\zeta_a+\zeta_b+\zeta_c)}((\mathbf{a}-\mathbf{1}_i)\mathbf{b}|\mathbf{c})^{(m+1)} \\ &+ \frac{b_i}{2(\zeta_a+\zeta_b+\zeta_c)}(\mathbf{a}(\mathbf{b}-\mathbf{1}_i)|\mathbf{c})^{(m+1)}\end{aligned} \tag{4.10}$$

which can also be derived from the VRR for the four-index ERIs by setting $\zeta_d = 0$ and $\mathbf{d} = \mathbf{0}$. This is referred to as a "general" algorithm because it produces the correct numerical result, regardless of whether the integrals are later spherically transformed.

For spherical contracted integrals evaluated using an "Ahlrichs" algorithm, the VRR for incrementing $|\mathbf{c})$ has the form

$$\begin{aligned}(\mathbf{a0}_B|\mathbf{c}+\mathbf{1}_i)^{(m)} = &\frac{\zeta_a+\zeta_b}{\zeta_a+\zeta_b+\zeta_c}\mathbf{PC}_i(\mathbf{a0}_B|\mathbf{c})^{(m+1)} \\ &+ \frac{a_i}{2(\zeta_a+\zeta_b+\zeta_c)}((\mathbf{a}-\mathbf{1}_i)\mathbf{0}_B|\mathbf{c})^{(m+1)}.\end{aligned} \tag{4.11}$$

This VRR expression was derived from the general expression (Eq. 4.10) by Ahlrichs [198], who recognized that some terms in the general expression cancel during spherical transformation,

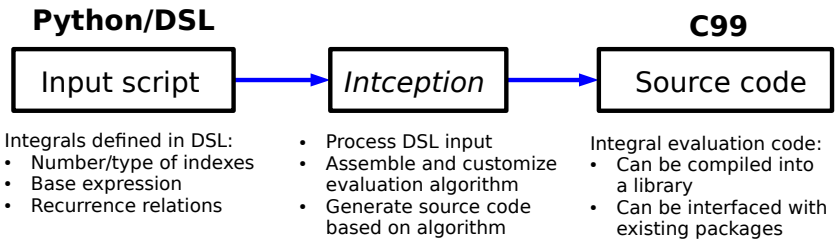

Figure 4.3: Schematic diagram of the process of code generation, starting with integral types defined in the DSL and ending with C source code for the evaluation of those integral types.

and can therefore be neglected for integrals that are destined to be spherically transformed.

In Eq. 4.11, $|\mathbf{b}| = 0$ since in Ahlrichs' original derivation [198] it was assumed that angular momentum would be incremented in $|\mathbf{b})$ using a HRR after the VRRs had been applied to $|\mathbf{a})$ and $|\mathbf{c})$. In the results presented in section 4.4.5, the simplified VRR is also used in a VRR-only algorithm. In this case, angular momentum is incremented in $|\mathbf{b})$ using the general VRR only after it has been incremented in $|\mathbf{c})$ using Eq. 4.11, ensuring $|\mathbf{b}| = 0$ when the simplified VRR is applied.

The use of Ahlrichs' simplified VRR (Eq. 4.11) offers some clear advantages, Firstly, the VRR expression contains fewer terms, and is thus less computationally expensive to evaluate than the general expression. Even when $|\mathbf{b}| = 0$, Eq. 4.10 contains four distinct intermediate integrals, whereas Eq. 4.11 has only two. Second, the implementation of Eq. 4.11 is simpler, since the auxiliary index can be made implicit. Both intermediate integrals in the simplified VRR have $\mathbf{c}$ decremented by one unit of angular momentum, and m incremented by one, i.e. there is only a single m for each value of $L = |\mathbf{c}|$.[2] The value of m can therefore be implicit in the angular momentum of integral index $|\mathbf{c})$. For the general VRR, there are multiple values of m for each value of $L = |\mathbf{c}|$ in the intermediate integrals, and thus m must be considered explicitly in addition to the angular momentum of $|\mathbf{c})$, as illustrated in Fig. 4.2.

4.3 Implementation

4.3.1 Overview

In the section 4.2, a general approach to the construction of algorithms for molecular integral evaluation was described. To automate this approach we developed "Intception", a software package which generates source code for the evaluation of molecular integrals based only on abstract definitions of integral types.

Intception is programmed in Python (version 3 [234]) and makes heavy use of the object-oriented features of the language. A domain-specific language (DSL), constructed using Python classes, is used to compactly represent the abstract molecular integral definitions necessary to generate source code. As described in section 4.3.2, the development of a DSL "on top" of

[2]Throughout this chapter, capitalized L is used to denote intermediate values of angular momentum, while ℓ is used to denote the final value, i.e. on completion of the integral evaluation algorithm.

Python offers significant benefits, particularly with regards to the parsing of algebraic expressions.

The strong object-oriented features of Python were the main motivation for selecting the language for this project, though other features also influenced the choice, including:

Interpreted language
Using an interpreted language avoids the complexities and time-consuming nature of compilation and linking. The availability of an interactive interpreter allows for rapid prototyping.

Widespread adoption
The Python language is available for many computer operating systems and architectures. It is also widely used for scientific applications, with specialist libraries such as SciPy , NumPy and Matplotlib available [235–237]. It is therefore likely that the target audience for Intception (quantum chemists) will have access and experience with Python.

String manipulation
Python provides powerful built-in components for manipulating strings and text. This is particularly important for source code generation, where a large amount of text manipulation and modification is necessary.

The integral evaluation code generated by Intception is intended to be interfaced with new and existing electronic structure software packages. For reasons of speed, these packages are typically written in statically-typed, compiled languages, with the *de facto* standard languages being Fortran and C/C++. To ensure broad compatibility with these packages, the code generated by Intception is output in the C programming language (adhering to the C99 standard [238]).

C is a *lingua franca*, with routines written in C being callable from programs written in many other languages, including C++, Fortran and Python. It is relatively simple to build interfaces for calling C code from Fortran using the `iso_c_binding` module and `bind(C)` attribute (introduced in the Fortran 2003 standard [239]), while C routines may be trivially called from C++ using the `extern "C"` keyword.

The general model for operation used in Intception is summarized in Fig. 4.3. From a user's perspective, the software takes integral types defined in a Python-derived DSL (section 4.3.2) and translates these into C source code for the evaluation of those integral types. In the intermediate steps, the DSL input is analyzed and used to assemble suitable integral evaluation algorithms (using the general integral evaluation framework described in section 4.2.2), which are then translated into a source code representation for output. These intermediate steps are considered in detail in the following sections (4.3.2 to 4.3.5).

The general approach taken by Intception offers some significant advantages for both users and developers, including:

Simple, extensible DSL
The DSL has been designed to be easy to use, and can be easily extended by addition of new classes which inherit from the core classes (see section 4.3.2). Python programmers should find the DSL particularly easy to learn, since the DSL is built "on top" of Python and therefore shares the same syntax and conventions.

Modularity

The DSL provides a standard set of objects and interfaces on which the code generator can act. This allows the DSL and code generator to be considered independently. New or modified code generators could be written using the same set of standard objects and interfaces provided by the DSL, e.g. for generating code in different languages, or using a different integral evaluation framework.

Broad compatibility of output

The output is in the C language (C99 standard [238]), which is broadly compatible with other programming languages and is widely available across hardware and software platforms. In addition, the output routines have a "flat" BLAS-like interface [205], where all arguments are passed as built-in types (or arrays of these), avoiding potential issues relating to the translation of more complicated abstractions (such as objects and aggregate data structures) between languages and software packages (see section 4.3.4).

Before describing the implementation of Intception in detail, it is worth considering the current capabilities and limitations of the software, since this information will be useful to both users and developers of the software. The key capabilities of the software are as follows:

- Generates self-contained libraries of integral evaluation subroutines that can be interfaced with other software packages.
- Implements the key features of the general integral evaluation framework described in section 4.2.2:
  - Base expression evaluation.
  - VRR evaluation up to arbitrary angular momentum.
  - HRR evaluation up to arbitrary angular momentum.
  - Contraction.
  - Spherical transformation.
- Can generate code for a wide variety of integral types with GTO indexes (primitive and contracted, spherical and Cartesian) where Obara-Saika-type recurrence relations and base expressions have been derived.
- Automatic generation of source code for Boys function evaluation (section 2.3.4), based on the implementation described in appendix D.

At the time of writing, the key features of the framework have been implemented, such that Intception is capable of generating code for a very wide range of integral types and evaluation algorithms (see section 4.4), including the main integral types necessary to calculate an SCF energy (see section 2.1.1). However, there remain some desirable theoretical possibilities that are not yet implemented.

At present, the main practical limitations of Intception are:

- All integral indexes of a given integral type must have the same coordinate system. Mixed spherical/Cartesian integrals are not supported.

- Spherical transformation is only possible for contracted integral types. Spherical primitive integral types are not supported.
- Only a single auxiliary index is supported per integral type. The evaluation of the base expression and VRRs for integrals with more than one auxiliary index has not been implemented.
- Only a single HRR may be applied in the integral evaluation algorithm for a given integral. Algorithms featuring more than one application of a HRR (e.g. the Head-Gordon-Pople algorithm for evaluating four-index ERIs [101]) cannot be automatically generated.
- The VRRs used to evaluate a given integral type can only contain integrals of that type. Algorithms featuring VRRs which contain more than one integral type cannot be automatically generated (though this can be achieved by a combination of code generation and manual modification, as described in section 4.3.7 for the two-index kinetic energy integrals).
- Multi-component integrals must be implemented as separate integral types (e.g. the x, y and z components of the dipole moment integrals (a|$\mathbf{r}$|b) are treated as separate integral types).
- Where no auxiliary indexes are present, it must be possible to evaluate the base expression for an integral type using only standard arithmetic operations and mathematical functions available in the C language (such as `exp` and `sqrt`).
- Where an auxiliary index is present, a special function dependent on the auxiliary index may be employed in the base expression, but the only function for which code may be automatically generated is the Boys function (using the implementation described in appendix D).

These limitations exist primarily because other capabilities were prioritised in the development of this version of Intception. None of the limitations are intrinsic to the fundamental design of Intception—it is anticipated that all these limitations could be overcome by modification and extension of the existing codebase.

4.3.2 Domain-specific language

A code generator able to generate code for arbitrary integral types necessarily requires some system of input in which integral types can be defined. The range of integral types, with associated base and recurrence relation expressions, possible in the general integral evaluation framework (section 4.2.2) is vast, requiring a sophisticated and flexible input system. Essentially, some kind of input language is necessary.

A common approach to providing a program with complex input is to specify an input file format, to be parsed by the program. This can be done by creating a custom input file format, as used in Molpro [151, 152] and many other scientific software packages. This has the advantage that the input format can be designed with syntax and formatting specific to the program,

but also requires the implementation of custom routines to parse and manipulate the input. Alternatively, existing languages or data formats, such as XML [240] and JSON [241, 242] can be used. In this case, the input syntax is more general, but the effort of parsing and manipulating the input can be reduced by use of standard libraries for parsing the input. A third strategy, which can combine the advantages of both these approaches, is the use of a domain-specific language (DSL).

In Ref. 243, van Deursen, Klint and Visser define DSLs as follows:

> A domain-specific language (DSL) is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain.

This definition is somewhat vague, and could encompass some sophisticated custom input formats (for example, the Molpro input language). It could be argued, however, that such input formats are designed to *control* a program, rather than *express* a problem domain.

Regardless of whether some input custom file formats can be considered to be DSLs, the DSL used in Intception is distinct from these in that it is an "embedded" DSL (as defined in Ref. 243), which makes use of the syntax and functionality of a host general-purpose language. In contrast, a custom input format is typically read at run-time, and does not utilize the properties of a host language. See Ref. 243 for further details regarding DSLs and examples of their application.

The most significant advantages of Intception's DSL arise because it is embedded in the Python language [234]:

- The Python interpreter is used to parse the DSL, so there is no need to write any kind of language-parsing code.
- Operator precedence is built into the Python interpreter, so mathematical expressions consisting of DSL objects can be automatically converted into trees of binary and unary operations (see Fig. 4.4).
- The DSL is easily extensible using Python's object-oriented features, in particular using class inheritance to construct new classes based on the fundamental DSL classes.
- Since Intception is entirely written in Python, DSL objects can be directly manipulated by the rest of Intception, without the need for any translation.
- As a dynamically-typed, interpreted language, Python allows DSL scripts to written very quickly, without the need to consider explicit variable declaration or compilation.

As an embedded DSL, the language adopts all the syntactic and formatting conventions of the host language, Python (version 3). In addition, since the DSL objects are standard Python objects, the full suite of Python language features is available to manipulate and modify these objects. Intception takes full advantage of this, creating and manipulating DSL objects internally to represent domain-specific entities and processes.

Intception's DSL is designed to allow the abstract objects associated with the integral evaluation framework (section 4.2.2), such as indexes, variables and the molecular integrals themselves to be simply and unambiguously represented. To completely define an integral type, such that source code for its evaluation can be automatically generated, the following information is required:

- The structure of the integral, i.e. the number and type of indexes.
- The base expression (the zero-angular-momentum case).
- Recurrence relation expressions for incrementing each integral index.

Using this information, in addition to an indication of the level of contraction (primitive/contracted) and coordinate system (Cartesian/spherical), an integral evaluation algorithm for a given integral type can be automatically inferred based on the general framework described in section 4.2.2. There is no need to provide an explicit definition of the type of operators involved, or the number of electron coordinates being integrated over, since this information is implicit in the base expression. In addition, it is unnecessary for the DSL to represent the order of component operations (e.g. VRR, HRR, contractions) in the algorithm, since this is implicit in the general integral evaluation framework, as shown in Fig. 4.1.

In the future, it may be desirable to introduce a more sophisticated general integral evaluation framework, requiring the representation of new information in the DSL. For example, if the framework was extended to allow for the use of TRRs (Eq. 4.4), it might be necessary to indicate whether an integral is translationally invariant (Eq. 2.153). For this reason, the DSL is designed to be extensible, allowing new attributes and methods to be added to existing classes and new classes to be introduced.

The application of the DSL is most easily demonstrated by example. Listing 4.1 is an example of the DSL input used to represent the two-index Coulomb integrals $(\mathbf{a}|r_{12}^{-1}|\mathbf{b})$. The example demonstrates the creation of several instances of classes that are part of the DSL (these are always named with the prefix dsl_, and will be generically referred to as "DSL classes", with instances of these referred to as "DSL objects"). Of particular importance is the dsl_integral object "coulomb_2idx", created on line 6 of the source code listing. The dsl_integral class is designed to encapsulate all the information necessary to generate source code for the evaluation of a given integral type, so that a single object for each integral type can be passed to the code generator. For the two-index Coulomb integrals, a dsl_integral object is created with two primitive Cartesian Gaussian indexes, and a single auxiliary index, which are represented by dsl_cartesian_gaussian (ga and gb) and dsl_integer_index objects (m).

In line 7 of Listing 4.1, the base expression is set for the integral type, while in lines 8–19, the two VRRs, incrementing angular momentum in $|\mathbf{b})$ and $|\mathbf{a})$, are defined (the base expression and VRR for two-index Coulomb integrals are reproduced in appendix C, Eqs. C.4 to C.6). Both of these expressions are constructed from DSL objects and built-in Python types.

The benefit of the object-oriented approach used is clearly demonstrated in the definition of the VRR expressions, in lines 9–12 and 15–18 of Listing 4.1. These expressions contain copies of the coulomb_2idx object (created using the "int" method of the dsl_integral class) in which the integral and auxiliary indexes are incremented or decremented. For example the

```
1  base_prefactor = dsl_scalar('base_prefactor',\
2          2.0 * sqrt( pi5 * o_o_xp ) * o_o_xaxb )
3  U = xa * xb * o_o_xp * RAB2
4  m  = dsl_integer_index(name = 'm')
5  boys = boys_f( m, U )
6  coulomb_2idx = dsl_integral(ga,gb,m,name='coulomb_2idx')
7  coulomb_2idx.set_base( base_prefactor * boys )
8  coulomb_2idx.add_vrr('vrr1', gb, \
9      PB * coulomb_2idx.int(ga,gb-1,m+1) + \
10     (gb-1) * o_o_2xb * ( coulomb_2idx.int(ga,gb-2,m) -\
11     xa * o_o_xp * coulomb_2idx.int(ga,gb-2,m+1) ) +\
12     ga * o_o_2xp * coulomb_2idx.int(ga-1,gb-1,m+1)\
13 )
14 coulomb_2idx.add_vrr('vrr2', ga, \
15     PA * coulomb_2idx.int(ga-1,gb,m+1) + \
16     (ga-1) * o_o_2xa * ( coulomb_2idx.int(ga-2,gb,m) -\
17     xb * o_o_xp * coulomb_2idx.int(ga-2,gb,m+1) ) +\
18     gb * o_o_2xp * coulomb_2idx.int(ga-1,gb-1,m+1)\
19 )
```

Listing 4.1: DSL input representing a two-index Coulomb integral and associated Obara-Saika-type VRRs. In this example, ga, gb are dsl_cartesian_gaussian objects, xa, xb, pi5, o_o_xp, o_o_2xp, o_o_xaxb, o_o_2xa, o_o_2xb and RAB2 are dsl_scalar objects and PA, PB are dsl_position objects. The instantiation of these objects occurs prior to the code in this listing, and is omitted for clarity. The dsl_integral object, **coulomb_2idx** is highlighted in red, and contains all the information necessary to generate source code for the evaluation of integrals of this type (primitive or contracted). See appendix C for the base expression and VRR used in this example.

term coulomb_2idx.int(ga,gb−1,m+1) on line 9 of Listing 4.1 produces a copy of the original coulomb_2idx object with index gb decremented by one unit of angular momentum and auxiliary index m incremented by 1, i.e.

$$(\mathbf{a}|\mathbf{b}-\mathbf{1}_i)^{(m+1)}.$$

In this way, VRR expressions defined in the DSL are able to express complicated relationships between instances of an integral with incremented and decremented indexes.

For each call to the add_vrr method of coulomb_2idx in Listing 4.1, the first argument specifies a name for the VRR (to be used in the generated source code) while the second argument specifies the index to be incremented. The third argument is the VRR expression, where the increments and decrements are in a single Cartesian direction. The assigned integral is implicit, and the convention used in the DSL is for this integral to not have any increment applied to integral indexes. Instead of having assigned integral $(\mathbf{a}+\mathbf{1}_i|\mathbf{b})^{(m)}$ on the left of the VRR equation (as with Eq. C.6 in appendix C) the VRR equation takes the form

$$\begin{aligned}(\mathbf{a}|\mathbf{b})^{(m)} = &\mathbf{PA}_i(\mathbf{a}-\mathbf{1}_i|\mathbf{b})^{(m+1)} \\ &+\frac{a_i}{2\zeta_a}\left((\mathbf{a}-\mathbf{2}_i|\mathbf{b})^{(m)}-\frac{\zeta_b}{\zeta_a+\zeta_b}(\mathbf{a}-\mathbf{2}_i|\mathbf{b})^{(m+1)}\right) \\ &+\frac{b_i}{2(\zeta_a+\zeta_b)}(\mathbf{a}-\mathbf{1}_i|\mathbf{b}-\mathbf{1}_i)^{(m+1)}\end{aligned} \tag{4.12}$$

i.e. the convention used in the DSL is that all increments and decrements relative to the assigned integral are expressed on the right of the equation.

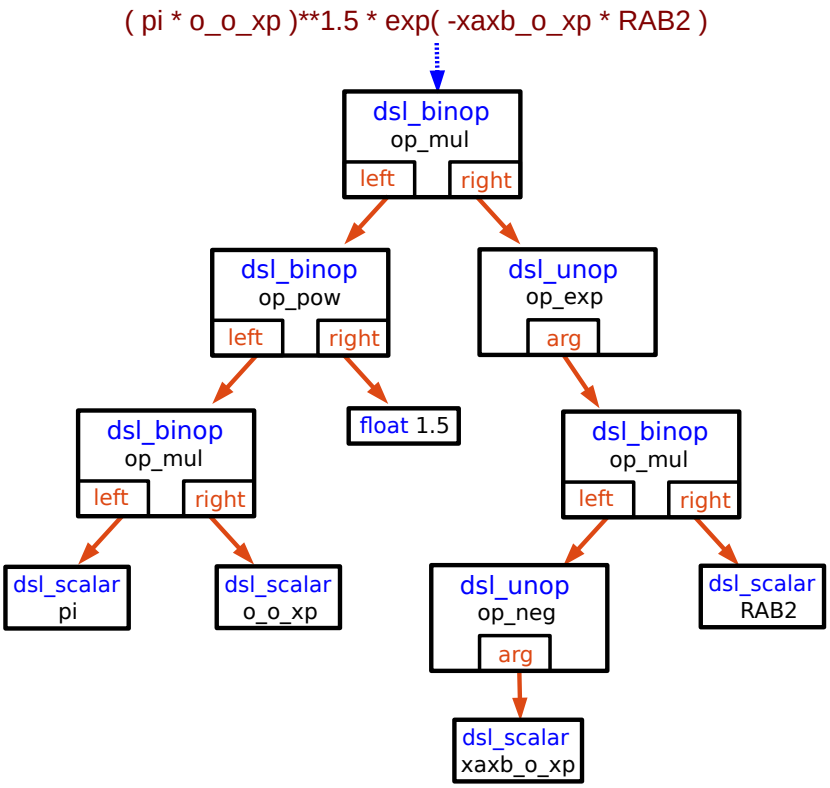

Figure 4.4: Diagram representing the DSL expression tree constructed by the Python interpreter when processing an example expression, consisting of DSL objects and Python built-in objects. The example expression is the base expression for two-index overlap integrals (Eq. 2.155). Instances of DSL classes are represented by boxes, labelled with the class name. dsl_unop and dsl_binop objects represent unary and binary operations, and have an associated operator object (an instance of dsl_op) and one or two methods that return the arguments of the operation (arg or left and right). The orange arrows point to the objects representing the arguments of unary or binary operations, with the tree branches terminating in dsl_scalar objects representing variables, or instances of Python built-in classes representing constants.

In Listing 4.1, the base expression and expressions for VRRs passed to the set_base and add_vrr methods of coulomb_2idx are formatted in the same way as standard Python expressions, but include DSL objects, as well as instances of Python's built-in classes. This is made possible by "operator overloading", a feature of many programming languages which allows the behaviour of operators to be defined depending on the nature of the arguments. For the construction of embedded DSLs, this is particularly useful, since it allows the behaviour of operators in the host language to be re-defined for domain-specific classes or data types.

In Python, the mathematical operators are simply methods of a class, for which the operands are arguments. For example, the __add__ method of an object is called when that object is added to another object, so when the interpreter encounters the expression object1 + object2, it performs the following call:[3]

```
object1.__add__(self, object2)
```

[3]In some circumstances, the __radd__ method of object2 may be called instead (e.g. if object1 is an instance of a class without an __add__ method).

i.e. a call to the __add__ method of object1. Similar special class methods exist for other mathematical operators.[4] The behaviour of instances of a class when acted on by mathematical operators can therefore be tailored by creating custom versions of the corresponding class methods.

In Intception's DSL, methods of DSL classes corresponding to binary and unary operations are overloaded such that they return an instance of the dsl_binop or dsl_unop classes. These classes represent binary and unary operations, with attributes corresponding to the operator and operand(s). When the Python interpreter encounters an expression containing binary or unary mathematical operations on DSL objects, dsl_binop and dsl_unop objects representing these operations are automatically created. For example, if a and b are dsl_scalar objects, then when the Python interpreter evaluates the expression $a * b$, the following function call will occur

```
a.__mul__(self,b)
```

which returns dsl_binop(op_mul,a,b), i.e. an instance of dsl_binop with the operator op_mul and operands a and b (op_mul represents the multiplication operator and is an instance of dsl_op, a DSL class representing operators). The evaluation of a symbolic expression containing overloaded operators therefore leads to the automatic creation of new objects that unambiguously represent the operators and operands involved.

A significant advantage of constructing an embedded DSL using overloaded mathematical operators of the host language is that the host language is responsible for the correct parsing of mathematical expressions. The overloaded mathematical operators retain their standard precedence and associativity, so that expressions containing multiple operations with DSL object operands ("DSL expressions") are evaluated by the interpreter in line with convention. The evaluation of such DSL expressions by the Python interpreter results in the automatic creation of binary expression trees, consisting of dsl_binop and dsl_unop objects, which unambiguously represent the relationship between operators and operands, and the order in which operations should occur. Fig. 4.4 is a diagram of one such tree, constructed when the Python interpreter evaluates the base expression for the two-index overlap integrals (Eq. 2.155).

The automatic construction of these binary expression trees occurs because the expression is evaluated by calling overloaded operator methods in an order consistent with the precedence and associativity of the mathematical operators. For example, for the expression $a + b * c$, where a, b and c are all DSL objects, the multiplication operator has precedence over the addition operator, and thus the expression evaluates to

```
dsl_binop( op_add, a, dsl_binop( op_mul, a, b ) )
```

where op_add and op_mul are dsl_op objects representing the addition and multiplication operators.

Another facet of Intception's DSL is the use of class inheritance to represent relationships between classes representing different abstract entities, as represented in Fig. 4.5. Inheritance allows classes to be derived from other classes, such that the derived "child" class inherits properties from the "parent" class. In Python, child classes inherit methods and attributes

[4] For further details of operator overloading for various operator types in Python, see the Python online documentation [194].

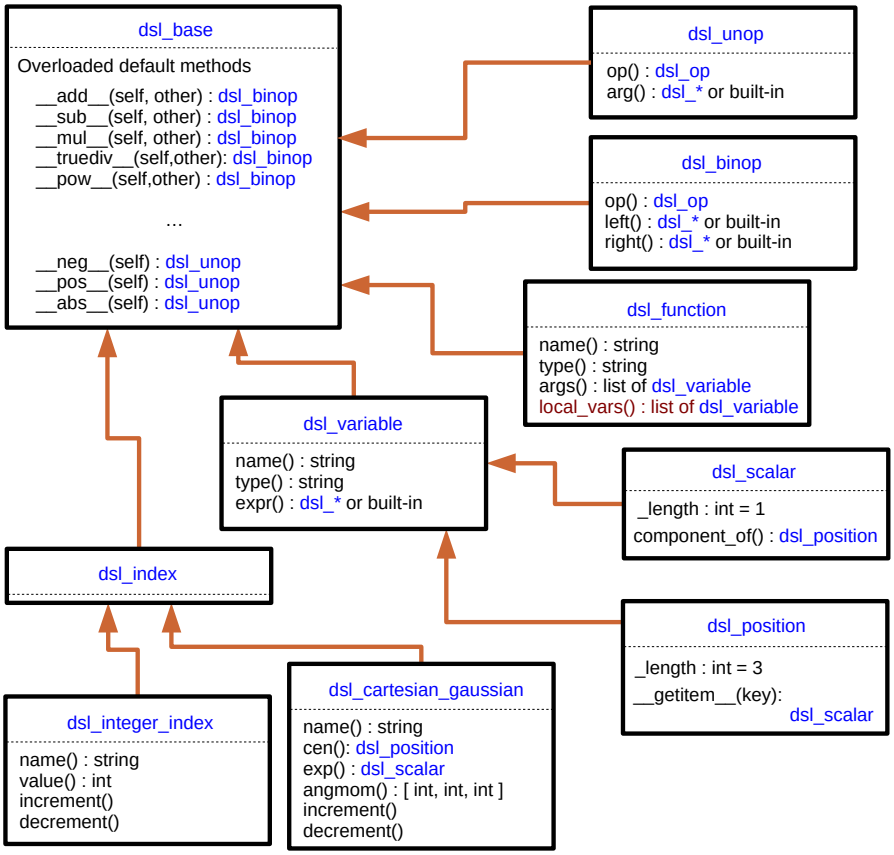

Figure 4.5: Diagram demonstrating the main class inheritance relationships between classes defined in the DSL. Each box represents a class, named above the dashed line. Below the dashed line, are attributes and methods associated with each class, presented in the format attribute : type = default value and method(arg1,arg2): return type. For clarity, only a selection of methods and attributes for each class are included, and Python default methods are only shown where they have been overloaded for that particular class. Orange arrows indicate inheritance, with the arrow pointing from child class to parent class.

from parent classes, and may also override or customize the methods of the parent class. In Intception's DSL, inheritance has two main functions:

Avoiding code reuse

Inheritance allows child classes to reuse code of parent classes. This is particularly useful for overridden operator methods, since for most classes the behaviour will be the same (returning an instance of dsl_binop or dsl_unop). Fig. 4.5 demonstrates how the overridden operators are inherited by many DSL classes from the dsl_base class.

Hierarchical type-checking

The Python built-in function isinstance (see the online documentation [194]) returns True if an object is an instance of a particular class *or is derived from that class* (and False otherwise). This allows type-checking to occur at different levels in an inheritance hierarchy. For example, isinstance would return True when checking for dsl_variable objects for instances of both dsl_scalar and dsl_position, since these are derived from dsl_variable (see Fig. 4.5).

The use of class inheritance also allows the DSL to be easily extended. For example, if a new type of integral index was required, a new class, derived from the dsl_index class could be created. This would inherit some general methods and attributes associated with indexes from dsl_index, and would also inherit the overridden operators (which dsl_index inherits from dsl_base), allowing instances of the new class to be used in mathematical expressions. Further down the inheritance hierarchy, classes become more specialized, offering methods and attributes that are more specific to the abstract entities they represent. For example, the dsl_cartesian_gaussian class provides methods which return the angular momentum, centre and exponent of a Cartesian Gaussian index, while dsl_integer_index has instead a method for returning the value of an integer index. Classes higher up in the hierarchy are more general: dsl_variable provides a method which returns the type of a variable (e.g. integer, floating point number), which is inherited by both dsl_scalar and dsl_position, though these child classes represent different entities (scalar variables and three-dimensional spatial vectors).

The preceding description of the DSL used in Intception is intended as an introduction to the essential features and rationale behind the language. A full specification of the DSL language is beyond the scope of this document—this information is available in the source code and documentation for Intception.[5]

4.3.3 Input structure

The input for Intception is written using the DSL, and defines the integral types for which code should be generated. Unlike many other programs, where input is read from a data file at run-time, the input for Intception is a program itself. In this program, DSL objects are created, corresponding to the integral types for which code should be generated, and the code generator is called with these objects as arguments. These "input scripts" are executed by the Python interpreter.

Listing 4.2 illustrates the basic structure of a typical input script used to generate source code for a set of integral types. Most of the script is concerned with the preparation of dsl_integral objects that can be passed to the code generator. In this case, some pre-defined dsl_integral objects, provided with Intception, are imported (lines 9–12) and an additional dsl_integral object is created to represent the two-index Coulomb integrals ("coulomb_2idx", lines 14–26). The definition is truncated for clarity in this source code listing—a more complete version can be seen in Listing 4.1. The indexes (lines 15–18) and variables (lines 20–22) defined in the script are used in the construction of the coulomb_2idx object. Once the dsl_integral objects have been imported, or defined using the DSL, they can be passed to the code generator (lines 28–38).

The code generator is represented by the generator class. To initialize the generator for a given set of integrals, a new instance of the generator class is created (labelled "gen" in Listing 4.2), with dsl_integral objects passed as arguments to the constructor. A generator_options object ("options") is also passed to the new instance of generator. The generator_options class encapsulates options that are global to the generated library of integral evaluation routines, or

[5] At the time of writing, the source code is not publicly available, though it is anticipated that this will be publicly released under an open source license in the near future.

```
### Import Python standard modules ###
import math

### Import Intception modules ###
from intception.dsl import *
from intception.dsl_extensions import dsl_integral
from intception.generator.generator import *

### Import pre-defined integral types ###
from intception.integrals.standard import \
        overlap_2idx, nuclear_2idx,\
        coulomb_3idx_hrr_ahlrichs

### Definition of additional integral types ###
# Define some indexes required for definitions
ga = dsl_cartesian_gaussian('a', constant = True)
gb = dsl_cartesian_gaussian('b', constant = True)
m  = dsl_integer_index(name = 'm')

# Define variables and constants required for definitions
xp = dsl_scalar('xp', xa + xb )
# ... etc

# Define integral types
coulomb_2idx = dsl_integral(ga,gb,m,name='coulomb_2idx')
# ... etc

### Set generator options and create generator object ###
options = generator_options( \
                  output_directory = "./output_directory/", \
                  contracted = True,\
                  spherical_transformed = True )
gen = generator( nuclear_2idx, \
                  overlap_2idx, \
                  coulomb_3idx_hrr_ahlrichs, \
                  coulomb_2idx, \
                  opt = options )
gen.out() # Output source code
```

Listing 4.2: Example code illustrating the general structure of an Intception input script. Using this input script the code to evaluate several types of spherical, contracted molecular integrals would be generated. This script uses pre-defined dsl_integral objects for two-index overlap, two-index nuclear attraction and three-index Coulomb integrals (labelled overlap_2idx, nuclear_2idx and coulomb_3idx_hrr_ahlrichs). An additional dsl_integral object, representing the two-index Coulomb integrals (coulomb_2idx) is defined in the script, though the code is truncated for clarity (see Listing 4.1 for more detail).

directly relate to code generation (e.g. the directory into which generated source code should be output), and is described in more detail in section 4.3.5. Finally, the out method of the generator object is called to output source code.

In general, an input script for generating code for a set of desired integral types using Intception can be constructed using the following recipe:

1. Import Intception modules for DSL and generator classes.
2. Import any pre-defined dsl_integral objects representing integral types.
3. Define any additional integral types:
   - Create integral and auxiliary indexes using DSL classes.
   - Create associated variables and constants using DSL classes.
   - Create dsl_integral objects representing integral types that are not pre-defined, and define base expression and RRs for these.
4. Create generator_options object with appropriate settings.
5. Create generator object, passing the generator_options and dsl_integral objects as arguments.
6. Call the out method of the generator object to output the source code.

4.3.4 Output structure

When the out method of a properly initialized instance of generator is called (see Listing 4.2), a self-contained library of functions for evaluating the requesting integral types is generated. For each integral type requested (each dsl_integral object passed to the generator), a separate source code and header file is created, containing the source code for the function which evaluates that integral type, as well as integral-type-specific supporting functions. In addition, a set of global source and header files are created, which contain supporting functions and data that are shared across integral types, for example, for evaluating the Boys function (section 2.3.4). The structure of the generated library is summarized in Fig. 4.6.

In normal usage, the generated source code would be compiled into a single library file for linking to an external program during compilation (e.g. an electronic structure package), as shown in Fig. 4.6. For each integral type included in the library, there are two functions that are intended to be called by the external program, with names ending "main" and "work_array_size" (these endings are prefixed with a unique name for the integral type).[6] The main function evaluates a shell-block of integrals of the corresponding type and places these in a memory location specified in the function arguments.

In order to evaluate a shell-block of integrals using an algorithm based on the general framework (section 4.2.2), a scratch space for temporarily storing intermediate quantities is required. The required size of this "work array" for evaluating a given shell-block varies depending on

[6] All Intception function names are also prefixed by the phrase "intception_integral" to avoid namespace collisions with functions in external programs.

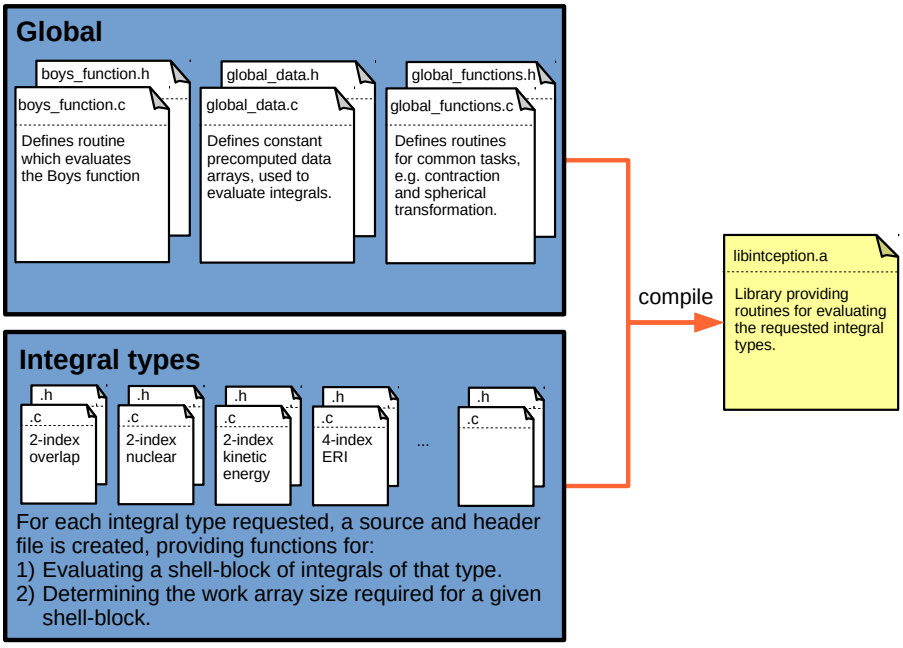

Figure 4.6: Diagram showing the organization of source code generated by Intception into C source and header files. When the out method of a generator object is called in an input script (see section 4.3.3), a set of integral-type-specific source and header files are generated with a corresponding set of "global" source and header files. These files would typically be compiled into a library, for linking to an external program.

the integral type, integral evaluation algorithm used, and nature of the shell-block itself. The work_array_size function complements the main function for an integral type, returning the size of work array needed to evaluate a given shell-block. An external program calling the main function to evaluate a shell-block of integrals can therefore use the work_array_size function to determine the amount of memory required for the work array. For example, if the external program needs to evaluate a number of shell-blocks in sequence, it can use work_array_size to determine the maximum amount of memory needed for all the shell-blocks and then only perform a single memory allocation for the largest amount of memory returned by work_array_size. In this way, costly allocation and deallocation of arrays inside Intception library functions is avoided and responsibility for memory management is left to the external program.

The interfaces for the main and work_array_size functions are generated in a systematic and predictable way based on the information needed to evaluate each integral type. Listing 4.3 shows an example of these interfaces, as generated for the spherical, contracted four-index ERIs. For the spherical, contracted four-index ERIs, $(\mathsf{ab}|r_{12}^{-1}|\mathsf{cd})$, there are four integral indexes, each of which must be specified in the function interfaces. To characterize a single spherical contracted GTO index $|\mathsf{a})$, the following arguments are required by the main function (lines 1–15 of Listing 4.3):

double * xa : An array of primitive exponents, ζ_a.

double cA[3] : The (Cartesian) spatial coordinate, $\mathbf{A}$, of the GTO centre.

int la : The total angular momentum, ℓ_a of the GTO.

int na : The number of angular momentum components (spherical) corresponding to the angular momentum, ℓ_a.

```
1  void intception_integral_eri_4idx_abcd_m_main(
2    double * xa, double cA[3], int la, int na,
3    int iskipa, int na_prim, int na_cont, int iskipa_cont,
4    double * contract_array_a, double * sph_trans_array_a,
5    double * xb, double cB[3], int lb, int nb,
6    int iskipb, int nb_prim, int nb_cont, int iskipb_cont,
7    double * contract_array_b, double * sph_trans_array_b,
8    double * xc, double cC[3], int lc, int nc,
9    int iskipc, int nc_prim, int nc_cont, int iskipc_cont,
10   double * contract_array_c, double * sph_trans_array_c,
11   double * xd, double cD[3], int ld, int nd,
12   int iskipd, int nd_prim, int nd_cont, int iskipd_cont,
13   double * contract_array_d, double * sph_trans_array_d,
14   double * work_array, double * output_array
15 );
16
17 int intception_integral_eri_4idx_abcd_m_work_array_size(
18   int la, int lb, int lc, int ld,
19   int na_prim, int nb_prim, int nc_prim, int nd_prim,
20   int na_cont, int nb_cont, int nc_cont, int nd_cont
21 );
```

Listing 4.3: Function prototypes for the main and work_array_size functions generated for the spherical, contracted four-index ERIs. The "const" keyword has been omitted from argument specifications in both functions to improve readability. The function interfaces are generated automatically during code generation, based on the information provided in the corresponding dsl_integral object provided in the input script (section 4.3.3).

int na_prim : The number of primitive functions to be contracted (degree of contraction, K_a).

int na_cont : The number of contractions corresponding to the set of primitives.

double * contract_array_a : An array of contraction coefficients, d_k of size na_prim * na_cont and leading dimension na_prim.

double * sph_trans_array_a : An array of spherical transformation coefficients $C_{\mathbf{a}}^{\ell_a,m_a}$ of size na_cart * na_sph and leading dimension na_cart.

In addition, the arguments iskipa and iskipa_cont allow flexible indexing of integrals in the output array, describing the "skip" in elements of the output array for each angular momentum component (iskipa) and contraction (iskipa_cont). The arguments for the other integral indexes follow the same convention. The final two arguments for main (on line 14 of Listing 4.3) are pointers to the locations of the work array (double * work_array) and where the result should be stored (double * output_array).

The definitions of the work_array_size function arguments (Listing 4.3, lines 17–21) for each index are as described for the main function. In this case, the angular momentum (la), number of primitive functions (na_prim) and number of contractions (na_cont) for each integral index is required to calculate the minimum size of the work array, which is returned as an integer.

For other integral types, the interfaces for main and work_array_size follow these basic conventions, with some minor variations. In some cases, certain arguments will be omitted, e.g. for primitive integrals, the contract_array arguments are not needed. In other cases, additional arguments may be added, e.g. for the nuclear attraction integrals, $(\mathbf{a}|r_C^{-1}|\mathbf{b})$, an additional argument is necessary to specify the nuclear centre $\mathbf{C}$. These variations are automatically handled by Intception, which constructs appropriate function interfaces based on the integral type

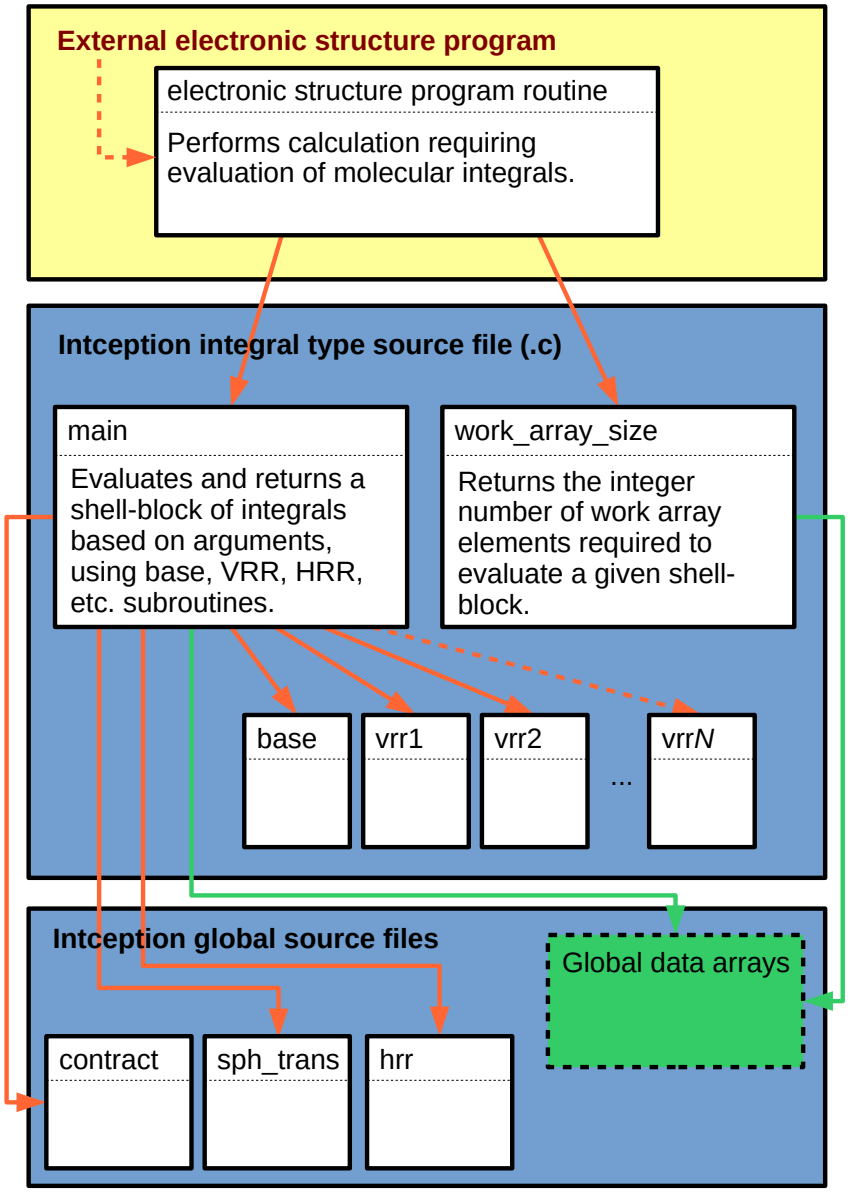

Figure 4.7: Diagram showing the relationships between program units in a generated library of Intception source files, and an external program. Executable program units (functions) are denoted by white boxes connected by orange arrows, with the arrow pointing from the unit issuing a function call to the unit being called. The program units are enclosed in larger boxes representing the separation of code into different files/software packages. The access of global data arrays is also shown using green arrows.

definition provided by each dsl_integral object.

The interfaces for the main and work_array_size functions are deliberately "flat", avoiding the use of composite data types other than simple arrays (which are passed as pointers).[7] This type of interface is designed to maximize compatibility with software written in other languages, where language-specific composite data types may not be easily passed between an external program and Intception.

[7]This style of interface resembles the interface used in implementations of the BLAS standard [205].

```
void intception_integral_overlap_2idx_ab_vrr1a(
  double o_o_2xp, double PA[3], int iwskipa, int lmax_loopa,
  double * work_array, int iwrk0 ){
if( lmax_loopa == 0 ) {
  return;
}
work_array[iwrk0+iwskipa*1]=PA[0]*work_array[iwrk0];
work_array[iwrk0+iwskipa*2]=PA[1]*work_array[iwrk0];
work_array[iwrk0+iwskipa*3]=PA[2]*work_array[iwrk0];
if( lmax_loopa == 1 ) {
  return;
}
work_array[iwrk0+iwskipa*4]=PA[0]*work_array[iwrk0+iwskipa*1]
  +o_o_2xp*1*work_array[iwrk0];
work_array[iwrk0+iwskipa*5]=PA[0]*work_array[iwrk0+iwskipa*2];
work_array[iwrk0+iwskipa*6]=PA[0]*work_array[iwrk0+iwskipa*3];
work_array[iwrk0+iwskipa*7]=PA[1]*work_array[iwrk0+iwskipa*2]
  +o_o_2xp*1*work_array[iwrk0];
work_array[iwrk0+iwskipa*8]=PA[1]*work_array[iwrk0+iwskipa*3];
work_array[iwrk0+iwskipa*9]=PA[2]*work_array[iwrk0+iwskipa*3]
  +o_o_2xp*1*work_array[iwrk0];
if( lmax_loopa == 2 ) {
  return;
}
};
```

Listing 4.4: A generated VRR function which evaluates the two-index overlap integrals $(\mathbf{a}|\mathbf{0}_B)$. The function contains an unrolled loop over Cartesian components of $\mathbf{a}$, which in this case truncates at $\ell_{\max} = 2$ (for normal usage, $\ell_{\max}$ would be higher). The VRR starts with the result of evaluating the base expression, $(\mathbf{0}_A|\mathbf{0}_B)$, stored in work_array[iwrk0]. The "const" keyword has been omitted from the argument list of the VRR function to improve readability.

When an external subprogram calls a main function, to evaluate a shell-block of integrals of a particular type, the main function executes the integral evaluation algorithm (section 4.2.2) that was assembled by the code generator for that integral type. The code that is executed upon calling the main function is arranged in a modular fashion, such that different algorithm components are mostly separated into independent functions. Fig. 4.7 summarizes the calling relationships between different program units.

The separation of different theoretical algorithm components into distinct functions is useful from the perspective of code generation, since it allows the code generation process to be broken down into per-algorithm-component operations. It is also useful in terms of profiling the generated code, since the cost of each algorithm component can be attached to the associated function (see section 4.4.5). However, this approach is likely to have performance costs, since function calls typically have an associated overhead and may limit the degree to which compiler optimization is effective compared to inlined code. This, and other potential performance improvements are, considered further in section 4.5.

A particular advantage of code generation is that it is simple to generate highly repetitious blocks of code which would be tedious and time-consuming to write otherwise. The implementation of VRRs in Intception utilizes this benefit, to generate VRR evaluation code in an "unrolled loop" form (see Ref. 122 (p.168) for a brief introduction to loop unrolling). An example of this implementation of a VRR (for two-index overlap integrals) is presented in Listing 4.4. In this

approach, explicit code for the evaluation of the VRR expression is output for each angular momentum component of the integral index being incremented.

The code shown in Listing 4.4 demonstrates the application of a VRR to incrementing angular momentum in $|\mathbf{a})$ from $|\mathbf{a}| = L_a = 0$ to $L_a = 2$, while $|\mathbf{b}| = L_b = 0$, for the two-index overlap integrals, $(\mathbf{a}|\mathbf{b})$. This corresponds to explicitly outputting a line of code for each angular momentum value of $\mathbf{a}$, i.e.

$(1,0,0|\mathbf{0}) = \mathbf{PA}_x(\mathbf{0}|\mathbf{0})$
$(0,1,0|\mathbf{0}) = \mathbf{PA}_y(\mathbf{0}|\mathbf{0})$
$(0,0,1|\mathbf{0}) = \mathbf{PA}_z(\mathbf{0}|\mathbf{0})$
$(2,0,0|\mathbf{0}) = \mathbf{PA}_x(1,0,0|\mathbf{0}) + (2\zeta)^{-1}(\mathbf{0}|\mathbf{0})$
$(1,1,0|\mathbf{0}) = \mathbf{PA}_x(0,1,0|\mathbf{0})$
$(1,0,1|\mathbf{0}) = \mathbf{PA}_x(0,0,1|\mathbf{0})$
$(0,2,0|\mathbf{0}) = \mathbf{PA}_y(0,1,0|\mathbf{0}) + (2\zeta)^{-1}(\mathbf{0}|\mathbf{0})$
$(0,1,1|\mathbf{0}) = \mathbf{PA}_y(0,0,1|\mathbf{0})$
$(0,0,2|\mathbf{0}) = \mathbf{PA}_z(0,0,1|\mathbf{0}) + (2\zeta)^{-1}(\mathbf{0}|\mathbf{0})$

where the $|a_x, a_y, a_z) \equiv |\mathbf{a})$.

The full VRR expression for the two-index overlap integrals where $\mathbf{b} = \mathbf{0}$ (see section 2.3.3) is

$$(\mathbf{a} + \mathbf{1}_i|\mathbf{0}) = \mathbf{PA}_i(\mathbf{a}|\mathbf{0}) + \frac{a_i}{2\zeta}(\mathbf{a} - \mathbf{1}_i|\mathbf{0}). \tag{4.13}$$

It is clear that some terms in the full VRR expression are zero for some values of $\mathbf{a}$ in the above pseudocode and Listing 4.4. Since each VRR expression evaluation is output on a separate line, the VRR expression can be simplified at the time of code generation. In this way, terms in VRR expression which will always evaluate to zero are explicitly excluded from run-time evaluation.

An alternative implementation of the VRR would be to loop over the full expression (Eq. 4.13), i.e.

```
for a in (1,0,0) to (0,0,2):
    (a|0) = PA_i(a − 1_i|0) + ((a_i − 1)/2ζ)(a − 2_i|0)
```

where it is assumed that the i direction for each VRR expression evaluation is selected on-the-fly. In this case, some terms which evaluate to zero would likely be evaluated at run time (unless the compiler is able to detect and eliminate these terms).

The approach for VRR code generation used in Intception corresponds only to a "partial" loop unrolling. The VRR functions (as pictured in Fig. 4.7) contain the unrolled loop over the angular momentum components of only the index being incremented in the assigned integral. In the example in Listing 4.4, $L_b = 0$ while $\mathbf{a}$ is incremented, so the unrolled loop is the only loop necessary. To calculate a shell-block of $(\mathbf{a}|\mathbf{b})$ with $\ell_b > 0$, a second application of the VRR, incrementing $\mathbf{b}$, could subsequently be applied. In this case, an additional loop over the angular momentum in $|\mathbf{a})$ would be necessary, i.e.

```
for a in (0,0,0) to (0,0,ℓ_a):
    for b in (1,0,0) to (0,0,ℓ_b):
        (a|b) = PB_i(a|b − 1_i) + (a_i/2ζ)(a − 1_i|b − 1_i) + ((b_i − 1)/2ζ)(a|b − 2_i)
```

where the VRR expression is equivalent to Eq. 2.157, with integral indexes $|\mathbf{a})$ and $|\mathbf{b})$ permuted. In the partial loop unrolling approach taken in Intception, only the innermost loop over $\mathbf{b}$ would be unrolled inside a VRR function, while the outer loop would be implemented as a C `for` loop.

```
1  for( iprima=0 ; iprima < na_prim ; iprima=iprima+1 ) {
2    for( iprimb=0 ; iprimb < nb_prim ; iprimb=iprimb+1 ) {
3      // Set per-primitive exponent variables
4      xp=xa[iprima]+xb[iprimb];
5      // ... etc
6      // Setup variables for indexing work_array
7      iwrk0=iwrk1_start+iprima*iskipa_prim+iprimb*iskipb_prim;
8      iwrk1=iwrk0;
9      // Evaluate base expression
10     intception_integral_overlap_2idx_ab_base(
11         RAB2, xaxb_o_xp, o_o_xp3, work_array, iwrk0
12     );
13     // Evaluate VRR for (0|0) to (a|0)
14     intception_integral_overlap_2idx_ab_vrr1a(
15         o_o_2xp, PA, iwskipa, la, work_array, iwrk0
16     );
17     iwrk0=iwrk1;
18     // Evaluate VRR for (a|0) to (a|b)
19     for( idxa=0 ; idxa < nwa ; idxa=idxa+1 ) {
20       intception_integral_overlap_2idx_ab_vrr2b_a(
21           o_o_2xp, PB, idxa, iwskipa, iwskipb, lb,
22           work_array, iwrk0
23       );
24       iwrk0=iwrk0+iwskipa;
25     }
26   }
27 }
```

Listing 4.5: Section of code from the main function generated for spherical, contracted two-index overlap integrals (using a VRR-only evaluation algorithm). In this section, shell-blocks of primitive integrals $(\mathbf{a}|\mathbf{b})$ are evaluated for each combination of primitive function exponents. For each exponent combination, a shell-block of primitive integrals with angular momentum values ℓ_a and ℓ_b is evaluated using the base function and VRR functions, and placed in the work array. Following this section of code in the main function, the primitive integrals are contracted and spherically transformed to yield the final result.

This is demonstrated in Listing 4.5, where lines 18–24 contain a loop over the angular momentum in $\mathbf{a}$, which calls the VRR function (intception_integral_overlap_2idx_ab_vrr2b_a). Inside this VRR function is the unrolled inner-loop over the angular momentum in $|\mathbf{b})$, similar to that presented in Listing 4.4.

In the partially unrolled loop approach, some terms in the VRR may still evaluate to zero at run-time. In Listing 4.5, the value of $\mathbf{a}$ is represented by the variable idxa, which is passed to the VRR function containing the unrolled inner-loop over $\mathbf{b}$. The VRR function cannot know the value of idxa in advance, so must always include the term $a_i(2\zeta)^{-1}(\mathbf{a}-\mathbf{1}_i|\mathbf{b}-\mathbf{1}_i)$ in VRR evaluation lines, even though sometimes $a_i = 0$.

A "full" loop unrolling approach would unroll both the loops over $\mathbf{a}$ and $\mathbf{b}$ and all terms that evaluate to zero could be eliminated during code generation. In early prototypes of Intception, a fully unrolled loop approach was tried, but quickly discarded in favour of the partial loop unrolling approach, since completely unrolling all loops for integral types with multiple indexes typically resulted in problematically large source files.

4.3.5 The generator

Early in the development of Intception, it was decided that the DSL should be divorced from the practical aspects of software development, and represent only the abstract mathematical aspects of molecular integral evaluation. This lead naturally to a modular design, where a "source-code-independent" DSL was processed and analysed by a "source-code-specific" code generator, which dealt with issues relating to the engineering of source code, such as defining functions, constructing loops and indexing arrays.

The "generator" processes the DSL objects (section 4.3.2) defined in the input script (section 4.3.3), constructs an algorithm based on the integral evaluation framework (section 4.2.2) and outputs source code corresponding to this. This role corresponds to the central box (labelled "Intception") in Fig. 4.3.

On being supplied with dsl_integral objects in the input script, the generator must

1. Analyze provided generator_options and dsl_integral objects to determine the appropriate algorithms for evaluating each integral type.

2. Assemble and customize the components of the integral evaluation algorithms.

3. Output valid source code corresponding to an algorithm for each integral type to source and header files.

In addition, the generator must output appropriate global functions and data arrays to support the functions specific to each integral type, organized as in Fig. 4.6.

The process by which abstract integral definitions, represented by DSL objects, are analysed and translated into source code is complex, involving many different interacting objects and procedures. In this section we will therefore only consider the process in a general sense, with emphasis on some key aspects.

The code generation process begins with the creation of an instance of the generator class with a set of dsl_integral objects and a generator_options object as arguments (see for example Listing 4.2, lines 33–37). The generator object's constructor method (__init__) is called with these arguments, and proceeds to call various other generator methods involved in processing and analyzing the input. Among these is the setup_integral_wrappers method, which, for each dsl_integral object, creates an instance of the class integral_wrapper.

The creation of "wrapper" objects around DSL objects is an important aspect of the design of the generator. These wrappers allow a DSL object, which embodies an abstract element of the problem, to be encapsulated in another object, which provides additional source-code-specific methods and attributes. The integral_wrapper class, for example, encapsulates a dsl_integral object, and provides additional attributes such as the filename associated with the generated source code and methods which set up representations of the various functions associated with evaluating that integral type. The encapsulation of objects representing abstract elements in objects which are closer to the source code enables the aforementioned separation of Intception into source-code-independent and source-code-specific components.

The integral_wrapper objects also contain methods which interrogate the associated dsl_integral object to determine the nature of the algorithm required to evaluate the integral type. Based

on this analysis, an instance of a class called algorithm is created and customized such that it represents the integral evaluation algorithm to be used for this integral type. A key attribute of the algorithm object associated with each integral_wrapper object is the so-called "algo_module". The algo_module attribute indicates which integral evaluation algorithm should be used from a set of broad categories of algorithm possible in the general framework (section 4.2.2):

1. VRR-only, no auxiliary index.
2. VRR-only with one auxiliary index.
3. VRR+HRR, no auxiliary index.
4. VRR+HRR with one auxiliary index.

Each of these may be applied to primitive or contracted integrals with Cartesian or spherical angular momentum representations. The broad algorithm categories indicate which components of the framework are required (e.g. VRRs, HRRs, contraction). The distinction between algorithms with and without an auxiliary index is necessary because the implementation of the VRRs and size of work array required for these two cases differs significantly. Once a category has been selected, code generation is a matter of appropriately customizing the algorithm components, e.g. by constructing function calls, loops and conditional structures which ensure the algorithm is correctly executed for an integral type.

In addition to the creation and customization of integral_wrapper and algorithm objects for each integral type, the constructor method for the generator class initiates numerous tasks related to the generation of source code, such as setting up lists of function arguments and local variables, and creating representations of global functions and data arrays. Since these are ancillary to the the main tasks of constructing the integral_wrapper and algorithm objects, we will not consider these in any further detail.

Once the instance of generator has been created (i.e. the constructor has completed execution), the generator is poised to output source code—the DSL input has been analyzed, an appropriate algorithm has been determined for each integral type, and source-code-specific representations have been constructed. The main elements of this process are illustrated in Fig. 4.8.

Fig. 4.9 illustrates the relationship between the generator object and the integral_wrapper objects. For each integral type defined by a dsl_integral object in the input script, the generator creates a corresponding integral_wrapper object. An important facet of this approach is that each integral_wrapper object provides all the information necessary to generate code for a specific integral type, as well as functions for outputting this code. This means that, once an integral_wrapper object has been setup for an integral type, source code specific to that integral type can be generated simply by calling the relevant methods of the integral_wrapper object. For example, to output the source of the main function associated with a particular integral type, represented by an integral_wrapper object labelled wi, the function

```
wi.main_function().source_out
```

should be called. This will use information from the integral_wrapper object wi (in particular the algorithm object) to produce output that is specific to the integral type represented by wi. This

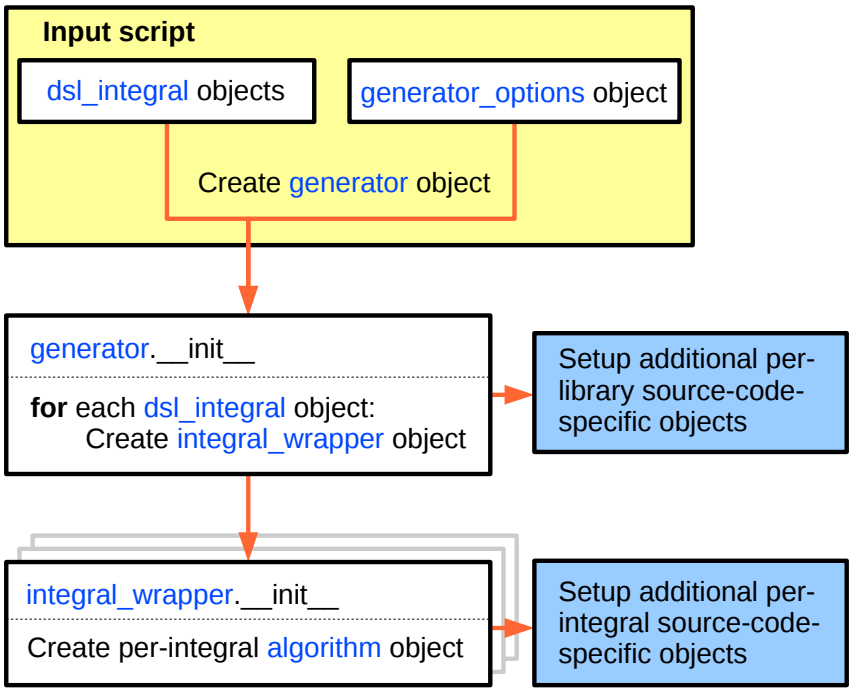

Figure 4.8: Diagram summarizing some of the main processes that occur when an instance of the generator class is created in an input script (section 4.3.3). Arrows loosely represent the flow of program execution and the movement of data.

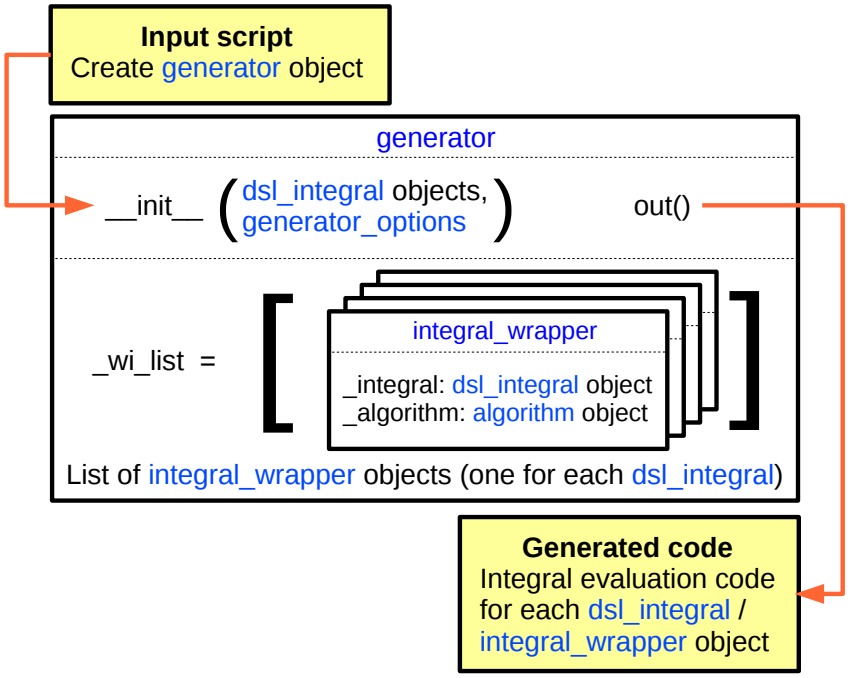

Figure 4.9: Diagram representing the relationship between the generator object and integral_wrapper objects associated with each dsl_integral defined in the input script. Arrows loosely represent the flow of program execution and the movement of data.

strategy of combining methods for outputting source code with data to customize this output is used throughout Intception.

To output integral evaluation source code based on the source-code-specific representations constructed during the creation of a generator object, the out method of this generator object is called (see Fig. 4.9). This results in the output of a collection of source and header files providing functions which evaluate the integral types requested in the input script, to a path set in the generator_options object (as illustrated in Fig. 4.6).

A key step in the execution of generator.out is looping over calls to functions attached to each integral_wrapper object to output code specific to the integral type represented by the object, i.e.

```
for wi in self._wi_list:
    self.integral_class_out( wi )
```

where self._wi_list is a list of integral_wrapper objects. Inside the integral_class_out function, various functions attached to the integral_wrapper objects are called to output source code, e.g.

```
wi.main_function().source_out
```

and,

```
wi.base_function().source_out
```

as well as similar functions for each VRR required. Since these methods take care of customizing the source code for each integral type, using data in the integral_wrapper object, the generator is only responsible for directing the output to the correct files, in an appropriate order. The process for outputting source code for global source and header files is similar.

We have not yet considered in any detail how abstract DSL input is analyzed and translated into representations suitable for source code output. This translation and analysis primarily occurs in the construction of algorithm objects for each integral_wrapper object. As has already been mentioned, algorithm objects represent the integral evaluation algorithm for a particular integral type and are constructed through analysis of a dsl_integral object. Since the overall process of constructing algorithm objects is quite complicated, we will only examine one key aspect of the process here—the determination of the broad algorithm category.

Earlier, it was stated that integral evaluation algorithms possible in the general framework (section 4.2.2) fall into broad categories (see p. 135). In Intception, these broad algorithm categories are represented by Python modules, containing class definitions which provide methods for outputting source code corresponding to the type of algorithm. During the setup of an algorithm object, the algo_module attribute of that object is set to a suitable module, based on analysis of a dsl_integral object. In this way, the broad category of integral evaluation algorithm for each integral type is indicated by the algo_module attached to the corresponding integral_wrapper object. Within the broad algorithm category, the output source code is customized based on additional information attached to the algorithm object, such as the number and ordering of VRR operations.

In the present version of Intception, there are three pieces of information used to determine the broad algorithm category, all of which are easily extracted from the dsl_integral and generator_options objects defined in an input script:

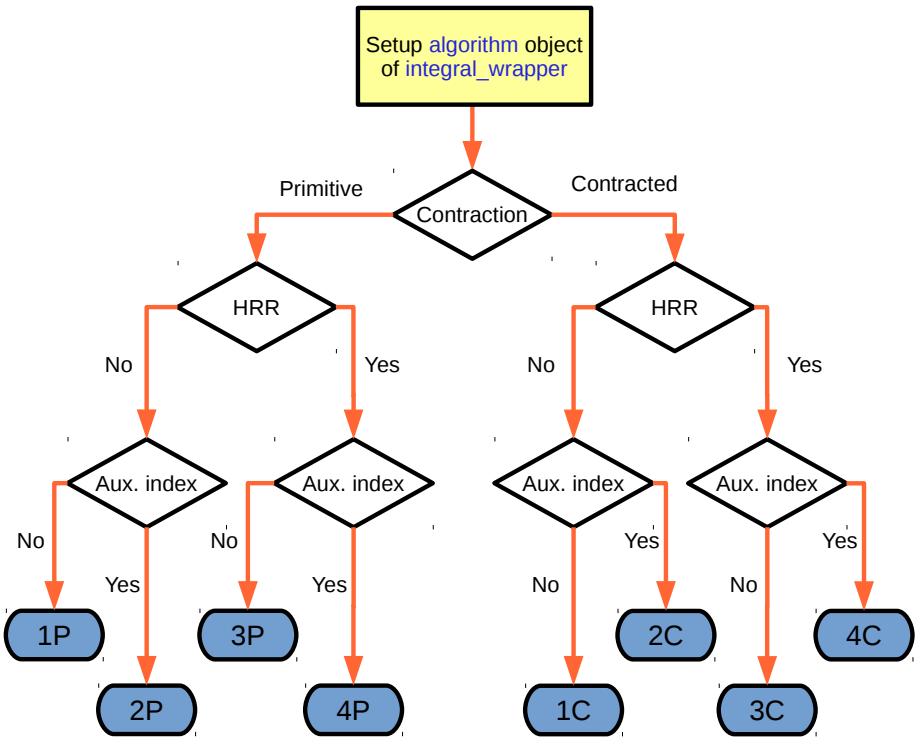

Figure 4.10: A decision tree showing the routes to the eight possible broad algorithm categories possible in the current version of Intception. Outcomes are labelled nP or nC, where n is 1–4 based on the numbering of the categories in the list on p. 135, and "P", "C" indicate primitive and contracted integrals, respectively.

Are the integrals contracted?

Whether code is to be generated for primitive or contracted integrals is determined by an attribute of the generator_options class, an instance of which is attached to each generator object. Accessing this information is simply a matter of determining whether this attribute is True or False.

Is a HRR involved?

Each dsl_integral object carries with it representations of the VRRs and HRRs (these are instances of the dsl_rr class) to be used in the evaluation algorithm. If a HRR is involved, then there will be an object representing a HRR attached. Note that only a single HRR is supported for each integral type at present (section 4.3.1).

Is there an auxiliary index in any of the VRRs?

Each dsl_integral object carries with it a list of auxiliary indexes used in defining that integral. If the length of this list is greater than zero, then there is an auxiliary index in use. Note that only a single auxiliary index for each integral type is supported at present (section 4.3.1).

In all three cases, the result is binary (i.e. true of false), leading to eight possible outcomes (primitive and contracted versions of the four categories listed on p. 135). Once these three pieces of information are determined, the appropriate module can be selected.

Perhaps the most obvious method of selecting an appropriate module would be to use a sequence of conditional tests, e.g.

```
if is_contracted == True and hrr_present == True and auxiliary_present == True :
  algorithm.algo_module = contracted_vrr_hrr_with_auxiliary
```

with one such test for each possible combination of True and False results. Fig. 4.10 is a diagram representing the sequence of conditional tests.

An issue with this approach is that, should an additional parameter be needed to describe future algorithm categories, for example, to allow for integrals with multiple Cartesian components, the entire block of conditional tests would need modifying. This corresponds to adding an entire new level to the decision tree presented in Fig. 4.10. To avoid this issue, Intception uses instances of a class called algo_descriptor to represent broad algorithm categories. The algo_descriptor class has the attributes is_contracted, hrr_present and aux_index_present, which may be set to True or False, and which represent the three binary choices determining the broad algorithm category. An instance of algo_descriptor can therefore completely characterize any of the eight algorithm categories possible at present.

Each module corresponding to broad algorithm category defines a unique instance of algo_descriptor. For example, the algo_descriptor object corresponding to the module for primitive, VRR-only algorithms, with an auxiliary index, is created with the following arguments, is_contracted = False, hrr_present = False and aux_index_present = True. The correct broad algorithm category module can then be selected by creating a new algo_descriptor object for each integral type defined in the input, and comparing this against the pre-defined algo_descriptor objects for each broad algorithm category.[8] The pre-defined algo_descriptor object that matches the per-integral object represents the correct broad algorithm category module for that integral type.

Though this method of selecting algorithm modules might appear more complicated than the use of conditional blocks, it is far more extensible. New parameters describing broad algorithm categories can easily be added to the algo_descriptor class, allowing new broad algorithm modules to be added in the future.

Once a broad algorithm module has been selected and attached to the algorithm object of an integral_wrapper, the basic route through the general integral evaluation framework (Fig. 4.1) for that integral type is set. However, the realization of this algorithm in source code will differ for each integral type, depending on other aspects of the integral definition, such as the number of integral indexes and whether there are any additional arguments (e.g. the centre **C** in the nuclear attraction integrals $(\mathbf{a}|r_C^{-1}|\mathbf{b})$).

Customization of the basic integral evaluation algorithm occurs during code output. Components of the broad algorithm module which are called by generator.out modify their behaviour depending on data stored in the integral_wrapper (and corresponding algorithm) objects for each integral. A simple example of this type of customization is in the output of loops over primitive function indexes for contracted integrals. Before contraction can occur, the base function and VRRs should be applied for all combinations of primitive exponents (Fig. 4.1). As a consequence, loops over primitive function indexes, as seen in Listing 4.5, are necessary—clearly, one loop per integral index is required. Inside the broad algorithm module, the code for outputting these loops is itself a loop, iterating over the number of integral indexes,

[8] This is simply achieved in Python using the built-in dictionary type.

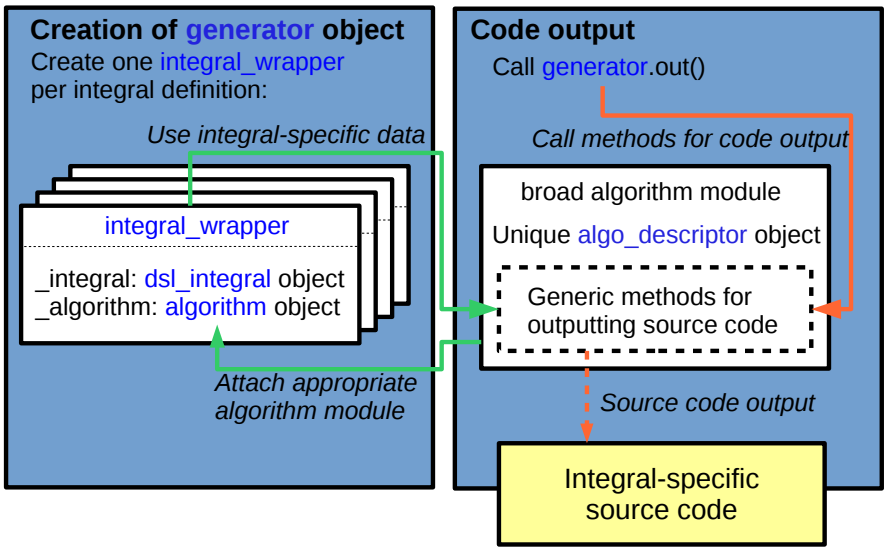

Figure 4.11: Diagram summarizing the process by which the broad algorithm module for a particular integral type is selected, and customized integral-specific source code is output using this module. The two blue boxes represent the two main steps in using the generator—creation of the generator object (and associated integral_wrapper objects), and calling generator.out to output source code. Orange arrows represent the flow of program execution. Green arrows represent sharing of data. During the creation of each integral_wrapper, an appropriate broad algorithm module is attached to the associated algorithm object. Later, during code output, the data in the integral_wrapper object is used to customize the code output by methods of this broad algorithm module.

```
for index in integral_index_list :
    p.forloop(...)

# Per-primitive combination inner loop:
# Output code to evaluate base function, VRRs etc.

for index in integral_index_list :
    p.endforloop()
```

where p.forloop and p.endforloop are functions which output the top and bottom parts of a C for loop structure to a file. The number of nested loop structures output by this code depends on the size of integral_index_list, which is provided by an integral_wrapper object.[9] In this way, the behaviour of the generic code generation methods of a broad algorithm module can be tailored to specific integral types. The process of selecting a broad algorithm module and using integral-specific data to customize code output by this module is summarized in Fig. 4.11.

The preceding overview of the code generating part of Intception is intended as an introduction, emphasising the overarching structure and behaviour of the software over the minutiae of its implementation. Many details and complications have necessarily been omitted—for further information, the source code and documentation of Intception should be consulted (see the footnote on p. 125). It is hoped that the information presented here will provide future developers with a starting point for working with Intception, and users a framework in which

[9]This example has been simplified, with variable names changed and code omitted for clarity.

to understand the behaviour of the software.

4.3.6 Practical application

There are two main prerequisites for running the current version of Intception and compiling the generated code:

A recent version of the Python 3 interpreter
The package has been tested with Python 3.4 [234], and should continue to work with future 3.x versions.

A C compiler
Recent versions of GCC (4.9.2 [244]), Clang (3.5.0 [245]) and the Intel C compiler (icc, 14.0.3 [246]) are all suitable for compiling generated code (see section 4.4.5).

The first step in generating integral evaluation source code is the preparation of a suitable input script. This is a Python script in which integral types are defined in the DSL (section 4.3.2) and the code generator is invoked using these definitions. Instructions on how to construct this script are provided in section 4.3.3, and an example is presented in Listing 4.2.

Once an input script containing definitions of the desired integral types has been created, the script should be executed using the Python interpreter,

```
python3 input_script.py
```

where input_script.py is the filename of the input script. This will cause a self-contained library of C source and header files to be output into the directory specified in the input script. For each integral type defined in the input script, there will be a corresponding source and header file, alongside a number of source and header files associated with global data and functions (see Fig. 4.6).

The generated C source code is intended to be compiled and linked to an external software package which can then call the integral evaluation routines as needed (Fig. 4.7). To simplify the linking process, it is useful to compile the source files into a library. A GNU makefile [247] is distributed with Intception that compiles all source files into object files, and then consolidates these into a static library, libintception.a.[10] This library can then be linked to an external program during compilation, e.g. using GCC

```
gcc --std=c99 -I./intception_src program.c libintception.a -o program.exe
```

where the C program program.c is compiled and linked to the library, producing the executable program.exe. To call generated routines from a C or C++ program, the directory path of the generated header files must be provided—in the above example, this is ./intception_src. For larger programs, consisting of multiple source files and using other libraries, the process of compiling may be more complicated.

To obtain shell-blocks of a particular integral type, the external program should only need to call the generated main and work_array_size functions corresponding to that integral type (as depicted in Fig. 4.7). In calling these routines, the external program must pass basis set and

[10]Creation of a library and the use of GNU make are not requirements—the source files could also be manually compiled and individually linked to the external software package.

geometry information to main and work_array_size functions. The interfaces for these routines are described in section 4.3.4, and a specific example is presented in Listing 4.3.

Some manipulation of the basis and geometry data may be necessary if the representation of this data used by the external program differs from the format required by the generated routines. In particular, where the routines are called from a Fortran program, it is important to ensure that the Fortran and C data types are compatible. This is greatly facilitated by use of the iso_c_binding module, introduced in the Fortran 2003 standard [239], which provides C-compatible Fortran types.

The way in which the generated routines are called will depend on the context, and the intended use of their output. A common scenario would be to build a matrix of all possible AO integrals of a given type. In this scenario, the integrals could be evaluated by looping over calls to main for all possible shell-blocks, and placing the output into suitable locations in the overall matrix. The work array only ever needs to provide enough scratch space to evaluate a single shell-block, so the maximum size required for calculating the overall matrix could be determined by looping over calls to work_array_size for all shell-blocks and allocating an array using the largest value returned by work_array_size.

An important subtlety associated with the evaluation of integrals over GTOs is the application of radial and angular normalization (see section 2.3.1). Different conventions are possible, in which normalization coefficients may, or may not be folded into the contraction and spherical transformation coefficients. The code generated by Intception follows the definitions set out in section 4.2.2, i.e.

- Contraction coefficients supplied by the external program may or may not incorporate radial normalization coefficients, and are always contracted with unnormalized primitive functions (Eq. 4.5).
- Spherical transformation coefficients supplied by the external program may or may not incorporate angular normalization coefficients (Eq. 4.6).

In addition, primitive integrals output by Intception are always over unnormalized primitive Cartesian Gaussians. The upshot of using these conventions is that the external program has complete responsibility for radial and angular normalization. If normalized integrals are required, the program should either present contraction or spherical transformation coefficients that have been pre-multiplied with normalization coefficients, or perform normalization as a separate operation.

4.3.7 Special cases

In section 4.4.4, the results of SCF calculations performed using generated integral routines are reported. As part of these calculations, two-index kinetic energy integrals $(\mathbf{a}|\hat{t}|\mathbf{b})$ $(\hat{t} = -\frac{1}{2}\nabla^2)$ were required to evaluate the core-Hamiltonian matrix (Eq. 2.26). The current version of Intception cannot automatically generate all the code necessary to evaluate the kinetic energy integrals without human intervention.

The Obara-Saika-type VRR for the two-index kinetic energy integrals presented in Refs. 1 and 10 (p.349) contains both kinetic energy and overlap integrals, i.e.

$$\begin{aligned}(\mathbf{a}+\mathbf{1}_i|\hat{t}|\mathbf{b}) = \mathbf{PA}_i(\mathbf{a}|\hat{t}|\mathbf{b}) + \frac{a_i}{2\zeta}(\mathbf{a}-\mathbf{1}_i|\hat{t}|\mathbf{b}) + \frac{b_i}{2\zeta}(\mathbf{a}|\hat{t}|\mathbf{b}-\mathbf{1}_i) \\ - 2\xi\left[(\mathbf{a}+\mathbf{1}_i|\mathbf{b}) - \frac{a_i}{2\zeta_a}(\mathbf{a}-\mathbf{1}_i|\mathbf{b})\right],\end{aligned} \tag{4.14}$$

where $\xi = \zeta_a\zeta_b/\zeta$.

As stated in section 4.3.1, Intception does not currently support VRR expressions which feature integrals of more than one integral type. The base expression

$$(\mathbf{0}_A|\hat{t}|\mathbf{0}_B) = \xi(3-2\xi|\mathbf{A}-\mathbf{B}|^2)(\mathbf{0}_A|\mathbf{0}_B), \tag{4.15}$$

also contains an overlap integral, but this is not a problem, since the expression for $(\mathbf{0}_A|\mathbf{0}_B)$ (Eq. 2.155) can be inserted to yield a form compatible with the current version of Intception:

$$(\mathbf{0}_A|\hat{t}|\mathbf{0}_B) = \xi(3-2\xi|\mathbf{A}-\mathbf{B}|^2)\left(\frac{\pi}{\zeta}\right)^{3/2}\exp(-\xi|\mathbf{A}-\mathbf{B}|^2). \tag{4.16}$$

To carry out SCF calculations using integral routines generated by Intception, the code for evaluating two-index kinetic energy integrals (primitive and contracted) was implemented by manually combining generated code for two-index overlap integrals with partially complete generated code for two-index kinetic integrals. The incomplete two-index kinetic energy integral code was manually modified to call the base and VRR functions generated for the primitive two-index overlap integrals such that the required overlap integrals could be accessed when evaluating the VRR for the kinetic energy integrals (Eq. 4.14). This was a relatively simple modification, and was certainly less time-consuming than completely hand-coding a two-index kinetic energy integral evaluation routine.

The kinetic energy integral code used to obtain results in section 4.4 (including the SCF calculations described in section 4.4.4) should therefore be considered to be "partially generated", since some human intervention was required to combine elements of the generated code for overlap and kinetic energy integrals. Support for VRRs featuring more than one integral type is planned for a future version of Intception.

4.4 Results and Discussion

To test the performance and accuracy of generated integral evaluation code, and assess the flexibility of the code generator, a variety of tests were performed, a representative selection of which are presented in the following sections. It should be noted that Intception is, at the time of writing, a work in progress, and it is anticipated that the software will be improved in the future, particularly with respect to the range of integrals it can automatically generate and the performance of generated code. These results are therefore intended to both illustrate the current capabilities of the software and provide an indication as to where improvements might be made in the future.

Before considering the results in the following sections, it is worth addressing the issue of numerical accuracy. Specifically, it is important to determine a threshold below which we can consider calculated molecular integrals to be of acceptable accuracy. Integrals calculated using code generated by Intception are expected to exhibit some degree of error with respect to exact integral evaluation. Two key reasons for this are:

1. The limited precision and range of the floating point data formats used to store integrals and intermediates (see Ref. 122 (ch.2) for details).

2. The approximate evaluation of the Boys function (section 2.3.4 and appendix D) used in integrals featuring the Coulomb operator.

Since the exact (or arbitrary-precision) evaluation of molecular integrals is generally impractical, the numerical accuracy of the generated code was checked against established integral code in Molpro [151,152]. Molpro's integral code is subject to the same constraints as the generated code in terms of the floating point data format and approximate Boys function evaluation, and is therefore also expected to produce results which differ with respect to exact integral evaluation by some amount. The threshold for accuracy in the generated code should therefore take account of the potential for error in the both the generated code and the established code it is being checked against.

For this work, an accuracy threshold of 10^{-12} was used, i.e. if the value of an integral calculated using the generated code and Molpro differed by less than 10^{-12}, then it was considered that these values were "in agreement" and that the generated code was sufficiently accurate. The 10^{-12} agreement threshold was pragmatically selected based on the assumption that integrals smaller than 10^{-10} are negligible and may be discarded in an electronic structure calculation.[11] A difference of 10^{-12} is then $100\times$ smaller than the largest integral which can be safely discarded.

4.4.1 Computational details

The following hardware and software was used to obtain the results presented in sections 4.4.2 to 4.4.5:

Molpro calculations

A development version of Molpro [151, 152] was used. This was compiled using GCC 4.9.2 [244] (g++ and gfortran), with compiler optimization level -O3. Molpro was linked to ATLAS 3.10.1 for linear algebra (BLAS/LAPACK) [206, 248], which had been tuned for the hardware it was running on. A static library of generated code was compiled using GCC 4.9.2 (gcc), with compiler optimization level -O3, and linked to Molpro in order to call generated routines from within Molpro. Since Molpro is written predominantly in Fortran, interfaces to Intception routines were built using the iso_c_binding module and bind(C) attribute (from the Fortran 2003 standard [239]). All Molpro calculations were run on a desktop computer, running Ubuntu 14.04 LTS (Linux x86_64, kernel 3.13.0), with an Intel

[11] The value of 10^{-10} is suggested in Ref. 10 (pp.398–405) when considering the scaling of the number of "significant" integrals (which cannot be discarded) with system size.

Pentium G3420 CPU (2 cores at 3.20 GHz) and 4 GiB RAM. The calculations were run in serial, using a single core.

In-house SCF code

A currently unreleased code was used to perform SCF calculations using integral code generated using Intception (section 4.4.4). The code was compiled using GCC 4.9.2 [244] (g++), with compiler optimization level -O3 and was linked to Armadillo 5.200.2 [249,250] for linear algebra (which itself was linked to the previously mentioned ATLAS 3.10.1 [206, 248]). A static library of generated integral evaluation code was compiled using GCC 4.9.2 (gcc), with compiler optimization level -O3, and linked to the in-house code to allow the SCF program to make use of generated routines. Since the SCF code is written in C++, the `extern "C"` keyword was used to allow the generated C routines to be used in the SCF code. Calculations performed using the in-house code were run on the same desktop computer used for Molpro calculations, and were run in serial, using a single core.

Benchmarking suite

A benchmarking suite was written to allow the generated code to be tested in a minimal environment, without the potential overhead arising from the larger infrastructure required for electronic structure calculations. This was written in C so that any overhead associated with interfacing between software written in different languages could also be eliminated. As for Molpro and the in-house SCF code, a static library of generated code was compiled and linked to the benchmarking suite. In order to assess the impact of compiler and optimization level on performance (section 4.4.5), the benchmarking suite and generated code were compiled using GCC (gcc, 4.9.2 [244]), Clang (clang, 3.5.0 [245]) and the Intel C compiler (icc, 14.0.3 [246]) with a range of optimization levels. The benchmarking suite was run on a different desktop computer to the Molpro and in-house code calculations, running Ubuntu 14.04 LTS (Linux x86_64, kernel 3.13), with an Intel Core i7-4790K CPU (4 cores at 4.00 GHz, 8 threads with hyperthreading) and 32 GiB RAM. The benchmarking tests were run in serial, using a single core.

4.4.2 Primitive integrals

Source code for evaluating the primitive Cartesian variants of all the integrals in Table 4.1 was generated using Intception. Where VRR-only and VRR+HRR algorithms were available, both were implemented, and the general VRR expression for incrementing angular momentum in $|\mathbf{c})$ of the three-index Coulomb integrals was used (Eq. 4.10). As described in section 4.3.7, the generation of source code for the kinetic energy integrals required some manual intervention.

The source code was compiled into a library and linked to Molpro, as described in section 4.4.1. The generated integral evaluation routines were called to evaluate shell-blocks of primitive integrals, and the results were compared to those obtained using Molpro's built-in routines to evaluate the same integrals. Some integral types were not fully implemented in Molpro and were evaluated using combinations of built-in routines and the Gaussian product theorem (GPT, appendix B). For example, the three-index overlap integrals were implemented

Table 4.1: Integral and algorithm types for which source code was generated and tested. References are provided for the base and VRR expressions used to generate the source code.

Indexes	Label	Integral	Algorithms tested		Base and VRR expressions
			VRR-only	VRR+HRR	
1	Gaussian	$(\mathbf{a})$	✓		Appendix C
2	Overlap	$(\mathbf{a}\|\mathbf{b})$	✓	✓	Eqs. 2.155, 2.157
3	Overlap	$(\mathbf{a}\|\mathbf{b}\|\mathbf{c})$	✓	✓	Ref. 1
4	Overlap	$(\mathbf{abcd})$	✓	✓	Appendix C
1	Nuclear	$(\mathbf{a}\|r_C^{-1})$	✓		Appendix C
2	Nuclear	$(\mathbf{a}\|r_C^{-1}\|\mathbf{b})$	✓		Ref. 1
2	Coulomb	$(\mathbf{a}\|r_{12}^{-1}\|\mathbf{b})$	✓		Eqs. 2.172, 2.173
3	Coulomb	$(\mathbf{ab}\|r_{12}^{-1}\|\mathbf{c})$	✓	✓	Section 4.2.3
4	ERI	$(\mathbf{ab}\|r_{12}^{-1}\|\mathbf{cd})$	✓	✓	Appendix C
2	Kinetic	$(\mathbf{a}\|\hat{t}\|\mathbf{b})$	✓		Eqs. 4.14, 4.15
3	F-F	$(\mathbf{a}\|f_{12}\|\mathbf{b}\|f_{23}\|\mathbf{c})$	✓		Appendix C
3	J-F	$(\mathbf{a}\|r_{12}^{-1}\|\mathbf{b}\|f_{23}\|\mathbf{c})$	✓		Appendix C

Table 4.2: Boundaries for random test data generated for testing primitive integrals. Random data was generated for each primitive Cartesian Gaussian integral index $g(\mathbf{r};\zeta_a,\mathbf{a},\mathbf{A})$, where $\ell_a = |\mathbf{a}|$.

	Exponent	Ang. mom.	Centre		
	ζ_a	ℓ_a	A_x	A_y	A_z
Test set 1					
Minimum	0.01	0	-5.0	-5.0	-5.0
Maximum	10.0	6	5.0	5.0	5.0
Test set 2					
Minimum	0.01	0	-1.0	-1.0	-1.0
Maximum	10.0	2	1.0	1.0	1.0

using Molpro's built-in primitive two-index overlap integral and GPT routines, i.e.

$$(\mathbf{a}|\mathbf{b}|\mathbf{c}) = \sum_{\mathbf{p}} T_{\mathbf{p}}^{\mathbf{ab}} (\mathbf{p}|\mathbf{c}). \tag{4.17}$$

The exponent, angular momentum and centre information for each primitive Cartesian Gaussian integral index was randomly generated. Twenty combinations of randomly generated primitive Cartesian Gaussian indexes were created and a shell-block of each integral type was evaluated for each of the twenty combinations (which could have up to four integral indexes). The random data was constrained to physically realistic values by a set of upper and lower boundaries for the exponent, angular momentum and centre. Two random test sets were used, with different angular momentum and centre boundaries (Table 4.2).[12]

For all the primitive integral types and algorithms listed in Table 4.1, the 10^{-12} agreement threshold (see p. 144) was met for both random tests sets. The maximum difference between integrals evaluated using the generated code, and integrals evaluated using Molpro's routines was often several orders of magnitude smaller than 10^{-12}. For example, for the four-index ERIs (both algorithm types), the maximum difference between integrals evaluated by generated code and Molpro over all twenty test combinations was $\sim 10^{-20}$ for test set 1, and $\sim 10^{-17}$ for test set 2.

[12]In all test cases, the correlation factors, f_{12} and f_{23}, in the F-F and J-F three-electron integral types were represented by single Gaussian geminals with fixed exponents (see appendix C, Eq. C.19) such that $f_{12} = g_{12}^{\mu}$ and $f_{23} = g_{23}^{\nu}$. For F-F integrals $\mu = 0.8$ and $\nu = 0.1$ and for J-F integrals $\nu = 0.5$.

These results indicate that Intception is able to flexibly generate source code for the evaluation of a wide range of primitive integral types and evaluation algorithms (Table 4.1). The generated source code consistently produces results in close numerical agreement with existing integral evaluation source code. Notably, code for the evaluation of the three-electron F-F and J-F integral types, required to implement the DF3-MP2-F12 method described in chapter 3, can be automatically generated, using the base and VRR expressions derived by May [89] (reproduced in appendix C). This demonstrates that the current version of Intception is capable of generating numerically accurate code for more unusual integral types, as might be required in new theoretical methods.

Although timings were recorded for the evaluation of primitive integral types by the generated code and by Molpro, they were not found to be useful. The main reason for this was that the random test sets used were designed to test the numerical accuracy of the primitive integrals across a range of possible inputs, rather than to be representative of typical inputs in electronic structure calculations. The performance of the generated code is assessed in sections 4.4.3 to 4.4.5, where more representative timings were obtained through the use of realistic molecular geometry and basis set data.

4.4.3 Contracted integrals

As for the primitive integrals, source code for evaluating the spherical, contracted variants of all the integral and algorithm combinations in Table 4.1 was generated using Intception, compiled into a library, and linked to Molpro. The simplified VRR expression proposed by Ahlrichs [198] for incrementing angular momentum in $|\mathbf{c})$ was used (Eq. 4.11) to generate the source code for the VRR+HRR variant of the three-index Coulomb integrals, $(\mathsf{ab}|r_{12}^{-1}|\mathsf{c})$. As before, the code for the two-index kinetic energy integrals was partially generated, with some post-generation modification necessary (section 4.3.7).

The generated integral routines were called to evaluate spherical contracted shell-blocks, using basis set and geometry data provided by Molpro. The geometries of benzene, glycine (zwitterion) and H_2O (see appendix I for details) were used, with the cc-pVDZ and cc-pVTZ AO basis sets [51] used for all atoms. For each molecule/basis set combination, all possible shell-blocks of each contracted spherical integral type were evaluated and stored in memory.[13] For example, H_2O with a cc-pVDZ basis set has a total of 24 AO basis functions, so 24^3 units of memory would be needed to store all the possible three-index Coulomb integrals (using the IEEE double precision data format [120], this would occupy 108 KiB of memory). As with the primitive integrals, the spherical contracted integrals were evaluated using generated code and Molpro's built-in routines and per-integral differences were determined. In addition, the root-mean-square deviation (RMSD) of the complete block of integrals calculated using generated code was calculated with respect to the benchmark values provided by Molpro.

For all molecules and basis sets, the per-integral difference between the generated code and Molpro's routines was always less than, or of the order of the 10^{-12} agreement threshold. The RMSD values indicated strong agreement between the generated code and Molpro's routines—in

[13] Four-index integral types were not calculated for benzene/cc-pVTZ and glycine/cc-pVTZ, since impractically large blocks of memory would have been required to store all shell-blocks of these integrals.

all cases these were smaller than 10^{-12}. As with the primitive integrals, these results demonstrate that Intception is capable of flexibly generating numerically accurate source code for a wide range of integral types and algorithms.

Results relating to the performance and numerical accuracy of the three-electron F-F and J-F integral types are presented in Table 4.3. The time taken to evaluate all shell-blocks for benzene and glycine with cc-pVDZ and cc-pVTZ basis sets using the hand-coded implementation described in chapter 3 and the generated code is shown. Alongside this are the maximum per-integral difference and RMSDs for each molecule/basis set combination. In all cases, the generated code offers a 4- to 10-fold decrease in execution time compared to the hand-coded routines, while simultaneously demonstrating numerical agreement with the hand-coded routines. In an electronic structure calculation it is typical to need to evaluate all possible shell-blocks of an integral type for a given geometry and basis set, so these timings can be considered representative of real-world performance.

The results in Table 4.3 demonstrate that Intception is able to generate source code which significantly improves upon the the performance of the hand-coded implementation of the three-electron F-F and J-F integrals (described in chapter 3), while maintaining numerical accuracy. Had Intception been available at the start of the work presented in chapter 3, many hours of work hand-coding a less efficient implementation of these integrals could have been saved.

It should be noted that the data in Table 4.3 represents a comparison with a hand-coded implementation which is known to be inefficient. The timing data therefore does not provide any indication of the competitiveness of generated code with existing, optimized, implementations of the standard integral types required in electronic structure methods. The question of competitiveness is addressed in section 4.4.5, in relation to the three-index Coulomb integrals (see also section 4.2.3).

4.4.4 Electronic structure calculations

To assess the behaviour of code generated by Intception in the context of full electronic structure calculations, generated integral evaluation routines were incorporated into an in-house SCF code, as described in section 4.4.1. Since the in-house SCF code was in the early stages of development, it was possible to build SCF routines around the integral evaluation code generated by Intception. To incorporate generated code into Molpro's SCF routines would have been far more complicated, due to the established, highly optimized nature of that code.

At the time the results were generated, the in-house SCF code supported the self-consistent calculation of the KS-DFT energy (section 2.2.5) without the exchange-correlation contribution, i.e. the Hartree energy (Eq. 2.30). This used density fitting (section 2.4) to evaluate the Coulomb contribution to the SCF equations, i.e.

$$\begin{aligned} J_{\mathrm{ab}} &= \sum_{\mathrm{cd}} P_{\mathrm{cd}} (\widetilde{\mathrm{ab}}|\widetilde{\mathrm{cd}}) \\ &= \sum_{\mathrm{cd}} P_{\mathrm{cd}} \sum_{AB} (\mathrm{ab}|r_{12}^{-1}|A)[\mathbf{J}^{-1}]_{AB}(B|r_{12}^{-1}|\mathrm{cd}) \end{aligned} \tag{4.18}$$

where A, B are density fitting functions, P_{cd} is the density matrix (see section 2.1.1), and $\mathbf{J}^{-1}$

Table 4.3: Comparison of the execution times of a hand-coded implementation (described in chapter 3) and generated code for the evaluation of three-electron integral types. The timings are for the evaluation of all possible shell-blocks of the F-F and J-F integral types for the benzene and glycine molecules (glycine in zwitterionic form, see appendix I) with cc-pVDZ and cc-pVTZ AO basis sets [51] (labelled "D" and "T"). The correlation factors, f_{12} and f_{23}, were represented by single Gaussian geminals, g_{12}^{μ} and g_{23}^{ν}, with fixed exponents $\mu = 0.8$ and $\nu = 0.1$ for F-F integrals and $\nu = 0.8$ for J-F integrals. The maximum per-integral difference and RMSD are for comparisons of the entire matrix of all shell-blocks and are quoted to the nearest order of magnitude. All timing data is averaged over three independent test runs and is quoted to two significant figures.

	Execution time / s		Ratio	Max. diff.	RMSD
	Hand-coded	Intception	$t_{\mathrm{Int}}/t_{\mathrm{Mol}}$		
Benzene, F-F integrals					
D	2.5	0.62	0.25	10^{-14}	10^{-15}
T	11	1.5	0.14	10^{-14}	10^{-16}
Benzene, J-F integrals					
D	2.0	0.37	0.19	10^{-13}	10^{-15}
T	10	2.4	0.23	10^{-12}	10^{-15}
Glycine, F-F integrals					
D	1.2	0.12	0.10	10^{-14}	10^{-15}
T	6.3	0.87	0.14	10^{-14}	10^{-16}
Glycine, J-F integrals					
D	1.2	0.22	0.19	10^{-13}	10^{-15}
T	5.9	1.4	0.23	10^{-12}	10^{-15}

is the inverse of the matrix of two-index Coulomb integrals $(A|r_{12}^{-1}|B)$. Overall, the SCF energy evaluation using the in-house code made use of generated routines for the evaluation of the following spherical, contracted integral types (see Table 4.1):

- Two-index overlap (VRR-only).
- Two-index kinetic energy (VRR-only).
- Two-index nuclear attraction (VRR-only).
- Two-index Coulomb (VRR-only).
- Three-index Coulomb integrals (VRR+HRR, with simplified VRR, Eq. 4.11).

Additionally, generated code for the four-index ERIs (VRR+HRR) was used to perform integral screening using Schwarz's inequality (Eq. 2.192), with a screening threshold of 10^{-12}.

The Hartree energy calculated using the in-house code was compared to the same energy calculated using Molpro. This was evaluated using Molpro's density-fitted KS-DFT routines [143], but setting the exchange-correlation functional uniformly to zero. Though this resulted in the evaluation of the Hartree energy expression (Eq. 2.30), the process by which this was calculated involved some additional redundant work associated with evaluation of the unused exchange-correlation functional. As a consequence, the execution times for the two programs could not be compared in any detail.

Table 4.4 contains the SCF (Hartree) energies calculated for several molecules, using the in-house code (with generated integral routines) and Molpro (using built-in routines). These

Table 4.4: Hartree energies calculated for H_2O, glycine (zwitterion) and naphthalene (see appendix I) with cc-pVDZ [51] AO basis and auxiliary basis from Refs. 251, 252 (denoted "Ahlrichs-Cfit" in Molpro), using Molpro and an in-house SCF code (described in section 4.4.1). The energies are shown with all digits output by the programs (Molpro provided more figures than the in-house code). Absolute differences between the energies are quoted to the nearest order of magnitude.

	Energy / hartree		
Molecule	Molpro	In-house	Abs. Diff.
H_2O	-67.324844380247	-67.324844381	10^{-9}
Glycine	-248.445085042386	-248.445085046	10^{-9}
Naphthalene	-329.497952585570	-329.497952586	10^{-10}

differ by only 10^{-10} to 10^{-9} hartrees, corresponding to agreement in all but the last digit of the energy printed by the in-house code. This constitutes very close agreement, given the potential differences in program components other than the integral evaluation routines (e.g. in the definition of fundamental constants, or the implementation of SCF iterations). These results demonstrate that it is possible to create a complete, accurate, electronic structure program using integral evaluation routines generated by Intception.

As has been already mentioned, the difference in the work done by the two programs in order to evaluate the Hartree energy prohibited detailed comparisons of execution time. Nevertheless, it is worth considering the general timescales for the SCF calculations. The in-house code took less than 1 s, and just over 3 s to complete screened, density-fitted SCF Hartree energy calculations on glycine and naphthalene, using cc-pVDZ basis sets [51]. The corresponding calculations using Molpro (with additional work) took 6 s for glycine and 19 s for naphthalene. Though not directly comparable, these timings indicate that the in-house code, using generated integral routines, performs SCF calculations in a reasonable time frame (i.e. several seconds, rather than minutes).

4.4.5 Performance benchmarking

The results presented in the preceding sections demonstrate the numerical accuracy of integral evaluation code generated by Intception. In this section, we consider the computational performance of the generated code, in particular the routines generated to evaluate three-index Coulomb integrals.

As mentioned in section 4.2.3, the three-index Coulomb integrals are well-suited for benchmarking the performance of the generated code. This is because the evaluation of three-index integrals represents one of the more costly steps in density-fitted methods (section 2.4), and thus existing electronic structure codes possess highly optimized routines for evaluating three-index Coulomb integrals. Comparison of the performance of the generated code against these efficient routines should provide a good indication of the competitiveness of the generated code with optimized integral evaluation code.

Table 4.5 shows the time taken for Molpro and code generated by Intception to evaluate all shell-blocks of spherical contracted three-index Coulomb integrals for several molecule/basis set combinations. The Intception routines were called from within Molpro, as described in section 4.4.1. In these tests, an AO basis set was used for two indexes and a density fitting

Table 4.5: Comparison of the execution times of Molpro's built-in routines and generated code for the evaluation of three-index Coulomb integrals for benzene, glycine (zwitterion) and naphthalene (appendix I). The generated code used a VRR+HRR algorithm, with simplified VRR (Eq. 4.11, Ref. 198). The execution times are for the evaluation of all possible shell-blocks of spherical contracted three-index Coulomb integrals $(\mathrm{ab}|r_{12}^{-1}|C)$ (Eq. 4.19) with cc-pVXZ AO basis sets [51] and cc-pVXZ/MP2FIT auxiliary basis sets [185, 253] ($X = \mathrm{D}, \mathrm{T}$, with the same X for both AO and density fitting sets). Execution times are averaged over three independent test runs and are reported to two significant figures.

	Execution time / s		Ratio
	Molpro	Intception	$t_{\mathrm{Int}}/t_{\mathrm{Mol}}$
Benzene			
D	0.26	0.51	1.9
T	1.1	3.5	3.3
Glycine			
D	0.15	0.29	1.9
T	0.62	2.0	3.2
Naphthalene			
D	1.0	2.0	1.9
T	4.2	13	3.2

basis set was used for the other index, to better represent the typical use of the three-index Coulomb integrals in density-fitted methods (section 2.4). The three-index Coulomb integrals in question have the form

$$(\mathrm{ab}|r_{12}^{-1}|C) = \int \mathrm{d}\mathbf{r}_1 \mathrm{d}\mathbf{r}_2 \phi_a(\mathbf{r}_1)\phi_b(\mathbf{r}_1) r_{12}^{-1} \phi_C(\mathbf{r}_2), \tag{4.19}$$

where |a), |b) are contracted AO basis functions and $|C)$ is a density fitting basis function. The generated code used the VRR+HRR algorithm, with simplified VRR (Eq. 4.11, based on Ref. 198) to evaluate the integrals, as this was the fastest implementation available (see Figs. 4.12 and 4.13).

For all molecule/basis set combinations tested, Molpro's built-in routines for evaluating three-index Coulomb integrals performed better than the generated code. The generated code was typically two times slower for the DZ AO and density fitting basis sets and three times slower for the TZ sets—the generated code for three-index Coulomb integrals is clearly slower than Molpro's corresponding routines. Nevertheless, the fact that the generated code is able to evaluate all shell-blocks of the Coulomb three-index integrals in times even approaching those of a production code is impressive, given the generality and flexibility of the code generator.

An interesting facet of the results presented in Table 4.5 is that the ratio between the execution times for the generated code and Molpro's implementation are larger for the TZ basis sets. This does not appear to be related to the size of the basis set, since all the $t_{\mathrm{Int}}/t_{\mathrm{Mol}}$ ratios for sets with the same cardinal number, X, are essentially the same, even though they differ substantially in the number of basis functions. For example, the cc-pVDZ AO set and cc-pVDZ/MP2FIT density fitting sets for glycine (zwitterion) contain 95 and 350 functions, respectively, while the same basis sets for naphthalene contain 180 and 672 functions. It seems likely therefore, that the poorer relative performance of the generated code for the TZ sets is related to maximum angular momentum, ℓ_{max}, in the basis sets. The cc-pVDZ and cc-pVDZ/MP2FIT sets for the

first-row atoms contain functions up to $\ell = 2$ and $\ell = 3$, respectively, while the corresponding cc-pVTZ and cc-pVTZ/MP2FIT sets contain functions up to $\ell = 3$ and $\ell = 4$ [51, 185, 253].

To investigate the performance of the generated code in more detail, and in isolation from the larger infrastructure of an electronic structure package, a benchmarking suite was written (described in section 4.4.1). As mentioned in section 4.3.4, the structure of the source code output by Intception, where algorithm components from the integral evaluation framework (section 4.2.2) correspond to mostly separate functions, lends itself to detailed benchmarking. This structure allows the total cost of evaluating integrals to be broken down into contributions from individual algorithm components.

Figs. 4.12 and 4.13 show the time taken to execute different algorithm components for six variants of generated code for the evaluation of spherical, contracted three-index Coulomb integrals:

A: VRR-only using general VRR.

B: VRR+HRR using general VRR.

C: VRR+HRR using general VRR with index switch.

D: VRR-only using simplified VRR.

E: VRR+HRR using simplified VRR

F: VRR+HRR using simplified VRR with index switch.

The base and VRR expressions used are described in section 4.2.3. The VRR expression used to increment angular momentum in $|\mathbf{c})$ is either "general" (Eq. 4.10), or "simplified" (Eq. 4.11, Ref. 198).

Variants C and F differ from B and E in how they are called by the benchmarking suite, rather than the algorithm used for integral evaluation. In these "index switch" variants (C and F), a conditional block is used to call the main routine in a way which ensures that the HRR step shifts angular momentum from an integral index with higher angular momentum, to an index with lower final angular momentum. For example, if a shell-block of integrals with $\ell_a = 0$ and $\ell_b = 2$ is requested, the routine is called with indexes permuted, such that angular momentum is shifted from index $|\mathbf{b})$ to index $|\mathbf{a})$, i.e. the indexes are switched from the normal ordering. In the B and E variants, no conditional block is present, so the HRR step shifts angular momentum from $|\mathbf{a})$ to $|\mathbf{b})$ regardless of the angular momentum requested in each index.

In the benchmarking suite, the execution time was measured for the evaluation all shell-blocks of three-index Coulomb integrals by the generated main function (calls to work_array_size were not timed). To break this down into algorithm components, the generated source code was modified such that only the component of interest was executed—this involved recompiling the source code with the other components removed.[14] Thus, the execution time of each algorithm component was measured using an independent run of the benchmarking suite where the other algorithm components were disabled. This testing scheme has the following caveats:

[14] All the data comparing the cost of different algorithm components (Figs. 4.12, 4.13, 4.14 and 4.15) was collected from the benchmarking suite when it had been compiled using GCC (gcc, version 4.9.2) [244] and using optimization level -O1.

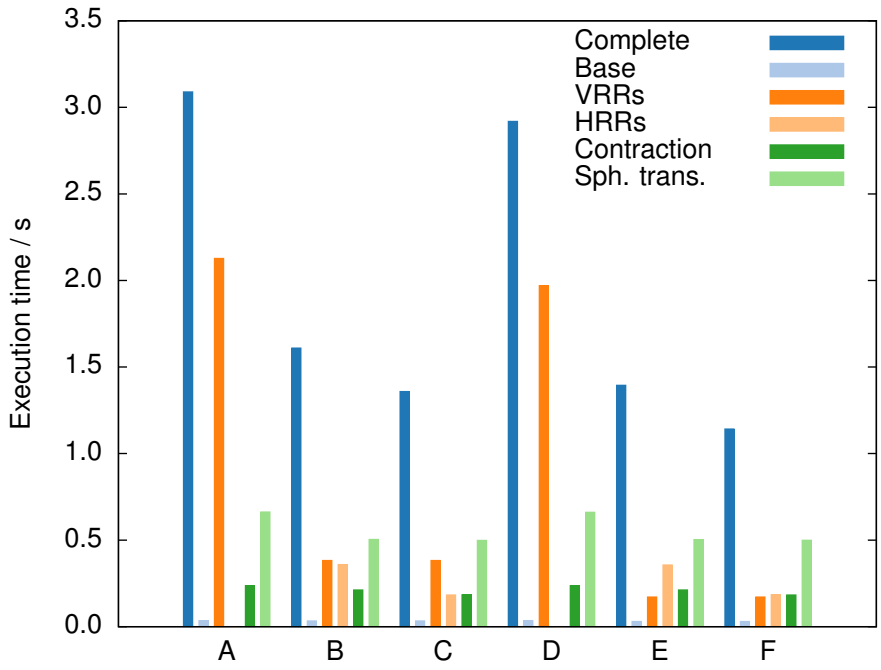

Figure 4.12: Execution time for evaluating all shell-blocks of three-index Coulomb integrals for H_2O (appendix I) using code generated for different algorithm variants (labelled A–F, see p. 152). The AO and density fitting basis sets used for all atoms were the cc-pV5Z [51, 52, 254] and cc-pV5Z/MP2FIT [185] sets, respectively. The execution time for the "Complete" code, featuring all algorithm components is shown alongside execution times for truncated codes featuring only a single component of the algorithm. The timing data is averaged over three independent test runs.

- All times measured contain a small overhead associated with the infrastructure needed to call the generated integral evaluation routines.
- The different algorithm components involve small amounts of shared infrastructure in the main function (e.g. code for declaring and initializing variables, loops, and array indexing), which also contribute to the times reported.
- Compiler optimization may be applied unequally when the code is compiled with different algorithm components active.

The data should therefore be considered as a broad illustration of the relative cost of algorithm components, sufficient to demonstrate general trends but not make fine comparisons.

The results presented in Figs. 4.12 (H_2O, cc-pV5Z and cc-pV5Z/MP2FIT basis sets) and 4.13 (benzene, cc-pVTZ and cc-pVTZ/MP2FIT basis sets) exhibit some broad trends:

- VRR+HRR algorithms are faster than VRR-only algorithms, as would be expected (see sections 2.3.3 and 2.3.5).
- Algorithms using the simplified VRR (Eq. 4.11) are faster than algorithms using the general VRR (Eq. 4.10) to increment angular momentum in $|\mathbf{c})$.

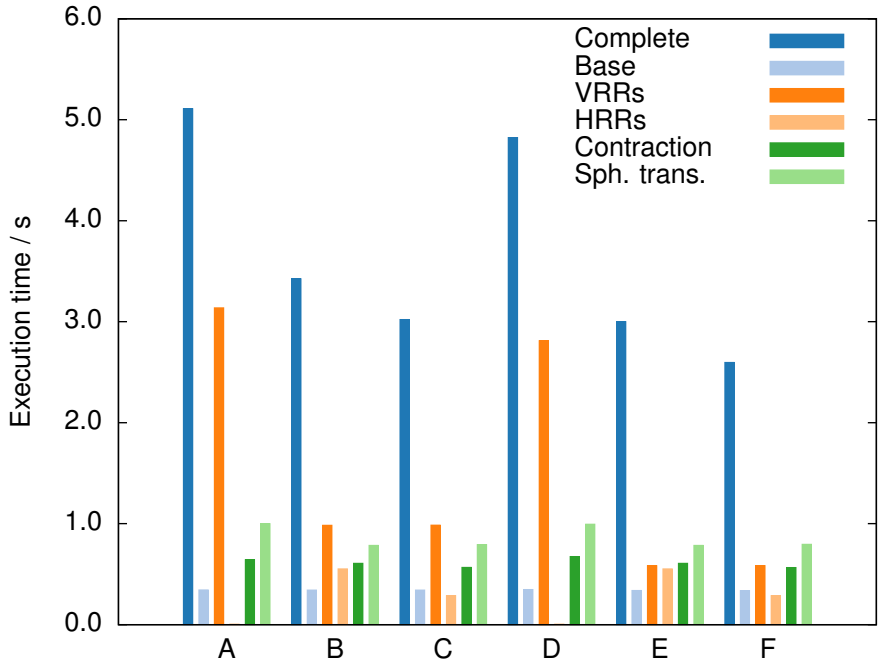

Figure 4.13: Execution time for evaluating all shell-blocks of three-index Coulomb integrals for benzene (appendix I) using code generated for different algorithm variants (labelled A–F, see p. 152), as described for Fig. 4.12, though with cc-pVTZ [51] and cc-pVTZ/MP2FIT [185] AO and density fitting basis sets.

For the VRR-only algorithms, the VRRs dominate the cost of integral evaluation, while for the VRR+HRR algorithms, the work is shared between the VRRs and HRR. The sum of the time taken for both the VRRs and HRRs in the VRR+HRR case is clearly less than the time taken to evaluate all VRR expressions in the VRR-only case, demonstrating the benefit of employing a HRR for this integral type. For the VRR+HRR algorithms, the use of the simplified VRR (Eq. 4.11) significantly decreases the cost of the VRRs, indicating the benefit of employing the simplified VRR over the general expression (Eq. 4.10). In contrast, the cost of evaluating the base function and of contraction remain fairly constant for all algorithm variants.

As would be expected, the switching of indexes to ensure that the HRR always shifts angular momentum from a higher angular momentum index to a lower angular momentum index reduces the cost of the HRR step. In Figs. 4.12 and 4.13, the cost of the HRR step is essentially halved when index switching is done (variants C and F).

Interestingly, Figs. 4.12 and 4.13 indicate that the spherical transformation step is less costly for the VRR+HRR algorithms than the VRR-only algorithms. This is likely because in the VRR+HRR algorithm, the spherical transformation of density fitting index $|C)$ occurs before the HRR step, so the transformation involves a matrix with $n_{\text{cart}}(\ell_a + \ell_b)$ Cartesian components of indexes |a) and |b), rather than $n_{\text{cart}}(\ell_a) \times n_{\text{cart}}(\ell_b)$ (the former expression has a lower polynomial order in ℓ_a and ℓ_b—see Eq. 2.134). Despite the lower cost of spherical transformation in the VRR+HRR algorithms, the spherical transformation step is the most costly single algorithm component for simplified VRR+HRR algorithms (variants E and F) in both molecule/basis set combinations. This is particularly marked in the case of H_2O with

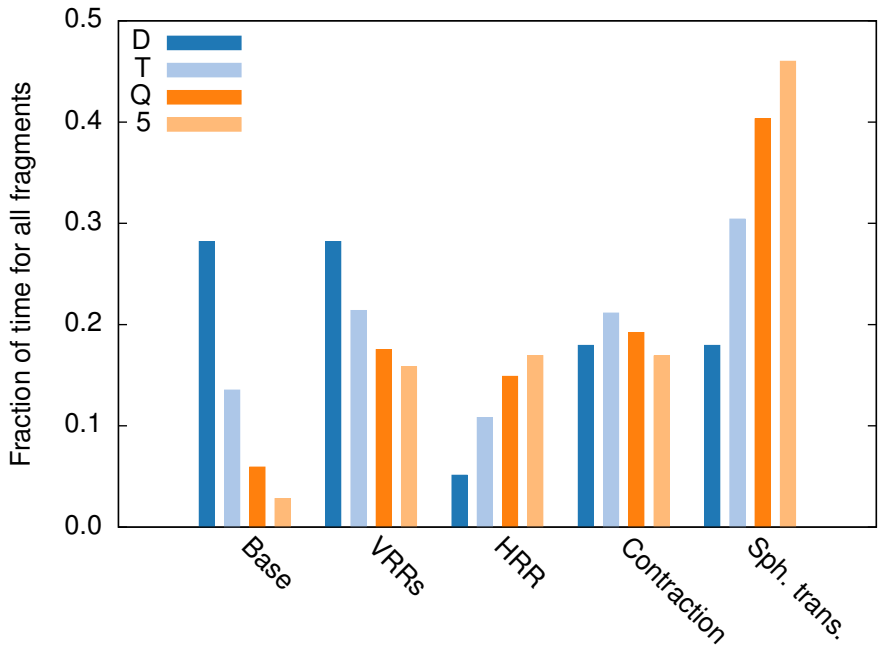

Figure 4.14: Relative execution time for evaluating all shell-blocks of three-index Coulomb integrals for H_2O (appendix I) with increasing basis set size and angular momentum, using code generated for algorithm variant F (VRR+HRR, with simplified VRR and index switch, see p. 152). The execution time for each algorithm component is plotted as a fraction of the summed times of all algorithm components for cc-pVXZ [51, 52, 254] and cc-pVXZ/MP2FIT [185, 253] AO and density fitting basis sets ($X = \mathrm{D}, \mathrm{T}, \mathrm{Q}, 5$). The timing data was averaged over three independent test runs.

cc-pV5Z and cc-pV5Z/MP2FIT AO and density fitting basis sets, where, for algorithm variant F, the spherical transformation step takes more than double the time taken by the VRR or HRR steps.

The results presented in Figs. 4.14 and 4.15 deal only with the fastest algorithm variant (F, see p. 152) and further emphasise the high cost of the spherical transformation step observed in Figs. 4.12 and 4.13. These charts show the time taken to execute code featuring each of the algorithm components as a fraction of the summed execution time of all algorithm components,[15] with increasing basis set size (and angular momentum). As the cardinal number, X, of the cc-pVXZ AO and cc-pVXZ/MP2FIT density fitting basis sets is increased, the fraction of the total execution time taken by spherical transformation grows rapidly and once $X = \mathrm{Q}, 5$, the spherical transformation step clearly dominates the cost of integral evaluation.

The rapid increase in the cost of the spherical transformation step demonstrated in Figs. 4.14 and 4.15 might help explain the observed decrease in the relative performance of the generated code with respect to Molpro's built-in routines on moving from DZ to TZ basis sets, as shown in Table 4.5. As discussed earlier, the relative performance of the two codes does not appear to change significantly when the size of the basis set is increased, but $\ell_{\max}$ remains constant, but

[15] The summed time includes the time taken to copy the result from the work array to the output array, though this a very small contribution (1 to 4 % of the total time).

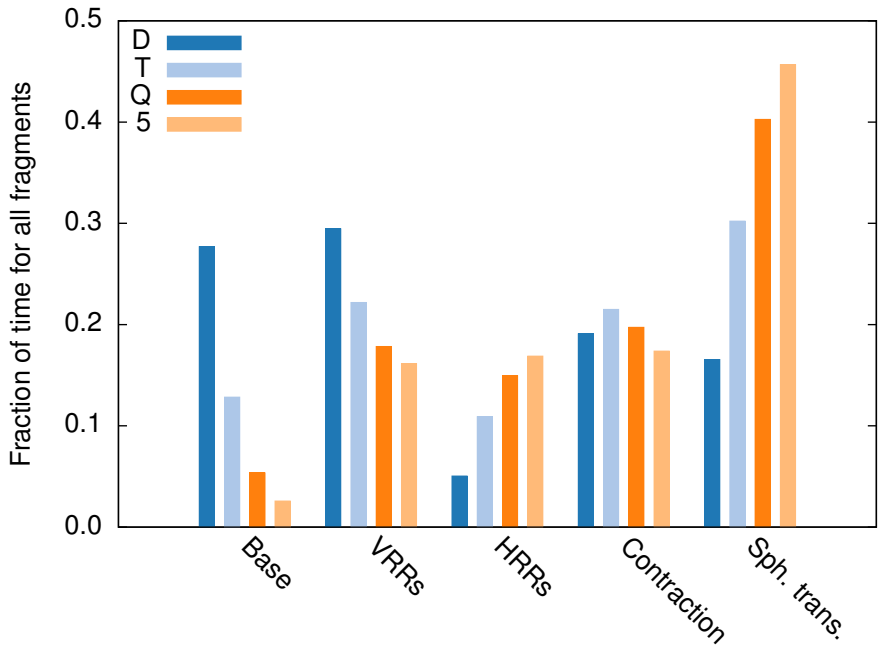

Figure 4.15: Relative execution time for evaluating all shell-blocks of three-index Coulomb integrals for benzene (appendix I) with increasing basis set size and angular momentum, as described for Fig. 4.14

does change when $\ell_{\max}$ is increased in the basis sets, i.e. when moving from $X = \mathrm{D}$ to $X = \mathrm{T}$.[16] It seems likely, based on the data in Figs. 4.14 and 4.15, that the cause of the decreasing relative performance of the generated code with respect to Molpro on moving from DZ to TZ basis sets is a consequence of the rapidly increasing cost of the spherical transformation step.

Taken together, the results in Table 4.5 and Figs. 4.12 to 4.15 suggest that the optimization of the spherical transformation code should be the immediate focus for future development work on Intception. At present, spherical transformation is implemented as a dense matrix multiplication of spherical transformation coefficients with primitive integrals (Eq. 4.6). Matrices of spherical transformation coefficients are typically sparse, with many elements equal to zero—it is likely that significant cost savings could be made by modifying Intception to generate spherical transformation routines that take advantage of this sparsity.

An important factor affecting the performance of numerical source code is the implementation of the compiler used, and level of compiler optimization applied. Source code written in a high-level programming language, such as C or Fortran is highly abstracted compared to its compiled form, and the way in which this is converted into machine code can significantly affect the behaviour of the resulting executable. Using compiler optimization, machine code can be compiled in a way which maximizes the speed of execution for a particular target environment.[17]

Tables 4.6 and 4.7, contain execution times for generated source code compiled using several different C compilers and optimization levels, for increasing basis set size. The execution times,

[16] For each increment in X, $\ell_{\max}$ increases by one for both the cc-pVXZ [51, 52, 254] and cc-pVXZ/MP2FIT [185, 253] basis set families.

[17] Details of compiler implementation and optimization can be found in many computer science textbooks, for example, Refs. 255 and 256. A brief introduction is given in Ref. 122 (p.188).

Table 4.6: Execution times for the evaluation of all shell-blocks of three-index Coulomb integrals for glycine (zwitterion, see appendix I), with increasing basis set size (and angular momentum). Algorithm variant F was used (VRR+HRR, with simplified VRR and index switch, see p. 152) to evaluate the integrals. Source code was compiled with a selection of compilers (see section 4.4.1 for details) and optimization levels. The cc-pVXZ [51, 52, 254] and cc-pVXZ/MP2FIT [185, 253] AO and density fitting basis set families were used for all atoms ($X = \mathrm{D}, \mathrm{T}, \mathrm{Q}, 5$). The ratio of the execution times for code compiled with the -O1, -O2, and -O3 optimization levels to the execution time for unoptimized (-O0) code, t/t_{O0}, is also shown. The execution times are averages over three independent test runs, reported to two significant figures.

	D		T		Q		5	
	t /s	t/t_{O0}	t /s	t/t_{O0}	t /s	t/t_{O0}	t /s	t/t_{O0}
gcc -O0	0.65	-	5.0	-	37	-	210	-
gcc -O1	0.24	0.36	1.5	0.30	11	0.28	61	0.28
gcc -O2	0.19	0.30	1.2	0.25	8.5	0.23	47	0.22
gcc -O3	0.19	0.30	1.2	0.25	8.5	0.23	47	0.22
clang -O0	0.60	-	4.6	-	35	-	200	-
clang -O1	0.23	0.39	1.5	0.32	10	0.28	56	0.28
clang -O2	0.24	0.41	1.5	0.33	9.8	0.28	51	0.25
clang -O3	0.24	0.41	1.5	0.33	9.8	0.28	51	0.25
icc -O0	0.63	-	4.9	-	37	-	220	-
icc -O1	0.21	0.34	1.4	0.28	9.3	0.25	52	0.24
icc -O2	0.19	0.31	1.3	0.26	9.0	0.24	50	0.23
icc -O3	0.22	0.35	1.5	0.30	10	0.27	54	0.25

as before, are for the evaluation of all shell-blocks of three-index Coulomb integrals, using the fastest variant of the generated source code: VRR+HRR with simplified VRR and index switch (F, p. 152). The GCC [244], Clang [245] and Intel C compilers [246] (see section 4.4.1 for details) were used to compile both the benchmarking suite and generated integral evaluation code at various levels of compiler optimization.

All three compilers offer multiple optimizations, consolidated into broad optimization "levels": -O0, -O1, -O2 and -O3, where -O0 indicates no optimization, and -O1 to -O3 indicate increasing levels of optimization. These broad levels correspond to collections of individual optimizations, such as automatic loop unrolling and function inlining, and are not equivalent for the various compilers (indeed the individual optimizations included at each level may even change between versions of the same compiler). However, it is worth comparing these broad optimization levels, since these are the options users of Intception will be likely to utilize when compiling this code.

For both glycine and benzene, across all compilers and basis set sizes, the effect of compiler optimization is very similar. Moving from no optimization (-O0) to the first broad optimization level (-O1) yields a significant speed-up, with the code compiled using -O1 completing execution in 20 to 40 % of the time taken to execute the code compiled with -O0. For all compilers, moving to higher optimization levels yields little or no improvement (and may even result in a very slight decrease in performance relative to lower optimization levels). Additionally, the impact of compiler optimization appears to be greater for larger basis sets, indicating that perhaps portions of code which are more effectively optimized take a larger proportion of the overall execution time for these larger basis sets.

The difference in execution times between the compilers is relatively small compared to the

Table 4.7: Execution times for the evaluation of all shell-blocks of three-index Coulomb integrals for benzene (appendix I), with increasing basis set size (and angular momentum), as described for Table 4.6.

	D		T		Q		5	
	t /s	t/t_{O0}	t /s	t/t_{O0}	t /s	t/t_{O0}	t /s	t/t_{O0}
gcc -O0	1.1	-	8.6	-	65	-	370	-
gcc -O1	0.41	0.36	2.6	0.30	18	0.28	100	0.28
gcc -O2	0.33	0.30	2.2	0.25	15	0.23	82	0.22
gcc -O3	0.33	0.30	2.2	0.25	15	0.23	82	0.22
clang -O0	1.0	-	8.0	-	61	-	350	-
clang -O1	0.40	0.39	2.5	0.31	17	0.28	97	0.28
clang -O2	0.42	0.41	2.6	0.32	17	0.28	88	0.25
clang -O3	0.42	0.41	2.6	0.33	17	0.28	88	0.25
icc -O0	1.1	-	8.6	-	65	-	370	-
icc -O1	0.37	0.34	2.4	0.28	16	0.25	89	0.24
icc -O2	0.33	0.31	2.3	0.26	16	0.24	86	0.23
icc -O3	0.38	0.35	2.6	0.30	17	0.27	94	0.25

total execution time. For the two molecules considered, GCC appears to offer a very slight improvement over the other compilers when optimization is applied, though it is not clear that this would translate to other hardware and software environments, chemical systems or integral types. Essentially, this data suggests that, for code generated by the current version of Intception, compiler optimization at any level above -O0 offers significant benefits, and that there are no significant differences between the three compilers tested.

4.5 Conclusions

In this chapter, the design, implementation and testing of Intception, a code generator for molecular integral evaluation routines, has been described. The original motivation for this work was the difficulty in implementing integral evaluation routines for new integral types, such as the three-electron integral types required for the work presented in chapter 3. In this respect, Intception has fulfilled its purpose, being capable of generating numerically accurate source code for the evaluation of these three-electron integral types which is significantly more efficient than the hand-coded implementation presented in chapter 3.

Throughout this work, the emphasis has been on the generality and flexibility of code generation. As demonstrated by the results presented in sections 4.4.2 and 4.4.3, Intception is capable of generating numerically accurate code for a wide range of integral types and evaluation algorithms. This flexibility can be largely attributed to the use of the general integral evaluation framework described in section 4.2.2 as the basis for code generation. Using such a framework, the problem of creating algorithms for integral evaluation is reduced to a mechanical process in which the framework is customized for each integral type. The implementation of this flexible approach is greatly assisted by the use of a DSL (section 4.3.2), both as input and as an internal representation of domain-specific entities.

The DSL developed for Intception was deliberately designed to be simple to learn and use, so that researchers engaged in method development could rapidly define and generate source code for the integral types they need. In addition, the DSL and code generator were designed with extensibility in mind, with the aim of allowing future developers to modify and extend the

existing code. In both cases, this was facilitated by the use of the Python language [234], which provides powerful object-oriented programming features, enabling the creation of an embedded DSL and the easy extension of existing code (for example using class inheritance, as shown in Fig. 4.5).

The suitability of code generated by Intception for use in electronic structure calculations is demonstrated in the results presented in section 4.4.4. Using generated code for the full set of standard integral types required in a density-fitted SCF calculation, the Hartree energy of several molecules was calculated to a high accuracy, and within a reasonable time frame. The ability of Intception to generate code for the full complement of integral types needed in a SCF calculation means that a basic electronic structure code can be built around a library of generated integral routines.

Although Intception could be used to generate code for all the integrals used in an electronic structure calculation, the resulting implementation would likely be too slow for large scale calculations. As demonstrated in section 4.4.5, there is room for improvement in the computational performance of the code generated by the current version of Intception. For the implementation of new integral types for use in new electronic structure methods, the computational performance is likely to be a secondary concern, outweighed by the vastly reduced time and effort required to implement these integrals (as in the case of the three-electron J-F and F-F integrals, for which generated code actually performs better than the hand-coded alternative, section 4.4.3). In contrast, for standard integral types, like the three-index Coulomb integrals, there exist highly optimized implementations and the performance of generated code is likely to be of greater concern. In this case, the somewhat slower performance observed for generated code compared to existing optimized code (Table 4.5) is problematic. Of particular concern is the apparent degradation in the relative performance of the generated code relative to Molpro's optimized three-index Coulomb integral code with increasing angular momentum in the basis set, as shown in Table 4.5.

Given the flexibility and generality of Intception, the fact that generated code for the evaluation of the three-index Coulomb integrals executes on even the same time scale as an optimized code is, perhaps, impressive. Nevertheless, for Intception to usefully generate code for use in full electronic structure calculations where computational performance is a significant concern, the relatively slow performance of the generated code needs addressing. To this end, a detailed analysis of the cost of various algorithm components in evaluating the three-index Coulomb integrals was performed (section 4.4.5). This strongly indicated that the observed decrease in relative performance with respect to Molpro, for increasing angular momentum in the AO and density fitting basis sets (Table 4.5), was a consequence of the inefficient implementation of the spherical transformation step (Figs. 4.14 and 4.15). As suggested in section 4.4.5, modifying Intception such that the code generated for spherical transformation takes advantage of the sparsity of matrices of spherical transformation coefficients may resolve this issue.

A short investigation into techniques for improving the computational performance of the code generated by Intception was performed by Tom Rumsey, an MEng computer science student [257]. The results obtained in this investigation suggested that restructuring the generated source code, for example by moving conditional expressions outside of unrolled loops (see List-

ing. 4.4) and inlining functions, could yield substantial speed-ups. A particularly interesting avenue for performance improvement was the optimization of the DSL expression trees (Fig. 4.4) used to represent recurrence relation and base expressions. Rumsey described a method by which certain operations and functions that translate into relatively inefficient C code could be automatically replaced by more efficient equivalents. A simple example of this is the replacement of `exp(a)*exp(b)` with `exp(a+b)`, which swaps two expensive calls to the `exp` function for a single call to `exp` and an addition.

The optimizations proposed by Rumsey had not been implemented in the main Intception codebase at the time of writing, and the results reported in section 4.4 therefore do not reflect these optimizations. It is difficult to predict the degree to which the performance of generated code would be improved by the proposed optimizations, since the results acquired by Rumsey in his short investigation were somewhat preliminary. Nevertheless, the fact that such code optimizations are possible indicates that there is room for performance improvement without changing the overall infrastructure or theoretical basis for Intception.

One important route to performance optimization which was considered in this work was the application of compiler optimization. The results in Tables 4.6 and Table 4.7 indicate that code generated by the present version of Intception is significantly accelerated by the application of compiler optimization. The results also demonstrate that the performance is not substantially affected by the compiler used—this is a useful property for code that is intended to be broadly applicable across hardware and software environments.

The results presented in this chapter demonstrate that the current version of Intception is a useful piece of software, capable of generating numerically accurate source code for a range of integral types. It is also a work in progress. At present, the software is constrained in ways which prevent the implementation of certain integral types (listed in section 4.3.1) and produces code which is somewhat less efficient than existing optimized implementations. The removal of these constraints and improvement of the computational performance of generated code are clear short-term goals for future work on Intception.

In addition to these short-term goals, there are some interesting long-term possibilities for future development. One such possibility is the adaptation of Intception to automatically generate source code which executes in parallel. The separation of the abstract representation of integral definitions from the source-code-specific implementation of evaluation algorithms for these integrals in Intception should facilitate this adaptation, by allowing developers to construct a parallel version of the code generator using the existing abstract representations. A particularly interesting avenue for this research would be the modification of Intception to produce GPU-specific source code. This could allow the implementation of multiple integral types for evaluation on GPUs, in contrast with the GPU-specific code generator projects referred to in section 4.1, which focus on ERI evaluation [112, 113, 115, 204, 228].

Another interesting long-term possibility is the modification of the code generator to produce source code which evaluates integrals to arbitrary precision. At present, the accuracy of integral evaluation by generated code is restricted by the use of the the double precision floating point data format [120] to store integrals and intermediate quantities (see the discussion of accuracy on p. 144). To overcome this limitation, the code generator could be modified to output source

code that performs arithmetic using arbitrary-precision data types, such as those provided by the GNU MPFR Library [258, 259]. In this case, an arbitrary-precision implementation of the Boys function would be necessary (rather than the approximate implementation currently used, described in appendix D). This could perhaps be achieved by expressing the Boys function in terms of the error function (see Ref. 10 (p.369)) and using an arbitary precision implementation of this (as available in GNU MPFR). The ability to generate arbitrary-precision integral evaluation code for a variety of integral types could be very useful for assessing the accuracy of other integral codes and checking the correctness of the fixed-precision code currently generated by Intception.

It may also be worth developing the code generation capabilities of Intception further to extend the range of code it can generate. For example, it should be possible, based on the abstract representation of each integral type, to generate self-test routines, which could be run to check that the generated code behaves as expected. These self-test routines could check that the generated code adheres to known properties of the integral, such as translational invariance (Eq. 2.153), or invariance to permutation of integral indexes. It is also possible that Intception could be extended to generate code which can evaluate integrals using multiple algorithm variants and dynamically select the most efficient algorithm at run-time, in analogy to the PRISM algorithm [98, 108–110]. Additionally, it may be useful to extend the current framework to automatically implement code which evaluates integral gradients, for use in geometry optimization.

Clearly, there are many possible directions for the future development of Intception. It is hoped that the current version of the code and the approach outlined in this chapter will provide a starting point for the realization of some of these possibilities.

CHAPTER 5

CONCLUDING REMARKS

The body of work described in the chapters 3 and 4 consists of two independent threads, connected by the common theme of molecular integral evaluation. In this final, brief, chapter we will consider the key results presented in each chapter, with an emphasis on the connections between the two threads and the broader context of this research. For detailed conclusions and suggestions for future work specific to each thread see the conclusion sections of the corresponding chapters (sections 3.5 and 4.5).

In chapter 3, the development of a new variant of MP2-F12 theory, DF3-MP2-F12, was described, in which three-electron integrals are approximated by density fitting, rather than resolutions of the identity (RIs). This work was motivated by the problematically high angular momentum functions required in the RI basis sets used to approximate the many-electron integrals in standard MP2-F12 theory. Density fitting appeared to offer an alternative route to the approximation of the many-electron integrals in which the maximum angular momentum required in the auxiliary basis set was lower. The key conclusion of this work was that density fitting is a viable alternative to RIs in MP2-F12 theory, demonstrating the expected lower angular-momentum requirement, and rapid convergence of the MP2-F12 energy with respect to basis set size (see also Ref. 150). In the broader context of the development of explicitly correlated methods, this work represents a proof-of-concept for the use of density fitting as an alternative method of approximating many-electron integrals. It is hoped that the advantages of density fitting many-electron integrals demonstrated in chapter 3 will encourage the development of other explicitly correlated methods based on this approach.

In order to assess the use of density fitting to approximate three-electron integrals in MP2-F12 theory, it was necessary to write code to evaluate certain three-electron integral types (see Table 3.1). Even though the complexity of this implementation was reduced by leveraging existing routines in Molpro [151, 152] for evaluating two-electron integral types and applying the GPT (section 3.3.2, appendix B), this proved to be extremely time-consuming. In addition, because an inefficient method for evaluating these integrals was used, to avoid the complexity of implementing full Obara-Saika-type recurrence relations (section 2.3.3, appendix C), the resulting implementation of DF3-MP2-F12 was very slow. This inefficiency constrained the assessment of the method, limiting tests to simple small molecules and atoms (section 3.4). Had relatively efficient code for evaluating the three-electron integral types been available at the start of the project, the implementation of DF3-MP2-F12 theory would have been far easier, and it would have likely been possible to perform a more in-depth assessment of the method.

The difficulties encountered in implementing code for the evaluation of three-electron integrals inspired the work presented in chapter 4. In this chapter, the design, implementation and application of "Intception", a molecular integral code generator, is described. The results reported in section 4.4 demonstrate that Intception fulfils its original purpose, and is able to flexibly generate numerically accurate source code for a variety of integral types. These include the troublesome three-electron integrals of chapter 3, for which the generated code is significantly faster than the previous hand-coded implementation (Table 4.3). The results also indicate that there is substantial room for improvement. In particular, although the generated code executes on a reasonable time scale, it is somewhat slower than existing optimized implementations. Possible routes to reducing this discrepancy were revealed by detailed benchmarking of the generated code for the three-index Coulomb integrals (section 4.4.5).

It is hoped that Intception will continue to be used, with further development expanding its capabilities and improving the performance of generated code. Several possible avenues for future development were suggested in the section 4.5. At present, code generated by Intception forms part of a new electronic structure code under active development (as used to produce the results in section 4.4.4). This is a promising combination, and it is hoped that further development of Intception will be driven by the needs of the new electronic structure package.

As stated in the introduction to chapter 4, the idea of using code generation to implement code for evaluating molecular integrals is not a new one, and Intception is one of several projects with similar goals, such as "Libint" [230] and "Libcint" [229]. The question arises: why create another molecular integral code generator? One reason for this is reproduction of results, a key element of the scientific method. The existence of multiple software packages that produce the same results, but in different ways, facilitates the reproduction of published results and the identification of errors in the individual programs. This kind of diverse software ecosystem exists for electronic structure software, where there are many different packages under development by different research groups.

In the case of Intception, however, the main motivation for developing a new molecular integral code generator was the freedom available in designing a complete software package. This meant that the design of Intception was not constrained by the choices made in other code generation projects, and could be focused upon the primary goal of flexible code generation for new integral types. The general integral evaluation framework and extensible Python-based DSL used in Intception are consequences of this focus on flexibility, and would likely have been difficult to recreate in the context of an established code generation project.

In summary, both of the research projects presented in this thesis have produced interesting results relating to the implementation of molecular integrals in electronic structure theory. In both cases, the results are a proof-of-concept, and further work will be necessary to produce theory and software which is suitable for more widespread use. It is therefore hoped that this thesis will serve as a useful resource for future researchers, who seek to develop related ideas, or extend the work presented in the preceding chapters.

APPENDIX A

THE SOLID HARMONICS

For convenience, the definition of the real-valued solid harmonics $S_{\ell m}$ is reproduced here. This is adapted from the detailed description presented in Ref. 10 (ch.6).

The complex solid harmonics are derived from the spherical harmonics, $Y_{\ell m}$,

$$\mathcal{Y}_{\ell m}(r,\theta,\phi) = r^{\ell} Y_{\ell m}(\theta,\varphi) \tag{A.1}$$

and may be expressed in Cartesian coordinates

$$\mathcal{Y}_{\ell m}(\mathbf{r}) = N_{\ell m}(r_x + \mathrm{sgn}(m)\,\mathrm{i}\,r_y)^{|m|} \sum_{t=0}^{(\ell-|m|)/2} C_t^{\ell|m|}(r_x^2 + r_y^2)^t r_z^{\ell-2t-|m|} \tag{A.2}$$

with

$$C_t^{\ell m} = \left(-\frac{1}{4}\right)^t \binom{\ell - t}{|m| + t}\binom{\ell}{t} \tag{A.3}$$

and

$$N_{\ell m} = (-1)^{(m+|m|)/2} \frac{1}{2^{|m|}\ell!}\left(\frac{2\ell+1}{4\pi}(\ell+|m|)!(\ell-|m|)!\right)^{1/2}. \tag{A.4}$$

The real-valued solid harmonics, as used in section 2.3.1 are:

$$S_{\ell 0}(\mathbf{r}) = \left(\frac{4\pi}{2\ell+1}\right)^{1/2} \mathcal{Y}_{\ell 0}(\mathbf{r}), \qquad m = 0 \tag{A.5}$$

$$S_{\ell m}(\mathbf{r}) = (-1)^m \left(\frac{8\pi}{2\ell+1}\right)^{1/2} \mathrm{Re}\,\mathcal{Y}_{\ell m}(\mathbf{r}), \qquad m > 0 \tag{A.6}$$

$$S_{\ell,m}(\mathbf{r}) = (-1)^{-m} \left(\frac{8\pi}{2\ell+1}\right)^{1/2} \mathrm{Im}\,\mathcal{Y}_{\ell,-m}(\mathbf{r}). \qquad m < 0 \tag{A.7}$$

In section 2.3.1, the relationship between the solid harmonics and Cartesian polynomials,

$$X_{\mathbf{a}}(\mathbf{r}) = r_x^{a_x} r_y^{a_y} r_z^{a_z}, \tag{A.8}$$

was given as

$$S_{\ell m}(\mathbf{r}) = N_{\ell m}^S \sum_{\mathbf{a}} C_{\mathbf{a}}^{\ell m} X_{\mathbf{a}}(\mathbf{r}), \tag{A.9}$$

where the summation runs over Cartesian components with $\ell = |\mathbf{a}|$. This form was used for simplicity and clarity, though it features terms in which $C_{\mathbf{a}}^{\ell m} = 0$.

The relationship derived in Ref. 10 (pp.214–215) features an explicit summation over non-zero contributions:

$$S_{\ell m}(\mathbf{r}) = N^S_{\ell m} \sum_{t,u,v} C^{\ell m}_{tuv} r_x^{2t+|m|-2(u+v)} r_y^{2(u+v)} r_z^{\ell-2t-|m|} \tag{A.10}$$

where,

$$\sum_{t,u,v} \equiv \sum_{t=0}^{\ell-|m|/2} \sum_{u=0}^{t} \sum_{v=v_m}^{(|m|/2-v_m)+v_m}$$

with

$$v_m = \begin{cases} 0 & m \geq 0, \\ 1/2 & m < 0. \end{cases} \tag{A.11}$$

The normalization and transformation coefficients (defined differently to those used Eq. A.9 and section 2.3.1) are:

$$C^{\ell m}_{tuv} = (-1)^{t+v-v_m} \left(\frac{1}{4}\right)^t \binom{\ell}{t} \binom{\ell-t}{|m|+t} \binom{t}{u} \binom{|m|}{2v}, \tag{A.12}$$

$$N^S_{\ell m} = \frac{1}{2^{|m|}\ell!} \left(\frac{2(\ell+|m|)!(\ell-|m|)!}{2^{\delta_{0m}}} \right)^{1/2}. \tag{A.13}$$

APPENDIX B

GAUSSIAN PRODUCT THEOREM

The product of two Gaussian functions on different centres can be expressed as a linear combination of Gaussians on a single centre [88], i.e.

$$g(\mathbf{r};\zeta_a,\mathbf{a},\mathbf{A})g(\mathbf{r};\zeta_b,\mathbf{b},\mathbf{B}) = \sum_{\mathbf{p}} T_{\mathbf{p}}^{\mathbf{ab}} g(\mathbf{r};\zeta,\mathbf{p},\mathbf{P}), \tag{B.1}$$

where $\zeta = \zeta_a + \zeta_b$, $\mathbf{p}$ runs from $(0,0,0)$ to $(a_x + b_x, a_y + b_y, a_z + b_z)$, and

$$\mathbf{P} = \frac{\zeta_a \mathbf{A} + \zeta_b \mathbf{B}}{\zeta}.$$

A full derivation of the form of $T_{\mathbf{p}}^{\mathbf{ab}}$ is presented in Ref 89. Since this is not widely available, the final result of the derivation is reproduced here:

$$T_{\mathbf{p}}^{\mathbf{ab}} = \exp(-\xi|\mathbf{A}-\mathbf{B}|^2) \prod_{i=x,y,z} f_{p_i}(a_i, b_i, P_i - A_i, P_i - B_i) \tag{B.2}$$

where

$$\xi = \frac{\zeta_a \zeta_b}{\zeta_a + \zeta_b}$$

and

$$f_{p_i}(a_i, b_i, P_i - A_i, P_i - B_i) = \sum_{j=0}^{p_i} \binom{a_i}{j} \binom{b_i}{p_i - j} (P_i - A_i)^{a_i - j} (P_i - B_i)^{b_i - p_i + j}. \tag{B.3}$$

APPENDIX C

RECURRENCE RELATIONS

C.1 Four-index ERIs

To complement the description in section 2.3.3, the recurrence relations and zero-angular-momentum expression derived for the four-index ERIs, $(\mathbf{ab}|r_{12}^{-1}|\mathbf{cd})$, in Ref. 1 are reproduced here.

The recurrence relation derived for the integral $(\mathbf{ab}|g_{12}^{-1}|\mathbf{cd})$ (see Eq. 2.168) is

$$\begin{aligned}((\mathbf{a}+\mathbf{1}_i)\mathbf{b}|g_{12}^{u^2}|\mathbf{cd}) &= \mathbf{PA}_i(\mathbf{ab}|g_{12}^{u^2}|\mathbf{cd}) + \mathbf{WP}_i\left(\frac{u^2}{\rho+u^2}\right)(\mathbf{ab}|g_{12}^{u^2}|\mathbf{cd}) \\ &+ \frac{a_i}{2\zeta}\left(1-\frac{\rho}{\zeta}\frac{u^2}{\rho+u^2}\right)((\mathbf{a}-\mathbf{1}_i)\mathbf{b}|g_{12}^{u^2}|\mathbf{cd}) \\ &+ \frac{b_i}{2\zeta}\left(1-\frac{\rho}{\zeta}\frac{u^2}{\rho+u^2}\right)(\mathbf{a}(\mathbf{b}-\mathbf{1}_i)|g_{12}^{u^2}|\mathbf{cd}) \\ &+ \frac{c_i}{2(\zeta+\eta)}\left(\frac{u^2}{\rho+u^2}\right)(\mathbf{ab}|g_{12}^{u^2}|(\mathbf{c}-\mathbf{1}_i)\mathbf{d}) \\ &+ \frac{d_i}{2(\zeta+\eta)}\left(\frac{u^2}{\rho+u^2}\right)(\mathbf{ab}|g_{12}^{u^2}|\mathbf{c}(\mathbf{d}-\mathbf{1}_i))\end{aligned} \tag{C.1}$$

with $\zeta = \zeta_a + \zeta_b$, $\eta = \zeta_c + \zeta_d$, and

$$\rho = \frac{\zeta\eta}{\zeta+\eta} \qquad \mathbf{P} = \frac{\zeta_a\mathbf{A}+\zeta_b\mathbf{B}}{\zeta_a+\zeta_b} \qquad \mathbf{Q} = \frac{\zeta_c\mathbf{C}+\zeta_d\mathbf{D}}{\zeta_c+\zeta_d} \qquad \mathbf{W} = \frac{\zeta\mathbf{P}+\eta\mathbf{Q}}{\zeta+\eta}.$$

Using the auxiliary integrals $(\mathbf{ab}|\mathbf{cd})^{(m)}$ (Eq. 2.169), the following recurrence relation can be derived:

$$\begin{aligned}((\mathbf{a}+\mathbf{1}_i)\mathbf{b}|\mathbf{cd})^{(m)} &= \mathbf{PA}_i(\mathbf{ab}|\mathbf{cd})^{(m)} + \mathbf{WP}_i(\mathbf{ab}|\mathbf{cd})^{(m+1)} \\ &+ \frac{a_i}{2\zeta}\left(((\mathbf{a}-\mathbf{1}_i)\mathbf{b}|\mathbf{cd})^{(m)} - \frac{\rho}{\zeta}((\mathbf{a}-\mathbf{1}_i)\mathbf{b}|\mathbf{cd})^{(m+1)}\right) \\ &+ \frac{b_i}{2\zeta}\left((\mathbf{a}(\mathbf{b}-\mathbf{1}_i)|\mathbf{cd})^{(m)} - \frac{\rho}{\zeta}(\mathbf{a}(\mathbf{b}-\mathbf{1}_i)|\mathbf{cd})^{(m+1)}\right) \\ &+ \frac{c_i}{2(\zeta+\eta)}(\mathbf{ab}|(\mathbf{c}-\mathbf{1}_i)\mathbf{d})^{(m+1)} + \frac{d_i}{2(\zeta+\eta)}(\mathbf{ab}|\mathbf{c}(\mathbf{d}-\mathbf{1}_i))^{(m+1)}.\end{aligned} \tag{C.2}$$

The zero-angular-momentum expression for the four-index auxiliary integrals is

$$(\mathbf{0}_A\mathbf{0}_B|\mathbf{0}_C\mathbf{0}_D)^{(m)} = 2\left(\frac{\rho}{\pi}\right)^{1/2}(\mathbf{0}_A|\mathbf{0}_B)(\mathbf{0}_C|\mathbf{0}_D)F_m(T) \tag{C.3}$$

where the $(\mathbf{0}_A|\mathbf{0}_B)$ and $(\mathbf{0}_C|\mathbf{0}_D)$ are two-index overlap integrals, as defined in Eq. 2.155, and $F_m(T)$ is the Boys function with argument $T = \rho|\mathbf{P}-\mathbf{Q}|^2$ (see section 2.3.4).

C.2 Two-index Coulomb integrals

The zero-angular-momentum expression and VRR for the two-index Coulomb integrals, $(\mathbf{a}|r_{12}^{-1}|\mathbf{b})$, used in chapter 4 were derived from the corresponding expressions for the four-index ERIs in Ref. 1 (reproduced in section C.1) by setting $\zeta_b = \zeta_d = 0$.

The zero-angular-momentum expression is

$$(\mathbf{0}_A|\mathbf{0}_B)^{(m)} = 2\pi^{5/2}(\zeta_a+\zeta_b)^{-1/2}(\zeta_a\zeta_b)^{-1}F_m(T) \tag{C.4}$$

where $F_m(T)$ is the Boys function (see section 2.3.4) with argument

$$T = \frac{\zeta_a\zeta_b}{\zeta_a+\zeta_b}|\mathbf{A}-\mathbf{B}|^2. \tag{C.5}$$

The VRR is

$$\begin{aligned}(\mathbf{a}+\mathbf{1}_i|\mathbf{b})^{(m)} =& \mathbf{PA}_i(\mathbf{a}|\mathbf{b})^{(m+1)} \\ &+ \frac{a_i}{2\zeta_a}\left((\mathbf{a}-\mathbf{1}_i|\mathbf{b})^{(m)} - \frac{\zeta_b}{\zeta_a+\zeta_b}(\mathbf{a}-\mathbf{1}_i|\mathbf{b})^{(m+1)}\right) \\ &+ \frac{b_i}{2(\zeta_a+\zeta_b)}(\mathbf{a}|\mathbf{b}-\mathbf{1}_i)^{(m+1)}\end{aligned} \tag{C.6}$$

where

$$\mathbf{P} = \frac{\zeta_a\mathbf{A}+\zeta_b\mathbf{B}}{\zeta_a+\zeta_b}.$$

C.3 One-index integral over a primitive Cartesian Gaussian function

The zero-angular-momentum case may be easily derived using the well-known integral over a 1D Gaussian:

$$\int_{-\infty}^{+\infty} \mathrm{d}x \exp(-(x-b)^2/c^2) = |c|\pi^{1/2}. \tag{C.7}$$

This leads to

$$(\mathbf{0}_A) = \int \mathrm{d}\mathbf{r} \exp(-\zeta_a|\mathbf{r}-\mathbf{A}|^2) = \left(\frac{\pi}{\zeta_a}\right)^{3/2}. \tag{C.8}$$

A one-term recurrence relation may be trivially derived by taking the derivative of the

integral with respect to a Cartesian component of the centre of the Gaussian, i.e.

$$\frac{\partial}{\partial A_i}(\mathbf{a}) = 2\zeta_a(\mathbf{a}+\mathbf{1}_i) - a_i(\mathbf{a}-\mathbf{1}_i) \tag{C.9}$$

Since the integral is translationally invariant (Eq. 2.153),

$$\frac{\partial}{\partial A_i}(\mathbf{a}) = 0, \tag{C.10}$$

and

$$(\mathbf{a}+\mathbf{1}_i) = \frac{a_i}{2\zeta_a}(\mathbf{a}-\mathbf{1}_i). \tag{C.11}$$

C.4 One-index nuclear attraction integrals

The zero-angular-momentum case and VRR can be derived from the corresponding expressions for two-index nuclear attraction integrals in Ref. 1 (Eqs. A14–A21), by setting $\zeta_b = 0$, $\mathbf{b} = \mathbf{0}$. The expression for the zero-angular-momentum case is

$$(\mathbf{0}_A|r_C^{-1})^{(m)} = \frac{2\pi}{\zeta_a}F_m(\zeta_a|\mathbf{A}-\mathbf{C}|^2), \tag{C.12}$$

while the VRR is

$$\begin{aligned}(\mathbf{a}+\mathbf{1}_i|r_C^{-1})^{(m)} &= -\mathbf{AC}_i(\mathbf{a}|r_C^{-1})^{(m+1)} \\ &\quad + \frac{a_i}{2\zeta_a}\left\{(\mathbf{a}-\mathbf{1}_i|r_C^{-1})^{(m)} - (\mathbf{a}-\mathbf{1}_i|r_C^{-1})^{(m+1)}\right\}.\end{aligned} \tag{C.13}$$

C.5 Four-index overlap integrals

The zero-angular-momentum case can be derived from the corresponding expression for the three-index overlap integral (Ref. 1, Eq. 21) by application of the Gaussian product theorem (see Appendix B):

$$\begin{aligned}&(\mathbf{0}_A\mathbf{0}_B\mathbf{0}_C\mathbf{0}_D) \\ &= (\zeta+\eta)^{-3/2}\pi^{3/2}\exp\left(-\frac{\zeta_a\zeta_b}{\zeta}|\mathbf{A}-\mathbf{B}|^2\right)\exp\left(-\frac{\zeta_c\zeta_d}{\eta}|\mathbf{C}-\mathbf{D}|^2\right) \\ &\quad \times \exp\left(-\frac{\zeta\eta}{\zeta+\eta}|\mathbf{P}-\mathbf{Q}|^2\right)\end{aligned} \tag{C.14}$$

with

$$\zeta = \zeta_a + \zeta_b \qquad \eta = \zeta_c + \zeta_d$$

and

$$\mathbf{P} = \frac{\zeta_a\mathbf{A}+\zeta_b\mathbf{B}}{\zeta_a+\zeta_b} \qquad \mathbf{Q} = \frac{\zeta_c\mathbf{C}+\zeta_c\mathbf{D}}{\zeta_c+\zeta_d}.$$

The VRR can be derived from the VRR for $(\mathbf{ab}|g_{12}^{u^2}|\mathbf{cd})$ (Ref. 1, Eq. 36). Taking the

exponent of the Gaussian geminal,

$$g_{12}^{u^2} \equiv \exp(-u^2|\mathbf{r}_1 - \mathbf{r}_2|^2) = \prod_{i=x,y,z} \exp(-u^2(r_{1i} - r_{2i})^2) \tag{C.15}$$

to infinity yields a product of Dirac deltas, i.e.

$$\lim_{u^2\to\infty} \left(\frac{u^2}{\pi}\right)^{3/2} \prod_{i=x,y,z} \exp(-u^2(r_{1i} - r_{2i})^2) = \prod_{i=x,y,z} \delta(r_{1i} - r_{2i}). \tag{C.16}$$

In this limit, $(\mathbf{ab}|g_{12}^{u^2}|\mathbf{cd})$ becomes equivalent to a four-index overlap integral:

$$\lim_{u^2\to\infty} \left(\frac{u^2}{\pi}\right)^{3/2} (\mathbf{ab}|g_{12}^{u^2}|\mathbf{cd}) = (\mathbf{abcd}). \tag{C.17}$$

Multiplying the VRR for $(\mathbf{ab}|g_{12}^{u^2}|\mathbf{cd})$ by $(u^2/\pi)^{3/2}$ and taking the limit $u^2 \to \infty$ yields the following VRR for four-index overlap integrals:

$$\begin{aligned}((\mathbf{a}+\mathbf{1}_i)\mathbf{bcd}) &= (W_i - A_i)(\mathbf{abcd}) \\ &+ \frac{a_i}{2\zeta}\left(1 - \frac{\rho}{\zeta}\right)((\mathbf{a}-\mathbf{1}_i)\mathbf{bcd}) + \frac{b_i}{2\zeta}\left(1 - \frac{\rho}{\zeta}\right)(\mathbf{a}(\mathbf{b}-\mathbf{1}_i)\mathbf{cd}) \\ &+ \frac{c_i}{2(\zeta+\eta)}(\mathbf{ab}(\mathbf{c}-\mathbf{1}_i)\mathbf{d}) + \frac{d_i}{2(\zeta+\eta)}(\mathbf{abc}(\mathbf{d}-\mathbf{1}_i))\end{aligned} \tag{C.18}$$

with

$$\mathbf{W} = \frac{\zeta\mathbf{P} + \eta\mathbf{Q}}{\zeta+\eta} \qquad \rho = \frac{\zeta\eta}{\zeta+\eta}.$$

C.6 Three-electron F-F and J-F integrals

Recurrence relations for the evaluation of primitive three-index three-electron F-F and J-F integrals, as required in the DF3-MP2-F12/3*A(FIX) method described in chapter 3, were previously described by May, in his PhD thesis [89]. However, in the proof-of-concept implementation of DF3-MP2-F12/3*A(FIX) presented in chapter 3, an alternative approach to the evaluation of three-electron integrals was used (see section 3.3.2) to avoid the time-consuming process of hand-coding and debugging recurrence relations. To demonstrate the effectiveness of the automatic code generator described in chapter 4, the three-index, three-electron recurrence relations for the F-F and J-F integrals were automatically implemented based on the original equations derived by May [89]. During this process, some minor errors in the equations presented in Ref. 89 were identified. The recurrence relations are reproduced here, with corrections applied.[1]

The following equations describe the evaluation of integrals over individual Gaussian geminals, e.g. $g_{12}^{\mu} \equiv g(\mathbf{r}_1;\mu,\mathbf{0},\mathbf{r}_2)$. The three-index F-F and J-F integrals, $(\mathbf{a}|f_{12}|\mathbf{b}|f_{23}|\mathbf{c})$ and

[1] To avoid introducing new errors in Eqs. C.20 to C.31, the original LaTeX source used to typeset equations in chapter 3 of Ref. 89 was used (corrected and modified to adhere to the notation used in this document) with permission from the author.

$(\mathbf{a}|r_{12}^{-1}|\mathbf{b}|f_{23}|\mathbf{c})$, are assumed to feature correlation factors f_{12} and f_{23} comprised of linear combinations of Gaussian geminals, and may therefore be constructed by summing over integrals over individual Gaussian geminals (as described in Refs. 70, 89), i.e.

$$\begin{aligned}(\mathbf{a}|f_{12}|\mathbf{b}|f_{23}|\mathbf{c}) &= \sum_{i,j} c_i c_j (\mathbf{a}|g_{12}^{\mu_i}|\mathbf{b}|g_{23}^{\nu_j}|\mathbf{c}),\\ (\mathbf{a}|r_{12}^{-1}|\mathbf{b}|f_{23}|\mathbf{c}) &= \sum_{i} c_i (\mathbf{a}|r_{12}^{-1}|\mathbf{b}|g_{23}^{\nu_i}|\mathbf{c}).\end{aligned} \tag{C.19}$$

3-electron F-F integral

The zero-angular-momentum case is

$$\begin{aligned}(\mathbf{0}_A|g_{12}^{\mu}|\mathbf{0}_B|g_{23}^{\nu}|\mathbf{0}_C) = \pi^{9/2}\omega^{-3/2}\exp\big(\,\omega^{-1}\,\big[&-|\mathbf{A}-\mathbf{B}|^2(\zeta_a\zeta_b\zeta_c\mu + \zeta_a\zeta_b\mu\nu)\\ &-|\mathbf{B}-\mathbf{C}|^2(\zeta_a\zeta_b\zeta_c\nu + \zeta_b\zeta_c\mu\nu)\\ &-|\mathbf{A}-\mathbf{C}|^2\zeta_a\zeta_c\mu\nu\big]\,\big)\end{aligned} \tag{C.20}$$

where

$$\omega = \zeta_a\zeta_b\zeta_c + \zeta_a\zeta_c\mu + \zeta_b\zeta_c\mu + \zeta_a\zeta_b\nu + \zeta_a\zeta_c\nu + \zeta_a\mu\nu + \zeta_b\mu\nu + \zeta_c\mu\nu. \tag{C.21}$$

The VRRs for incrementing angular momentum in $|\mathbf{a})$, $|\mathbf{b})$ and $|\mathbf{c})$ are as follows:

$$\begin{aligned}(\mathbf{a}+\mathbf{1}_i|g_{12}^{\mu}|\mathbf{b}|g_{23}^{\nu}|\mathbf{c}) =& -\frac{\mu}{\omega}\left[(\zeta_c\nu + \zeta_b(\zeta_c+\nu))\mathbf{AB}_i + \zeta_c\nu\mathbf{BC}_i\right](\mathbf{a}|g_{12}^{\mu}|\mathbf{b}|g_{23}^{\nu}|\mathbf{c})\\ &+\frac{\zeta_c(\zeta_b+\mu)+\nu(\zeta_b+\zeta_c+\mu)}{2\omega}a_i(\mathbf{a}-\mathbf{1}_i|g_{12}^{\mu}|\mathbf{b}|g_{23}^{\nu}|\mathbf{c})\\ &+\frac{\mu(\zeta_c+\nu)b_i}{2\omega}(\mathbf{a}|g_{12}^{\mu}|\mathbf{b}-\mathbf{1}_i|g_{23}^{\nu}|\mathbf{c})\\ &+\frac{\mu\nu c_i}{2\omega}(\mathbf{a}|g_{12}^{\mu}|\mathbf{b}|g_{23}^{\nu}|\mathbf{c}-\mathbf{1}_i),\end{aligned} \tag{C.22}$$

$$\begin{aligned}(\mathbf{a}|g_{12}^{\mu}|\mathbf{b}+\mathbf{1}_i|g_{23}^{\nu}|\mathbf{c}) =& \frac{\zeta_a\mu(\zeta_c+\nu)\mathbf{AB}_i - \zeta_c\nu(\zeta_a+\mu)\mathbf{BC}_i}{\omega}(\mathbf{a}|g_{12}^{\mu}|\mathbf{b}|g_{23}^{\nu}|\mathbf{c})\\ &+\frac{\mu(\zeta_c+\nu)a_i}{2\omega}(\mathbf{a}-\mathbf{1}_i|g_{12}^{\mu}|\mathbf{b}|g_{23}^{\nu}|\mathbf{c})\\ &+\frac{(\zeta_a+\mu)(\zeta_c+\nu)b_i}{2\omega}(\mathbf{a}|g_{12}^{\mu}|\mathbf{b}-\mathbf{1}_i|g_{23}^{\nu}|\mathbf{c})\\ &+\frac{\nu(\zeta_a+\mu)c_i}{2\omega}(\mathbf{a}|g_{12}^{\mu}|\mathbf{b}|g_{23}^{\nu}|\mathbf{c}-\mathbf{1}_i),\end{aligned} \tag{C.23}$$

$$\begin{aligned}(\mathbf{a}|g_{12}^{\mu}|\mathbf{b}|g_{23}^{\nu}|\mathbf{c}+\mathbf{1}_i) =& \frac{\nu}{\omega}\left[\zeta_a\mu\mathbf{AB}_i + (\zeta_a\mu + \zeta_b(\zeta_a+\mu)\mathbf{BC}_i)\right](\mathbf{a}|g_{12}^{\mu}|\mathbf{b}|g_{23}^{\nu}|\mathbf{c})\\ &+\frac{\mu\nu a_i}{2\omega}(\mathbf{a}-\mathbf{1}_i|g_{12}^{\mu}|\mathbf{b}|g_{23}^{\nu}|\mathbf{c})\\ &+\frac{\nu(\zeta_a+\mu)b_i}{2\omega}(\mathbf{a}|g_{12}^{\mu}|\mathbf{b}-\mathbf{1}_i|g_{23}^{\nu}|\mathbf{c})\\ &+\frac{\zeta_a(\zeta_b+\nu)+\mu(\zeta_b+\zeta_a+\nu)}{2\omega}c_i(\mathbf{a}|g_{12}^{\mu}|\mathbf{b}|g_{23}^{\nu}|\mathbf{c}-\mathbf{1}_i).\end{aligned} \tag{C.24}$$

3-electron J-F integral

The zero-angular-momentum case is

$$(\mathbf{0}_A|r_{12}^{-1}|\mathbf{0}_B|g_{23}^{\nu}|\mathbf{0}_C)^{(m)} = \frac{2\pi^4}{\eta^{3/2}\rho}\exp\left(-\frac{\zeta_a\zeta_b\zeta_c\nu}{\eta\rho}|\mathbf{B}-\mathbf{C}|^2\right)F_m(T). \tag{C.25}$$

where $F_m(T)$ is the Boys function (section 2.3.4),

$$T = \frac{1}{\eta\rho}\left\{\zeta_a\zeta_c\nu\rho|\mathbf{A}-\mathbf{C}|^2 + \zeta_a\zeta_b\rho(\zeta_c+\nu)|\mathbf{A}-\mathbf{B}|^2 + \zeta_b\zeta_c\nu(\rho-\zeta_a)|\mathbf{B}-\mathbf{C}|^2\right\}, \tag{C.26}$$

with

$$\rho = \frac{\zeta_a\zeta_b\zeta_c + \zeta_a\nu(\zeta_b+\zeta_c)}{\eta}, \tag{C.27}$$

and

$$\eta = \zeta_c(\zeta_a+\zeta_b) + \nu(\zeta_a+\zeta_b+\zeta_c). \tag{C.28}$$

The VRRs for incrementing angular momentum in $|\mathbf{a})$, $|\mathbf{b})$ and $|\mathbf{c})$ are as follows:

$$\begin{aligned}
&(\mathbf{a}+\mathbf{1}_i|r_{12}^{-1}|\mathbf{b}|g_{23}^{\nu}|\mathbf{c})^{(m)} \\
&= -\frac{(\zeta_c\nu+\zeta_b(\zeta_c+\nu))\mathbf{AB}_i + \zeta_c\nu\mathbf{BC}_i}{\eta}(\mathbf{a}|r_{12}^{-1}|\mathbf{b}|g_{23}^{\nu}|\mathbf{c})^{(m+1)} \\
&\quad + \frac{a_i}{2\eta\rho}\Big[(\zeta_c\nu+\zeta_b(\zeta_c+\nu))(\mathbf{a}-\mathbf{1}_i|r_{12}^{-1}|\mathbf{b}|g_{23}^{\nu}|\mathbf{c})^{(m)} \\
&\quad -(\zeta_c\nu+\zeta_b(\zeta_c+\nu)-\rho(\zeta_c+\nu))(\mathbf{a}-\mathbf{1}_i|r_{12}^{-1}|\mathbf{b}|g_{23}^{\nu}|\mathbf{c})^{(m+1)}\Big] \\
&\quad + \frac{(\zeta_c+\nu)b_i}{2\eta}(\mathbf{a}|r_{12}^{-1}|\mathbf{b}-\mathbf{1}_i|g_{23}^{\nu}|\mathbf{c})^{(m+1)} \\
&\quad + \frac{\nu c_i}{2\eta}(\mathbf{a}|r_{12}^{-1}|\mathbf{b}|g_{23}^{\nu}|\mathbf{c}-\mathbf{1}_i)^{(m+1)},
\end{aligned} \tag{C.29}$$

$$\begin{aligned}
&(\mathbf{a}|r_{12}^{-1}|\mathbf{b}+\mathbf{1}_i|g_{23}^{\nu}|\mathbf{c})^{(m)} = \frac{1}{\eta\rho}\Big[-\zeta_a\zeta_c\nu\mathbf{BC}_i(\mathbf{a}|r_{12}^{-1}|\mathbf{b}|g_{23}^{\nu}|\mathbf{c})^{(m)} \\
&+(\zeta_a\rho(\zeta_c+\nu)\mathbf{AB}_i - \zeta_c\nu(\rho-\zeta_a)\mathbf{BC}_i)(\mathbf{a}|r_{12}^{-1}|\mathbf{b}|g_{23}^{\nu}|\mathbf{c})^{(m+1)}\Big] \\
&+ \frac{(\zeta_c+\nu)a_i}{2\eta}(\mathbf{a}-\mathbf{1}_i|r_{12}^{-1}|\mathbf{b}|g_{23}^{\nu}|\mathbf{c})^{(m+1)} \\
&+ \frac{(\zeta_c+\nu)b_i}{2\eta\rho}\Big[\zeta_a(\mathbf{a}|r_{12}^{-1}|\mathbf{b}-\mathbf{1}_i|g_{23}^{\nu}|\mathbf{c})^{(m)} - (\zeta_a-\rho)(\mathbf{a}|r_{12}^{-1}|\mathbf{b}-\mathbf{1}_i|g_{23}^{\nu}|\mathbf{c})^{(m+1)}\Big] \\
&+ \frac{\nu c_i}{2\eta\rho}\Big[\zeta_a(\mathbf{a}|r_{12}^{-1}|\mathbf{b}|g_{23}^{\nu}|\mathbf{c}-\mathbf{1}_i)^{(m)} - (\zeta_a-\rho)(\mathbf{a}|r_{12}^{-1}|\mathbf{b}|g_{23}^{\nu}|\mathbf{c}-\mathbf{1}_i)^{(m+1)}\Big],
\end{aligned} \tag{C.30}$$

$$
\begin{aligned}
&(\mathbf{a}|r_{12}^{-1}|\mathbf{b}|g_{23}^{\nu}|\mathbf{c}+\mathbf{1}_i)^{(m)} \\
&= \frac{\nu}{\eta\rho}\Big[\zeta_a\zeta_b\mathbf{BC}_i(\mathbf{a}|r_{12}^{-1}|\mathbf{b}|g_{23}^{\nu}|\mathbf{c})^{(m)} \\
&\quad +(\zeta_a\rho\mathbf{AB}_i+(\rho(\zeta_a+\zeta_b)-\zeta_a\zeta_b)\mathbf{BC}_i)(\mathbf{a}|r_{12}^{-1}|\mathbf{b}|g_{23}^{\nu}|\mathbf{c})^{(m+1)}\Big] \\
&\quad +\frac{\nu a_i}{2\eta}(\mathbf{a}-\mathbf{1}_i|r_{12}^{-1}|\mathbf{b}|g_{23}^{\nu}|\mathbf{c})^{(m+1)} \\
&\quad +\frac{\nu b_i}{2\eta\rho}\Big[\zeta_a(\mathbf{a}|r_{12}^{-1}|\mathbf{b}-\mathbf{1}_i|g_{23}^{\nu}|\mathbf{c})^{(m)}+(\zeta_a-\rho)(\mathbf{a}|r_{12}^{-1}|\mathbf{b}-\mathbf{1}_i|g_{23}^{\nu}|\mathbf{c})^{(m+1)}\Big] \\
&\quad +\frac{c_i}{2\eta\rho}\Big[\zeta_a(\zeta_b+\nu)(\mathbf{a}|r_{12}^{-1}|\mathbf{b}|g_{23}^{\nu}|\mathbf{c}-\mathbf{1}_i)^{(m)} \\
&\quad -(\zeta_a(\zeta_b+\nu)-\rho(\zeta_a+\zeta_b+\nu))(\mathbf{a}|r_{12}^{-1}|\mathbf{b}|g_{23}^{\nu}|\mathbf{c}-\mathbf{1}_i)^{(m+1)}\Big]\,.
\end{aligned}
\tag{C.31}
$$

APPENDIX D

IMPLEMENTATION OF THE BOYS FUNCTION

The implementation of the Boys function (section 2.3.4) used in chapter 4 is based on an approach originally implemented by Kaito Miyamoto (2014) in Fortran (using Refs. 10 (pp.365–372), 260 and 261). This was re-implemented in Intception such that a C99 version very close to the original Fortran source could be automatically generated as needed. This is the default Boys function implementation in Intception at the time of writing.

To ensure the error in evaluating $F_m(T)$ is always less than 10^{-14}, different methods are applied depending on the value of T. There are three regimes:

Small T ($0 \leq T \leq 15$)

Use a Taylor expansion expanded around precomputed data points $F_m(T_t)$ at intervals of 0.1 to calculate $F_m(T)$ for the maximum m value required, i.e.

$$F_m(T_t + \Delta T) = \sum_{k=0}^{\infty} \frac{1}{k!} F_{m+k}(T_t)(-\Delta T)^k. \tag{D.1}$$

In practice a seven-term expansion is used. For smaller values of m, downward recursion using

$$F_m(T) = \frac{1}{2m+1}(2TF_{m+1}(T) + \exp(-T)) \tag{D.2}$$

is employed. This is based on Ref. 10 (pp.365–372).

Medium T ($15 < T < 35$)

Use a Taylor expansion expanded around precomputed data points $F_0(T_t)$ at intervals of 0.1 to calculate $F_0(T)$, i.e.

$$F_0(T_t + \Delta T) = \sum_{k=0}^{\infty} \frac{1}{k!} F_k(T_t)(-\Delta T)^k \tag{D.3}$$

A seven-term expansion is also used here. For $m > 0$, upward recursion using

$$F_{m+1}(T) = \frac{1}{2T}((2m+1)F_m(T) - \exp(-T)) \tag{D.4}$$

is employed. This is based on Ref. 10 (pp.365–372).

Large T ($35 \leq T$)

Calculate $F_0(T)$ using an approximate expression, suitable for large values of T [260, 261]:

$$F_0(T) \approx \frac{1}{2}\left(\frac{\pi}{T}\right)^{1/2}. \tag{D.5}$$

Calculate values for $m > 0$ using upward recursion (Eq. D.4), using the approximation $\exp(-T) = 0$, i.e.

$$F_{m+1}(T) = \frac{1}{2T}(2m+1)F_m(T). \tag{D.6}$$

APPENDIX E

RESOLUTION OF THE IDENTITY AS A FITTING PROCEDURE

In Ref. 131, the resolution of the identity (RI), as applied in explicitly correlated methods (sections 2.2.4 and 3.2.4), is described in terms of a fitting procedure. Since this relationship is relevant to the work presented in chapter 3, the derivation in Ref. 131 is reproduced and expanded upon here.

We start by defining an approximate resolution of the identity for a general three-electron integral:

$$\begin{aligned}\langle ijk|\hat{\nu}_{12}\hat{\omega}_{23}|nml\rangle &\approx \langle ijk|\hat{\nu}_{12}\hat{X}_2\hat{\omega}_{23}|nml\rangle \\ &= \langle ij|\hat{\nu}_{12}|nx\rangle\langle kx|\hat{\omega}_{12}|lm\rangle\end{aligned} \tag{E.1}$$

where $\hat{\nu}_{12}$ and $\hat{\omega}_{23}$ are two-electron operators (e.g. r_{12}^{-1}, f_{12}) and $\hat{X}_2 = |x\rangle\langle x|$ is an approximation to the identity operator in the orthonormal RI basis $\{|x\rangle\}$. In Eq. E.1 we have used the implicit summation notation described at the start of chapter 3—this will also be used in the following derivation.

Assuming that the operators $\hat{\nu}_{12}$ and $\hat{\omega}_{23}$ commute with the MOs, the three-electron integral in Eq. E.1 can be expressed as the overlap of two three-index objects:

$$|\psi_{in}^{j}\rangle = \int \mathrm{d}\mathbf{r}_1\, \psi_i(\mathbf{r}_1)\psi_n^*(\mathbf{r}_1)\hat{\nu}_{12}\psi_j(\mathbf{r}_2) \tag{E.2}$$

and,

$$|\phi_{kl}^{m}\rangle = \int \mathrm{d}\mathbf{r}_3\, \psi_k^*(\mathbf{r}_3)\psi_l(\mathbf{r}_3)\hat{\omega}_{23}\psi_m(\mathbf{r}_2), \tag{E.3}$$

i.e.

$$\langle ijk|\hat{\nu}_{12}\hat{\omega}_{23}|nml\rangle \equiv (in|\hat{\nu}_{12}|jm|\hat{\omega}_{23}|kl) \equiv \langle\psi_{in}^{j}|\phi_{kl}^{m}\rangle. \tag{E.4}$$

We can fit the three-index objects in an orthonormal basis $\{|x\rangle\}$,

$$|\widetilde{\psi_{in}^{j}}\rangle = C_{\psi,in}^{j;x}|x\rangle \tag{E.5}$$

$$|\widetilde{\phi_{kl}^{m}}\rangle = C_{\phi,kl}^{m;x}|x\rangle \tag{E.6}$$

by minimizing the overlap of the fitting residuals

$$\Delta_\psi = \langle \psi^j_{in} - \widetilde{\psi^j_{in}} | \psi^j_{in} - \widetilde{\psi^j_{in}} \rangle \tag{E.7}$$

$$\Delta_\phi = \langle \phi^m_{kl} - \widetilde{\phi^m_{kl}} | \phi^m_{kl} - \widetilde{\phi^m_{kl}} \rangle. \tag{E.8}$$

Provided $\{|x\rangle\}$ is orthonormal,

$$\Delta_\psi = \langle \psi^j_{in} | \psi^j_{in} \rangle + (C^{j;x}_{\psi,in})^2 - 2\langle \psi^j_{in} | x \rangle C^{j;x}_{\psi,in} \tag{E.9}$$

$$\Delta_\phi = \langle \psi^m_{kl} | \psi^m_{kl} \rangle + (C^{m;x}_{\phi,kl})^2 - 2\langle \psi^m_{kl} | x \rangle C^{m;x}_{\phi,kl}. \tag{E.10}$$

Evaluating $\partial\Delta_\psi/\partial C^{j;y}_{\psi,in}$ and $\partial\Delta_\phi/\partial C^{m;y}_{\phi,kl}$ and setting these to zero gives

$$C^{j;y}_{\psi,in} = \langle \psi^j_{in} | y \rangle = (in|\hat{\nu}_{12}|jy) = \langle ij|\hat{\nu}_{12}|ny\rangle \tag{E.11}$$

$$C^{m;y}_{\phi,kl} = \langle \phi^m_{kl} | y \rangle = (lk|\hat{\omega}_{23}|my) = \langle lm|\hat{\omega}_{12}|ky\rangle. \tag{E.12}$$

The expression for three-electron integral fitted using this scheme is thus

$$\begin{aligned} \langle \widetilde{\psi^j_{in}} | \widetilde{\phi^m_{kl}} \rangle &= C^{j;x}_{\psi,in} \langle x|y\rangle C^{m;y}_{\phi,kl} = C^{j;x}_{\psi,in} C^{m;x}_{\phi,kl} \\ &= \langle ij|\hat{\nu}_{12}|nx\rangle\langle kx|\hat{\omega}_{12}|lm\rangle \end{aligned} \tag{E.13}$$

which is identical to the expression obtained for applying the resolution of the identity in Eq. E.1.

As noted in Ref. 131, this fitting procedure is robust (see section 2.4), since although the error in the fitted integral is given by

$$\begin{aligned} \langle \psi^j_{in} | \phi^m_{kl} \rangle - \langle \widetilde{\psi^j_{in}} | \widetilde{\phi^m_{kl}} \rangle &= \langle \psi^j_{in} - \widetilde{\psi^j_{in}} | \phi^m_{kl} - \widetilde{\phi^m_{kl}} \rangle \\ &\quad + \langle \psi^j_{in} | \widetilde{\phi^m_{kl}} \rangle + \langle \widetilde{\psi^j_{in}} | \phi^m_{kl} \rangle - 2\langle \widetilde{\psi^j_{in}} | \widetilde{\phi^m_{kl}} \rangle, \end{aligned} \tag{E.14}$$

the last three terms cancel (using Eqs. E.11 and E.12)

$$\begin{aligned} &\langle \psi^j_{in} | \widetilde{\phi^m_{kl}} \rangle + \langle \widetilde{\psi^j_{in}} | \phi^m_{kl} \rangle - 2\langle \widetilde{\psi^j_{in}} | \widetilde{\phi^m_{kl}} \rangle \\ &\quad = C^{m;x}_{\phi,kl} \left(\langle \psi^j_{in} | x \rangle - C^{j;x}_{\psi,in} \right) + C^{j;x}_{\psi,in} \left(\langle x | \phi^m_{kl} \rangle - C^{m;x}_{\phi,kl} \right) = 0 \end{aligned} \tag{E.15}$$

and thus

$$\langle \psi^j_{in} | \phi^m_{kl} \rangle - \langle \widetilde{\psi^j_{in}} | \widetilde{\phi^m_{kl}} \rangle = \langle \psi^j_{in} - \widetilde{\psi^j_{in}} | \phi^m_{kl} - \widetilde{\phi^m_{kl}} \rangle, \tag{E.16}$$

i.e. the error in the fitted integral is quadratic in the error in the fitted objects (Eqs. E.5 and E.6).

APPENDIX F

GAUSSIAN TRANSFORM OF A CARTESIAN GAUSSIAN

In section 3.3.2 an integral transform of a primitive Cartesian Gaussian (which we refer to as a "Gaussian transform") is used to transform two-electron, three-index integrals into three-electron, three-index integrals, i.e.

$$(\mathbf{a}|\hat{\omega}_{12}|\mathbf{b}|g^{\mu}_{23}|\mathbf{c}) = C^{\mu}_{\mathbf{cc'}}(\mathbf{a}|\hat{u}_{12}|\mathbf{bc'}) \tag{F.1}$$

where $\hat{\omega}_{12}$ is a general two-electron operator, the summation over $\mathbf{c'}$ is implicit (see the note on notation at the start of chapter 3), running from $\mathbf{c'} = (0,0,0)$ to $\mathbf{c'} = (c_x, c_y, c_z)$, and with Gaussian geminal $g^{\mu}_{23} \equiv g(\mathbf{r}_2; \mu, \mathbf{0}, \mathbf{r}_3)$.

The Gaussian transform was described in unpublished work by F.R. Manby and A.J. May, with the result reproduced in Ref. 150. Since the derivation of the coefficients $C^{\mu}_{\mathbf{cc'}}$ does not appear to have been published elsewhere, a description is provided here.

The Gaussian transform may be written as follows

$$\begin{aligned}\int \mathrm{d}\mathbf{r}_2\, g^{\mu}_{12} g(\mathbf{r}_2; \zeta_a, \mathbf{a}, \mathbf{A}) &\equiv \langle \mathbf{0}_{\mathbf{r}_1} | \mathbf{a} \rangle \\ &= C^{\mu}_{\mathbf{aa'}} g\left(\mathbf{r}_1; \frac{\zeta_a \mu}{\zeta_a + \mu}, \mathbf{a'}, \mathbf{A}\right)\end{aligned} \tag{F.2}$$

where the summation over $\mathbf{a'}$ runs from $(0,0,0)$ to (a_x, a_y, a_z). The coefficients $C^{\mu}_{\mathbf{aa'}}$ are obtained as follows. First, applying the Gaussian product theorem (appendix B) yields

$$\begin{aligned}\langle \mathbf{0}_{\mathbf{r}_1} | \mathbf{a} \rangle &= \exp(-\frac{\zeta_a \mu}{\zeta_a + \mu} |\mathbf{r}_1 - \mathbf{A}|^2) \\ &\times \int \mathrm{d}\mathbf{r}_2 (r_{2x} - A_x)^{a_x} (r_{2y} - A_y)^{a_y} (r_{2z} - A_z)^{a_z} \exp(-(\zeta_a + \mu)|\mathbf{r}_2 - \mathbf{P}|^2)\end{aligned} \tag{F.3}$$

with $\mathbf{P} = (\zeta_a \mathbf{A} + \mu \mathbf{r_1})/(\zeta_a + \mu)$. Using

$$\begin{aligned} r_{2i} - A_i &= (r_{2i} - P_i) + (P_i - A_i) \\ &= (r_{2i} - P_i) + \frac{\mu}{\zeta_a + \mu}(r_{1i} - A_i)\end{aligned} \tag{F.4}$$

and applying the binomial theorem

$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^{n-k} b^k \tag{F.5}$$

to $(r_{2i} - A_i)^{a_i}$, as expanded in Eq. F.4, gives

$$(r_{2i} - A_i)^{a_i} = \binom{a_i}{a_i'} \left(\frac{\mu}{\zeta_a + \mu}\right)^{a_i'} (r_{2i} - P_i)^{a_i - a_i'} (r_{1i} - A_i)^{a_i'}, \tag{F.6}$$

where the implicit summation over a_i' runs from 0 to a_i.

Inserting Eq. F.6 into Eq. F.3 for $i = x, y, z$ gives

$$\langle \mathbf{0}_{\mathbf{r}_1} | \mathbf{a} \rangle = g\left(\mathbf{r}_1; \frac{\zeta_a \mu}{\zeta_a + \mu}, \mathbf{a}', \mathbf{A}\right) \binom{a_x}{a_x'} \binom{a_y}{a_y'} \binom{a_z}{a_z'} \left(\frac{\mu}{\zeta_a + \mu}\right)^{a_x' + a_y' + a_z'} L_{\mathbf{a}-\mathbf{a}'} \tag{F.7}$$

where

$$\begin{aligned} L_{\mathbf{a}-\mathbf{a}'} &= \prod_{i=x,y,z} \int_{-\infty}^{\infty} \mathrm{d}r_{2i}\, (r_{2i} - P_i)^{a_i - a_i'} \exp(-(\zeta_a + \mu)(r_{2i} - P_i)^2) \\ &= \prod_{i=x,y,z} \int_{-\infty}^{\infty} \mathrm{d}r_i\, r_i^{a_i - a_i'} \exp(-(\zeta_a + \mu) r_i^2). \end{aligned} \tag{F.8}$$

Thus, the Gaussian transform coefficients of Eq. F.2 are

$$C_{\mathbf{a}\mathbf{a}'}^{\mu} = \binom{a_x}{a_x'} \binom{a_y}{a_y'} \binom{a_z}{a_z'} \left(\frac{\mu}{\zeta_a + \mu}\right)^{a_x' + a_y' + a_z'} L_{\mathbf{a}-\mathbf{a}'}. \tag{F.9}$$

APPENDIX G

FT INTEGRALS

In Ref. 184, Klopper and Röhse derived an expression for the $(\mathbf{ab}|[r_{12},\hat{t}_1]|\mathbf{cd})$ integrals that arise in R12 theory:

$$(\mathbf{ab}|[r_{12},\hat{t}_1]|\mathbf{cd}) = \frac{\zeta_a-\zeta_b}{\zeta_a+\zeta_b}(\mathbf{ab}|r_{12}^{-1}|\mathbf{cd}) + \nabla_P\cdot\nabla_R(\mathbf{ab}|r_{12}|\mathbf{cd}), \tag{G.1}$$

with kinetic energy operator $\hat{t}_1 = -\frac{1}{2}\nabla_1^2$. Later, Manby and May [70] published a similar expression for the $(\mathbf{ab}|[\hat{t}_1,f_{12}]|\mathbf{c})$ integrals, which arose in their formulation of MP2-F12 theory with f_{12} comprised of Gaussian geminals. As we shall see, the equation in Ref. 70 is in error by a factor of -1, and the correct expression is:

$$(\mathbf{ab}|[\hat{t}_1,f_{12}]|\mathbf{c}) = -\frac{1}{2}\frac{\zeta_a-\zeta_b}{\zeta_a+\zeta_b}(\mathbf{ab}|\{\nabla_1^2 f_{12}\}|\mathbf{c}) - \nabla_P\cdot\nabla_R(\mathbf{ab}|f_{12}|\mathbf{c}). \tag{G.2}$$

The derivation of this expression (important in the derivation of the FT1-F type integrals in chapter 3) does not appear to have been previously published, despite the result appearing in Ref. 70. We therefore provide below a derivation following Klopper and Röhse's original work, but with a Gaussian geminal type f_{12} correlation factor, i.e.

$$f_{12} = \sum_i c_i \exp(-\mu_i|\mathbf{r}_1-\mathbf{r}_2|^2). \tag{G.3}$$

We start by defining the "FT" integrals,

$$(\mathbf{ab}|[\hat{t}_1,f_{12}]|\mathbf{c}) = \int \mathrm{d}\mathbf{r}_1\mathrm{d}\mathbf{r}_2 g_a(\mathbf{r}_1)[\hat{t}_1,f_{12}]g_b(\mathbf{r}_1)g_c(\mathbf{r}_2) \tag{G.4}$$

where $g_a(\mathbf{r}_1) = g(\mathbf{r}_1;\zeta_a,\mathbf{a},\mathbf{A})$ is a primitive Gaussian function.

Expanding the commutator, we obtain

$$(\mathbf{ab}|[\hat{t}_1,f_{12}]|\mathbf{c}) = -\frac{1}{2}(\mathbf{ab}|\nabla_1^2 f_{12}|\mathbf{c}) + \frac{1}{2}(\mathbf{ab}|f_{12}\nabla_1^2|\mathbf{c}). \tag{G.5}$$

Now consider the action of ∇_1^2 in the first term, using the product rule:

$$\begin{aligned}\nabla_1^2 f_{12} g_b(\mathbf{r}_1) &= \nabla_1 \cdot (\nabla_1 f_{12} g_b(\mathbf{r}_1)) \\ &= \nabla_1 \cdot (g_b(\mathbf{r}_1)\nabla_1 f_{12} + f_{12}\nabla_1 g_b(\mathbf{r}_1)) \\ &= (\nabla_1 g_b(\mathbf{r}_1)) \cdot (\nabla_1 f_{12}) + g_b(\mathbf{r}_1)\nabla_1 \cdot (\nabla_1 f_{12}) \\ &\quad + (\nabla_1 f_{12}) \cdot (\nabla_1 g_b(\mathbf{r}_1)) + f_{12}\nabla_1 \cdot (\nabla_1 g_b(\mathbf{r}_1)) \\ &= 2(\nabla_1 g_b(\mathbf{r}_1)) \cdot (\nabla_1 f_{12}) + g_b(\mathbf{r}_1)\nabla_1^2 f_{12} + f_{12}\nabla_1^2 g_b(\mathbf{r}_1).\end{aligned} \tag{G.6}$$

Substituting this into Eq. G.5, we obtain

$$(\mathbf{ab}|[\hat{t}_1, f_{12}]|\mathbf{c}) = -(\mathbf{ab}|(\nabla_1 f_{12}) \cdot \nabla_1|\mathbf{c}) - \frac{1}{2}(\mathbf{ab}|\{\nabla_1^2 f_{12}\}|\mathbf{c}) \tag{G.7}$$

where the $(\mathbf{ab}|f_{12}\nabla_1^2|\mathbf{c})$ terms cancel, and

$$(\mathbf{ab}|(\nabla_1 f_{12}) \cdot \nabla_1|\mathbf{c}) = \int \mathrm{d}\mathbf{r}_1 \mathrm{d}\mathbf{r}_2 g_a(\mathbf{r}_1) g_c(\mathbf{r}_2)(\nabla_1 f_{12}) \cdot (\nabla_1 g_b(\mathbf{r}_1)) \tag{G.8}$$

and

$$(\mathbf{ab}|\{\nabla_1^2 f_{12}\}|\mathbf{c}) = \int \mathrm{d}\mathbf{r}_1 \mathrm{d}\mathbf{r}_2 g_a(\mathbf{r}_1)\{\nabla_1^2 f_{12}\} g_b(\mathbf{r}_1) g_c(\mathbf{r}_2). \tag{G.9}$$

Since $g_b(\mathbf{r}_1)$ is a Gaussian function centred on $\mathbf{B}$, we recognize that

$$\nabla_1 g_b(\mathbf{r}_1) = -\nabla_B g_b(\mathbf{r}_1) \tag{G.10}$$

and thus that

$$(\mathbf{ab}|(\nabla_1 f_{12}) \cdot \nabla_1|\mathbf{c}) = -\nabla_B(\mathbf{ab}|(\nabla_1 f_{12})|\mathbf{c}). \tag{G.11}$$

At this point, define some new coordinates:

$$\mathbf{P} = \frac{\zeta_a}{\zeta}\mathbf{A} + \frac{\zeta_b}{\zeta}\mathbf{B} \tag{G.12}$$

and

$$\mathbf{R} = \mathbf{A} - \mathbf{B} \tag{G.13}$$

where $\zeta = \zeta_a + \zeta_b$.

We can express ∇_A and ∇_B in terms of ∇_P and ∇_R using the total differential. Consider an arbitrary function $f(P_x, R_x)$. Since P_x and R_x both depend on A_x we must use the total differential [262, p.145] to find $\frac{\partial f(P_x,R_x)}{\partial A_x}$:

$$\begin{aligned}\frac{\partial f(P_x, R_x)}{\partial A_x} &= \frac{\partial P_x}{\partial Ax}\frac{\partial f(P_x, R_x)}{\partial P_x} + \frac{\partial R_x}{\partial Ax}\frac{\partial f(P_x, R_x)}{\partial R_x} \\ &= \frac{\zeta_a}{\zeta}\frac{\partial f(P_x, R_x)}{\partial P_x} + \frac{\partial f(P_x, R_x)}{\partial R_x}.\end{aligned} \tag{G.14}$$

Suppressing the f, we have

$$\frac{\partial}{\partial A_x} = \frac{\zeta_a}{\zeta}\frac{\partial}{\partial P_x} + \frac{\partial}{\partial R_x} \tag{G.15}$$

which implies

$$\nabla_A = \frac{\zeta_a}{\zeta}\nabla_P + \nabla_R. \tag{G.16}$$

Using the same procedure for ∇_B, we find that

$$\nabla_B = \frac{\zeta_b}{\zeta}\nabla_P - \nabla_R. \tag{G.17}$$

Substituting this into Eq. G.11 yields

$$(\mathbf{ab}|(\nabla_1 f_{12})\cdot\nabla_1|\mathbf{c}) = -\frac{\zeta_b}{\zeta}\nabla_P(\mathbf{ab}|(\nabla_1 f_{12})|\mathbf{c}) + \nabla_R(\mathbf{ab}|(\nabla_1 f_{12})|\mathbf{c}). \tag{G.18}$$

We will shortly make use of the relation

$$\nabla_P g_a(\mathbf{r}_1)g_b(\mathbf{r}_1) = -\nabla_1 g_a(\mathbf{r}_1)g_b(\mathbf{r}_1) \tag{G.19}$$

from Ref. 184, which can be derived as follows:

$$\begin{aligned}
\nabla_1 g_a(\mathbf{r}_1)g_b(\mathbf{r}_1) &= g_a(\mathbf{r}_1)\nabla_1 g_b(\mathbf{r}_1) + g_b(\mathbf{r}_1)\nabla_1 g_a(\mathbf{r}_1) \\
&= -\nabla_A g_a(\mathbf{r}_1)g_b(\mathbf{r}_1) - \nabla_B g_a(\mathbf{r}_1)g_b(\mathbf{r}_1) \\
&= -\left[\left(\frac{\zeta_a}{\zeta}\nabla_P + \nabla_R\right) g_a(\mathbf{r}_1)g_b(\mathbf{r}_1) + \left(\frac{\zeta_b}{\zeta}\nabla_P - \nabla_R\right) g_a(\mathbf{r}_1)g_b(\mathbf{r}_1)\right] \\
&= -\left[\frac{\zeta_a}{\zeta}\nabla_P g_a(\mathbf{r}_1)g_b(\mathbf{r}_1) + \frac{\zeta_b}{\zeta}\nabla_P g_a(\mathbf{r}_1)g_b(\mathbf{r}_1)\right] \\
&= -\frac{\zeta_a+\zeta_b}{\zeta_a+\zeta_b}\left[\nabla_P g_a(\mathbf{r}_1)g_b(\mathbf{r}_1)\right] \\
&= -\nabla_P g_a(\mathbf{r}_1)g_b(\mathbf{r}_1).
\end{aligned} \tag{G.20}$$

Klopper and Röhse state that

$$\nabla_P(\mathbf{ab}|M_{12}|\mathbf{cd}) = (\mathbf{ab}|\{\nabla_1 M_{12}\}|\mathbf{cd}) \tag{G.21}$$

where $M_{12} = \frac{\mathbf{r}_{12}}{r_{12}}$ or $M_{12} = r_{12}$. This is because

$$\nabla_1\cdot(g_a(\mathbf{r}_1)g_b(\mathbf{r}_1)M_{12}) = g_a(\mathbf{r}_1)g_b(\mathbf{r}_1)\nabla_1\cdot M_{12} + M_{12}\cdot(\nabla_1 g_a(\mathbf{r}_1)g_b(\mathbf{r}_1)) \tag{G.22}$$

for vector M_{12}, or

$$\nabla_1(g_a(\mathbf{r}_1)g_b(\mathbf{r}_1)M_{12}) = g_a(\mathbf{r}_1)g_b(\mathbf{r}_1)\{\nabla_1 M_{12}\} + M_{12}(\nabla_1 g_a(\mathbf{r}_1)g_b(\mathbf{r}_1)) \tag{G.23}$$

for scalar M_{12}. This implies

$$M_{12}(\nabla_1 g_a(\mathbf{r}_1)g_b(\mathbf{r}_1)) = -g_a(\mathbf{r}_1)g_b(\mathbf{r}_1)\{\nabla_1 M_{12}\} + \nabla_1(g_a(\mathbf{r}_1)g_b(\mathbf{r}_1)M_{12}) \tag{G.24}$$

and thus, using Eq. G.19

$$\begin{aligned}\nabla_P(\mathbf{ab}|M_{12}|\mathbf{cd}) &= -\int \mathrm{d}\mathbf{r}_1\mathrm{d}\mathbf{r}_2(\nabla_1 g_a(\mathbf{r}_1)g_b(\mathbf{r}_1))M_{12}g_c(\mathbf{r}_2)g_d(\mathbf{r}_2) \\ &= \int \mathrm{d}\mathbf{r}_1\mathrm{d}\mathbf{r}_2 g_a(\mathbf{r}_1)g_b(\mathbf{r}_1)\{\nabla_1 M_{12}\}g_c(\mathbf{r}_2)g_d(\mathbf{r}_2) \\ &\quad - \int \mathrm{d}\mathbf{r}_1\mathrm{d}\mathbf{r}_2\nabla_1(g_a(\mathbf{r}_1)g_b(\mathbf{r}_1)M_{12})g_c(\mathbf{r}_2)g_d(\mathbf{r}_2).\end{aligned} \tag{G.25}$$

Using integration by parts, we can show that the second term is zero:

$$\begin{aligned}&\int_{-\infty}^{+\infty}\mathrm{d}x_1\frac{\partial}{\partial x_1}(g_a(\mathbf{r}_1)g_b(\mathbf{r}_1)M_{12}) \\ &\quad = [g_a(\mathbf{r}_1)g_b(\mathbf{r}_1)M_{12}]_{-\infty}^{+\infty} - \int_{-\infty}^{+\infty}\mathrm{d}x_1\left(\frac{\partial}{\partial x_1}1\right)g_a(\mathbf{r}_1)g_b(\mathbf{r}_1)M_{12}.\end{aligned} \tag{G.26}$$

The first term on the right of the integration by parts is zero, because, in the limit of $x_1 \to \pm\infty$ the $g_a(\mathbf{r}_1)g_b(\mathbf{r}_1)$ part goes to zero faster than r_{12} goes to ∞. For $M_{12} = \frac{\mathbf{r_{12}}}{r_{12}}$, we have the same situation, since we are dealing with the integrand

$$\nabla_1 \cdot (g_a(\mathbf{r}_1)g_b(\mathbf{r}_1)\frac{\mathbf{r}_{12}}{r_{12}}) = \sum_{i=x,y,z}\frac{\partial}{\partial r_{1i}}(g_a(\mathbf{r}_1)g_b(\mathbf{r}_1)\frac{r_{12i}}{r_{12}}) \tag{G.27}$$

and $\frac{r_{12i}}{r_{12}} \to 1$ as $r_{1i} \to \pm\infty$. The second term in Eq. G.26 is always zero, since,

$$\frac{\partial}{\partial x_1}1 = 0. \tag{G.28}$$

Thus

$$\int \mathrm{d}\mathbf{r}_1\mathrm{d}\mathbf{r}_2\nabla_1(g_a(\mathbf{r}_1)g_b(\mathbf{r}_1)M_{12})g_c(\mathbf{r}_2)g_d(\mathbf{r}_2) = 0 \tag{G.29}$$

and

$$\nabla_P(\mathbf{ab}|M_{12}|\mathbf{cd}) = (\mathbf{ab}|\{\nabla_1 M_{12}\}|\mathbf{cd}). \tag{G.30}$$

We would like to show that

$$\nabla_P(\mathbf{ab}|(\nabla_1 f_{12})|\mathbf{c}) = (\mathbf{ab}|\{\nabla_1^2 f_{12}\}|\mathbf{c}). \tag{G.31}$$

Eq. G.22 applies for $M_{12} = \nabla_1 f_{12}$, thus:

$$(\nabla_1 f_{12}) \cdot (\nabla_1 g_a(\mathbf{r}_1)g_b(\mathbf{r}_1)) = -g_a(\mathbf{r}_1)g_b(\mathbf{r}_1)\{\nabla_1^2 f_{12}\} + \nabla_1 \cdot (g_a(\mathbf{r}_1)g_b(\mathbf{r}_1)\nabla_1 f_{12}). \tag{G.32}$$

Using integration by parts (as in Eq. G.26) we can show that the integral over the second term on the right is zero, since

$$\left[g_a(\mathbf{r}_1)g_b(\mathbf{r}_1)\frac{\partial}{\partial x_1}f_{12}\right]_{-\infty}^{+\infty} = 0 \tag{G.33}$$

because each of $g_a(\mathbf{r}_1)$, $g_b(\mathbf{r}_1)$ and ∇f_{12} go to zero as $x_1 \to \pm\infty$, and thus go to zero along any of x_1, y_1, z_1 as they tend towards $\pm\infty$). Thus, Eq. G.31 holds.

Substituting Eq. G.31 into Eq. G.18, we obtain

$$(\mathbf{ab}|(\nabla_1 f_{12}) \cdot \nabla_1|\mathbf{c}) = -\frac{\zeta_b}{\zeta}(\mathbf{ab}|\{\nabla_1^2 f_{12}\}|\mathbf{c}) + \nabla_R(\mathbf{ab}|(\nabla_1 f_{12})|\mathbf{c}). \tag{G.34}$$

Recognizing that Eq. G.30 applies for $M_{12} = f_{12}$,

$$\nabla_P(\mathbf{ab}|f_{12}|\mathbf{c}) = (\mathbf{ab}|(\nabla_1 f_{12})|\mathbf{c}) \tag{G.35}$$

we can see that:

$$\nabla_R(\mathbf{ab}|(\nabla_1 f_{12})|\mathbf{c}) = \nabla_P \cdot \nabla_R(\mathbf{ab}|f_{12}|\mathbf{c}). \tag{G.36}$$

We therefore obtain

$$(\mathbf{ab}|(\nabla_1 f_{12}) \cdot \nabla_1|\mathbf{c}) = -\frac{\zeta_b}{\zeta}(\mathbf{ab}|\{\nabla_1^2 f_{12}\}|\mathbf{c}) + \nabla_P \cdot \nabla_R(\mathbf{ab}|f_{12}|\mathbf{c}). \tag{G.37}$$

Finally, inserting the result of Eq. G.37 into Eq. G.7, we obtain

$$\begin{aligned}(\mathbf{ab}|[\hat{t}_1, f_{12}]|\mathbf{c}) &= \frac{\zeta_b}{\zeta}(\mathbf{ab}|\{\nabla_1^2 f_{12}\}|\mathbf{c}) - \nabla_P \cdot \nabla_R(\mathbf{ab}|f_{12}|\mathbf{c}) - \frac{1}{2}(\mathbf{ab}|\{\nabla_1^2 f_{12}\}|\mathbf{c}) \\ &= -\frac{1}{2}\frac{\zeta_a - \zeta_b}{\zeta_a + \zeta_b}(\mathbf{ab}|\{\nabla_1^2 f_{12}\}|\mathbf{c}) - \nabla_P \cdot \nabla_R(\mathbf{ab}|f_{12}|\mathbf{c}).\end{aligned} \tag{G.38}$$

Note that this result is different from the one published by May and Manby in Ref. 70, which differs by a factor of -1 (Eq. 53 in Ref. 70). May and Manby's equation is at odds with that of Klopper and Röhse in this regard, since given that May and Manby's equation is derived for $[\hat{t}_1, f_{12}]$, while Klopper and Röhse's is derived for $[r_{12}, \hat{t}_1]$ (Eq. 21 in Ref. 184), we would expect the opposite sign (as shown in Eqs. G.1 and G.2).

APPENDIX H

THE $Q_{\mathbf{p}}^{\mathbf{ab}}$ COEFFICIENTS

The evaluation of the FT (appendix G) and FT1-F (section 3.3.2) integrals involves the computation of terms like

$$\nabla_P \cdot \nabla_R (\mathbf{ab}|f_{12}|\mathbf{c}\cdots) \tag{H.1}$$

where

$$\mathbf{P} = \frac{\zeta_a}{\zeta}\mathbf{A} + \frac{\zeta_b}{\zeta}\mathbf{B} \tag{H.2}$$

$$\mathbf{R} = \mathbf{A} - \mathbf{B} \tag{H.3}$$

with $\zeta = \zeta_a + \zeta_b$, and $(\mathbf{ab}|f_{12}|\mathbf{c}\cdots)$ is a general integral over primitive Cartesian Gaussians, featuring the product $|\mathbf{ab}) = g_a(\mathbf{r}_1)g_b(\mathbf{r}_1)$.

The product with differential operators applied $\nabla_P \cdot \nabla_R|\mathbf{ab})$ can be expanded in terms of primitive Cartesian Gaussians on a single centre, just as a standard orbital product $|\mathbf{ab})$ can be expanded in the GPT (appendix B), i.e.

$$\nabla_P \cdot \nabla_R|\mathbf{ab}) = \sum_{\mathbf{p}} Q_{\mathbf{p}}^{\mathbf{ab}}|\mathbf{p}) \tag{H.4}$$

where the summation runs from $|\mathbf{p}| = 0$ to $|\mathbf{a}| + |\mathbf{b}| + 1$.

In Molpro [151,152], a subroutine exists which calculates the coefficients $Q_{\mathbf{p}}^{\mathbf{ab}}$ in terms of the GPT coefficients, $T_{\mathbf{p}}^{\mathbf{ab}}$ (Eq. B.2). The use of the $Q_{\mathbf{p}}^{\mathbf{ab}}$ coefficients for evaluating the FT integrals is referred to in Ref. 131, though the definition and derivation of the coefficients is not provided. The definition of the $Q_{\mathbf{p}}^{\mathbf{ab}}$ coefficients was later published in Ref. 150, which describes the work presented in chapter 3. The derivation of the $Q_{\mathbf{p}}^{\mathbf{ab}}$ coefficients in terms of the GPT coefficients (Eq. B.2) does not appear to have been published previously. For this reason, the derivation is presented here.

We start by expanding the operator $\nabla_P \cdot \nabla_R$ in terms of ∇_A and ∇_B, expressing $\mathbf{A}$ and $\mathbf{B}$

in terms of $\mathbf{P}$ and $\mathbf{R}$. Given the definitions of $\mathbf{P}$ and $\mathbf{R}$ given in equations H.2 and H.3,

$$\begin{aligned}\mathbf{P} &= \frac{\alpha}{\zeta}\mathbf{A} + \frac{\beta}{\zeta}\mathbf{B} \\ &= \frac{\alpha}{\zeta}(\mathbf{B}+\mathbf{R}) + \frac{\beta}{\zeta}\mathbf{B} \\ &= \frac{\alpha}{\zeta}\mathbf{A} + \frac{\beta}{\zeta}(\mathbf{A}-\mathbf{R}) \\ &= \mathbf{B} + \frac{\alpha}{\zeta}\mathbf{R} = \mathbf{A} - \frac{\beta}{\zeta}\mathbf{R}\end{aligned} \tag{H.5}$$

and thus

$$\mathbf{A} = \frac{\beta}{\zeta}\mathbf{R} + \mathbf{P} \tag{H.6}$$

and

$$\mathbf{B} = -\frac{\alpha}{\beta}\mathbf{R} + \mathbf{P}. \tag{H.7}$$

Now we can use the total differential [262, p.145] to express ∇_P and ∇_R in terms of ∇_A and ∇_B:

$$\begin{aligned}\frac{\partial}{\partial P_i} &= \frac{\partial A_i}{\partial P_i}\frac{\partial}{\partial A_i} + \frac{\partial B_i}{\partial P_i}\frac{\partial}{\partial B_i} \\ &= \frac{\partial}{\partial A_i} + \frac{\partial}{\partial B_i}\end{aligned} \tag{H.8}$$

and

$$\begin{aligned}\frac{\partial}{\partial R_i} &= \frac{\partial A_i}{\partial R_i}\frac{\partial}{\partial A_i} + \frac{\partial B_i}{\partial R_i}\frac{\partial}{\partial B_i} \\ &= \frac{\beta}{\zeta}\frac{\partial}{\partial A_i} - \frac{\alpha}{\zeta}\frac{\partial}{\partial B_i}.\end{aligned} \tag{H.9}$$

The operator $\nabla_P \cdot \nabla_R$ can therefore be expanded to give

$$\begin{aligned}\nabla_P \cdot \nabla_R |\mathbf{ab}) &= \sum_{i=x,y,z}\left(\frac{\partial}{\partial A_i} + \frac{\partial}{\partial B_i}\right)\left(\frac{\beta}{\zeta}\frac{\partial}{\partial A_i} - \frac{\alpha}{\zeta}\frac{\partial}{\partial B_i}\right)|\mathbf{ab}) \\ &= \sum_{i=x,y,z}\left\{\frac{\beta}{\zeta}\frac{\partial^2}{\partial A_i^2} - \frac{\alpha}{\zeta}\frac{\partial^2}{\partial B_i^2} + \left(\frac{\beta}{\zeta} - \frac{\alpha}{\zeta}\right)\frac{\partial}{\partial A_i}\frac{\partial}{\partial B_i}\right\}|\mathbf{ab}).\end{aligned} \tag{H.10}$$

At this point, it is useful to note some properties of the product of Gaussians $|\mathbf{ab})$.

The first derivatives of $|\mathbf{ab})$ are

$$\frac{\partial}{\partial A_i}|\mathbf{ab}) = 2\alpha|(\mathbf{a}+\mathbf{1}_i)\mathbf{b}) - a_i|(\mathbf{a}-\mathbf{1}_i)\mathbf{b}) \tag{H.11}$$

and

$$\frac{\partial}{\partial B_i}|\mathbf{ab}) = 2\beta|\mathbf{a}(\mathbf{b}+\mathbf{1}_i)) - b_i|\mathbf{a}(\mathbf{b}-\mathbf{1}_i)). \tag{H.12}$$

We can shift angular momentum between the Gaussian functions using

$$\begin{aligned}|(\mathbf{a}+\mathbf{2}_i)\mathbf{b}) &= (r_i - A_i)|(\mathbf{a}+\mathbf{1}_i)\mathbf{b}) \\ &= [(r_i - B_i) - (A_i - B_i)]\,|(\mathbf{a}+\mathbf{1}_i)\mathbf{b}) \\ &= |(\mathbf{a}+\mathbf{1}_i)(\mathbf{b}+\mathbf{1}_i)) - R_i|(\mathbf{a}+\mathbf{1}_i)\mathbf{b}),\end{aligned} \tag{H.13}$$

$$\begin{aligned}|\mathbf{a}(\mathbf{b}+\mathbf{2}_i)) &= |(\mathbf{a}+\mathbf{1}_i)(\mathbf{b}+\mathbf{1}_i)) + R_i|\mathbf{a}(\mathbf{b}+\mathbf{1}_i)) \\ &= |(\mathbf{a}+\mathbf{1}_i)(\mathbf{b}+\mathbf{1}_i)) + R_i|(\mathbf{a}+\mathbf{1}_i)\mathbf{b}) + R_i^2|\mathbf{ab}),\end{aligned} \tag{H.14}$$

and

$$|\mathbf{a}(\mathbf{b}+\mathbf{1}_i)) = |(\mathbf{a}+\mathbf{1}_i)\mathbf{b}) + R_i|\mathbf{ab}), \tag{H.15}$$

where $R_i = A_i - B_i$.

Now we evaluate each term in equation H.10:

$$\begin{aligned}&\frac{\beta}{\zeta}\frac{\partial^2}{\partial A_i^2}|\mathbf{ab}) = \frac{\beta}{\zeta}\left[2\alpha\left\{2\alpha|(\mathbf{a}+\mathbf{2}_i)\mathbf{b}) - (a_i+1)|\mathbf{ab})\right\}\right. \\ &\quad \left. - a_i\left\{2\alpha|\mathbf{ab}) - (a_i-1)|(\mathbf{a}-\mathbf{2}_i)\mathbf{b})\right\}\right] \\ &= \frac{\beta}{\zeta}\left[4\alpha^2|(\mathbf{a}+\mathbf{2}_i)\mathbf{b}) - 4\alpha a_i|\mathbf{ab}) - 2\alpha|\mathbf{ab}) + a_i(a_i-1)|(\mathbf{a}-\mathbf{2}_i)\mathbf{b})\right] \\ &= 4\frac{\alpha^2\beta}{\zeta}|(\mathbf{a}+\mathbf{2}_i)\mathbf{b}) - 4\frac{\alpha\beta}{\zeta}a_i|\mathbf{ab}) - 2\frac{\alpha\beta}{\zeta}|\mathbf{ab}) + \frac{\beta}{\zeta}a_i(a_i-1)|(\mathbf{a}-\mathbf{2}_i)\mathbf{b}) \\ &= 4\frac{\alpha^2\beta}{\zeta}|(\mathbf{a}+\mathbf{1}_i)(\mathbf{b}+\mathbf{1}_i)) - 4\frac{\alpha^2\beta}{\zeta}R_i|(\mathbf{a}+\mathbf{1}_i)\mathbf{b}) - 4\frac{\alpha\beta}{\zeta}a_i|\mathbf{ab}) \\ &\quad - 2\frac{\alpha\beta}{\zeta}|\mathbf{ab}) + \frac{\beta}{\zeta}a_i(a_i-1)|(\mathbf{a}-\mathbf{2}_i)\mathbf{b}),\end{aligned} \tag{H.16}$$

$$\begin{aligned}&\frac{\alpha}{\zeta}\frac{\partial^2}{\partial B_i^2}|\mathbf{ab}) \\ &= 4\frac{\alpha\beta^2}{\zeta}|\mathbf{a}(\mathbf{b}+\mathbf{2}_i)) - 4\frac{\alpha\beta}{\zeta}b_i|\mathbf{ab}) - 2\frac{\alpha\beta}{\zeta}|\mathbf{ab}) + \frac{\alpha}{\zeta}b_i(b_i-1)|\mathbf{a}(\mathbf{b}-\mathbf{2}_i)) \\ &= 4\frac{\alpha\beta^2}{\zeta}|(\mathbf{a}+\mathbf{1}_i)(\mathbf{b}+\mathbf{1}_i)) + 4\frac{\alpha\beta^2}{\zeta}R_i|(\mathbf{a}+\mathbf{1}_i)\mathbf{b}) + 4\frac{\alpha\beta^2}{\zeta}R_i^2|\mathbf{ab}) \\ &\quad - 4\frac{\alpha\beta}{\zeta}b_i|\mathbf{ab}) - 2\frac{\alpha\beta}{\zeta}|\mathbf{ab}) + \frac{\alpha}{\zeta}b_i(b_i-1)|\mathbf{a}(\mathbf{b}-\mathbf{2}_i)),\end{aligned} \tag{H.17}$$

$$
\begin{aligned}
&\frac{1}{\zeta}(\beta-\alpha)\frac{\partial}{\partial A_i}\frac{\partial}{\partial B_i}|\mathbf{ab}) \\
&= -\frac{1}{\zeta}(\alpha-\beta)\left[2\beta\{2\alpha|(\mathbf{a}+\mathbf{1}_i)(\mathbf{b}+\mathbf{1}_i)) - a_i|(\mathbf{a}-\mathbf{1}_i)(\mathbf{b}+\mathbf{1}_i))\}\right. \\
&\quad \left.-b_i\{2\alpha|(\mathbf{a}+\mathbf{1}_i)(\mathbf{b}-\mathbf{1}_i)) - a_i|(\mathbf{a}-\mathbf{1}_i)(\mathbf{b}-\mathbf{1}_i))\}\right] \\
&= -\frac{1}{\zeta}(\alpha-\beta)\left[4\alpha\beta|(\mathbf{a}+\mathbf{1}_i)(\mathbf{b}+\mathbf{1}_i)) - 2\beta a_i|(\mathbf{a}-\mathbf{1}_i)(\mathbf{b}+\mathbf{1}_i))\right. \\
&\quad \left.-2\alpha b_i|(\mathbf{a}+\mathbf{1}_i)(\mathbf{b}-\mathbf{1}_i)) + a_i b_i|(\mathbf{a}-\mathbf{1}_i)(\mathbf{b}-\mathbf{1}_i))\right] \\
&= -\frac{1}{\zeta}(\alpha-\beta)\left[4\alpha\beta|(\mathbf{a}+\mathbf{1}_i)(\mathbf{b}+\mathbf{1}_i)) - 2\beta a_i|\mathbf{ab}) - 2\beta a_i R_i|(\mathbf{a}-\mathbf{1}_i)\mathbf{b})\right. \\
&\quad \left.-2\alpha b_i|(\mathbf{a}+\mathbf{1}_i)(\mathbf{b}-\mathbf{1}_i)) + a_i b_i|(\mathbf{a}-\mathbf{1}_i)(\mathbf{b}-\mathbf{1}_i))\right].
\end{aligned}
\tag{H.18}
$$

We can see that the $|(\mathbf{a}+\mathbf{1}_i)(\mathbf{b}+\mathbf{1}_i))$ terms cancel in equation H.10, since

$$
\frac{4}{\zeta}\{\alpha^2\beta - \beta^2\alpha + (\beta-\alpha)\alpha\beta\}|(\mathbf{a}+\mathbf{1}_i)(\mathbf{b}+\mathbf{1}_i)) = 0. \tag{H.19}
$$

We can also see that the $2\frac{\alpha\beta}{\zeta}|\mathbf{ab})$ terms of equations H.16 and H.17 will cancel in equation H.10.

Substituting the remaining terms into equation H.10, we obtain

$$
\begin{aligned}
\nabla_P \cdot \nabla_R|\mathbf{ab}) = \sum_{i=x,y,z} & -4\frac{\alpha^2\beta}{\zeta}R_i|(\mathbf{a}+\mathbf{1}_i)\mathbf{b}) - 4\frac{\alpha\beta}{\zeta}a_i|\mathbf{ab}) \\
& + \frac{\beta}{\zeta}a_i(a_i-1)|(\mathbf{a}-\mathbf{2}_i)\mathbf{b}) \\
& - 4\frac{\alpha\beta^2}{\zeta}R_i|(\mathbf{a}+\mathbf{1}_i)\mathbf{b}) - 4\frac{\alpha\beta^2}{\zeta}R_i^2|\mathbf{ab}) \\
& + 4\frac{\alpha\beta}{\zeta}b_i|\mathbf{ab}) - \frac{\alpha}{\zeta}b_i(b_i-1)|\mathbf{a}(\mathbf{b}-\mathbf{2}_i)) \\
& - \frac{1}{\zeta}(\alpha-\beta)\left[-2\beta a_i|\mathbf{ab}) - 2\beta a_i R_i|(\mathbf{a}-\mathbf{1}_i)\mathbf{b})\right. \\
& \left.-2\alpha b_i|(\mathbf{a}+\mathbf{1}_i)(\mathbf{b}-\mathbf{1}_i)) + a_i b_i|(\mathbf{a}-\mathbf{1}_i)(\mathbf{b}-\mathbf{1}_i))\right] \\
= \sum_{i=x,y,z} & -4\alpha\beta R_i|(\mathbf{a}+\mathbf{1}_i)\mathbf{b}) - 4\beta\xi R_i^2|\mathbf{ab}) \\
& + \frac{\beta}{\zeta}a_i(a_i-1)|(\mathbf{a}-\mathbf{2}_i)\mathbf{b}) - \frac{\alpha}{\zeta}b_i(b_i-1)|\mathbf{a}(\mathbf{b}-\mathbf{2}_i)) \\
& - 2\beta a_i|\mathbf{ab}) + 2\beta\eta a_i R_i|(\mathbf{a}-\mathbf{1}_i)\mathbf{b}) - \eta a_i b_i|(\mathbf{a}-\mathbf{1}_i)(\mathbf{b}-\mathbf{1}_i)) \\
& + 4\frac{\alpha\beta}{\zeta}b_i|\mathbf{ab}) + 2\alpha\eta b_i|(\mathbf{a}+\mathbf{1}_i)(\mathbf{b}-\mathbf{1}_i))
\end{aligned}
\tag{H.20}
$$

where

$$
\xi = \frac{\alpha\beta}{\zeta}, \tag{H.21}
$$

$$
\eta = \frac{\alpha-\beta}{\zeta}. \tag{H.22}
$$

Since

$$\begin{aligned}
&4\frac{\alpha\beta}{\zeta}b_i|\mathbf{ab}) + 2\frac{\alpha^2}{\zeta}b_i|(\mathbf{a}+\mathbf{1}_i)(\mathbf{b}-\mathbf{1}_i)) - 2\frac{\alpha\beta}{\zeta}b_i|(\mathbf{a}+\mathbf{1}_i)(\mathbf{b}-\mathbf{1}_i)) \\
&\quad = 4\frac{\alpha\beta}{\zeta}b_i|(\mathbf{a}+\mathbf{1}_i)(\mathbf{b}-\mathbf{1}_i)) + 2\frac{\alpha^2}{\zeta}b_i|(\mathbf{a}+\mathbf{1}_i)(\mathbf{b}-\mathbf{1}_i)) \\
&\qquad - 2\frac{\alpha\beta}{\zeta}b_i|(\mathbf{a}+\mathbf{1}_i)(\mathbf{b}-\mathbf{1}_i)) + 4\frac{\alpha\beta}{\zeta}b_iR_i|\mathbf{a}(\mathbf{b}-\mathbf{1}_i)) \\
&\quad = 2\alpha b_i|(\mathbf{a}+\mathbf{1}_i)(\mathbf{b}-\mathbf{1}_i)) + 4\xi b_iR_i|\mathbf{a}(\mathbf{b}-\mathbf{1}_i))
\end{aligned} \tag{H.23}$$

and

$$-4\beta\xi R_i^2|\mathbf{ab}) - 2\beta a_i|\mathbf{ab}) = -2\beta(2\xi R_i^2 + a_i)|\mathbf{ab}) \tag{H.24}$$

we obtain

$$\begin{aligned}
&\nabla_P \cdot \nabla_R|\mathbf{ab}) \\
&= \sum_{i=x,y,z} -2\beta(2\xi R_i^2 + a_i)|\mathbf{ab}) - 4\alpha\beta R_i|(\mathbf{a}+\mathbf{1}_i)\mathbf{b}) + 2\alpha b_i|(\mathbf{a}+\mathbf{1}_i)(\mathbf{b}-\mathbf{1}_i)) \\
&\quad + 2\beta\eta a_iR_i|(\mathbf{a}-\mathbf{1}_i)\mathbf{b}) + 4\xi b_iR_i|\mathbf{a}(\mathbf{b}-\mathbf{1}_i)) - \eta a_ib_i|(\mathbf{a}-\mathbf{1}_i)(\mathbf{b}-\mathbf{1}_i)) \\
&\quad + \frac{\beta}{\zeta}a_i(a_i-1)|(\mathbf{a}-\mathbf{2}_i)\mathbf{b}) - \frac{\alpha}{\zeta}b_i(b_i-1)|\mathbf{a}(\mathbf{b}-\mathbf{2}_i)).
\end{aligned} \tag{H.25}$$

Finally, using the GPT (appendix B)

$$|\mathbf{ab}) = \sum_{\mathbf{p}} T_{\mathbf{p}}^{\mathbf{ab}}|\mathbf{p}) \tag{H.26}$$

with the summation over $\mathbf{p}$ running from $|\mathbf{p}| = 0$ to $|\mathbf{p}| = |\mathbf{a}| + |\mathbf{b}|$, we find that

$$\begin{aligned}
\nabla_P \cdot \nabla_R|\mathbf{ab}) = \sum_{|\mathbf{p}|=0}^{|\mathbf{a}|+|\mathbf{b}|+1} \sum_{i=x,y,z} \Big(& -2\beta(2\xi R_i^2 + a_i)T_{\mathbf{p}}^{\mathbf{ab}} \\
& - 4\alpha\beta R_i T_{\mathbf{p}}^{\mathbf{a}+\mathbf{1}_i,\mathbf{b}} + 2\alpha b_i T_{\mathbf{p}}^{\mathbf{a}+\mathbf{1}_i,\mathbf{b}-\mathbf{1}_i} \\
& + 2\beta\eta a_i R_i T_{\mathbf{p}}^{\mathbf{a}-\mathbf{1}_i,\mathbf{b}} + 4\xi b_i R_i T_{\mathbf{p}}^{\mathbf{a},\mathbf{b}-\mathbf{1}_i} \\
& - \eta a_i b_i T_{\mathbf{p}}^{\mathbf{a}-\mathbf{1}_i,\mathbf{b}-\mathbf{1}_i} + \frac{\beta}{\zeta}a_i(a_i-1)T_{\mathbf{p}}^{\mathbf{a}-\mathbf{2}_i,\mathbf{b}} \\
& - \frac{\alpha}{\zeta}b_i(b_i-1)T_{\mathbf{p}}^{\mathbf{a},\mathbf{b}-\mathbf{2}_i} \Big) |\mathbf{p})
\end{aligned} \tag{H.27}$$

and thus

$$\begin{aligned}
Q_{\mathbf{p}}^{\mathbf{ab}} = \sum_{i=x,y,z} \Big(& -2\beta(2\xi R_i^2 + a_i)T_{\mathbf{p}}^{\mathbf{ab}} - 4\alpha\beta R_i T_{\mathbf{p}}^{\mathbf{a}+\mathbf{1}_i,\mathbf{b}} + 2\alpha b_i T_{\mathbf{p}}^{\mathbf{a}+\mathbf{1}_i,\mathbf{b}-\mathbf{1}_i} \\
& + 2\beta\eta a_i R_i T_{\mathbf{p}}^{\mathbf{a}-\mathbf{1}_i,\mathbf{b}} + 4\xi b_i R_i T_{\mathbf{p}}^{\mathbf{a},\mathbf{b}-\mathbf{1}_i} - \eta a_i b_i T_{\mathbf{p}}^{\mathbf{a}-\mathbf{1}_i,\mathbf{b}-\mathbf{1}_i} \\
& + \frac{\beta}{\zeta}a_i(a_i-1)T_{\mathbf{p}}^{\mathbf{a}-\mathbf{2}_i,\mathbf{b}} - \frac{\alpha}{\zeta}b_i(b_i-1)T_{\mathbf{p}}^{\mathbf{a},\mathbf{b}-\mathbf{2}_i} \Big).
\end{aligned} \tag{H.28}$$

In Eq. H.27, the summation over $\mathbf{p}$ goes from $|\mathbf{p}| = 0$ to $|\mathbf{p}| = |\mathbf{a}|+|\mathbf{b}|+1$. Where $|\mathbf{p}| > |\mathbf{a}| + |\mathbf{b}|$,

$T_{\mathbf{p}}^{\mathbf{ab}} = 0$ (appendix B). Terms which would feature $T_{\mathbf{p}}^{\mathbf{a}',\mathbf{b}'}$ with $\mathbf{a}'$ or $\mathbf{b}'$ having negative components are set to zero by multiplication by a_i, $a_i(a_i - 1)$ or b_i, $b_i(b_i - 1)$. For example if $a_i = 0$, terms featuring $T_{\mathbf{p}}^{\mathbf{a}-\mathbf{2}_i,\mathbf{b}}$ (i.e. $\mathbf{a}' = \mathbf{a} - \mathbf{2}_i$) are zero, since they also feature $a_i(a_i - 1) = 0$. Equation H.27 is identical to the equation used in Molpro's subroutine [151, 152].

APPENDIX I

MOLECULAR DATA

H_2O

Used in results presented in chapter 3 (see Fig. 3.2) and chapter 4 (sections 4.4.3 and 4.4.5). Experimental geometry data obtained from NIST CCBDB [263], based on Ref. 264. The Z-matrix and associated parameters used in Molpro [151, 152] input files are as follows:

```
O
H  1  r
H  1  r  2  θ
```

$$r = 0.9578\,\text{Å} \qquad \theta = 104.4776^\circ$$

For the electronic structure calculations described in section 4.4.4, an alternative geometry in XYZ format (produced by a local-MP2 geometry optimization with aug-cc-pVDZ basis set) with $r = 0.9664\,\text{Å}$ and $\theta = 103.8^\circ$ was used:

```
3

O  0.00000000   0.00000000   -0.06669620
H  0.00000000   0.76075414    0.52934657
H  0.00000000  -0.76075414    0.52934657
```

HF

Used in results presented in chapter 3 (see Fig. 3.3). Experimental geometry data obtained from NIST Diatomic Spectral Database [265]. The Z-matrix and associated parameters used in Molpro input files are as follows:

```
H
F  1  r
```

$$r = 0.916\,809\,\text{Å}$$

Benzene, C_6H_6

Geometry used to test integral evaluation routines in chapter 4 (section 4.4), obtained from Molpro [151,152] test scripts. The XYZ geometry specification used is as follows, with distances measured in angstroms:

```
12

C    0.000   1.396   0.000
C    1.209   0.698   0.000
C    1.209  -0.698   0.000
C    0.000  -1.396   0.000
C   -1.209  -0.698   0.000
C   -1.209   0.698   0.000
H    0.000   2.479   0.000
H    2.147   1.240   0.000
H    2.147  -1.240   0.000
H    0.000  -2.479   0.000
H   -2.147  -1.240   0.000
H   -2.147   1.240   0.000
```

Glycine, $NH_3{}^+CH_2COO^-$

Geometry used to test integral evaluation routines in chapter 4 (section 4.4), obtained from Molpro [151,152] test scripts. The structure is for the zwitterionic form of glycine, i.e. $NH_3{}^+CH_2COO^-$. The XYZ geometry specification used is as follows, with distances measured in angstroms:

```
10

O    1.081302   1.129735   1.195158
O   -0.967942   1.693585   0.543683
N    2.060859   1.075277  -1.315237
C    0.249391   1.494424   0.336070
C    0.760991   1.733681  -1.081882
H    2.396597   1.201189  -2.305828
H    2.790965   1.427758   -.669398
H    1.985133   0.067145  -1.148141
H    0.883860   2.805965  -1.234913
H    0.041439   1.369111  -1.813528
```

Naphthalene, $C_{10}H_8$

Geometry used to test integral evaluation routines in chapter 4 (section 4.4). The structure used has C-C bond lengths of 1.42 Å, C-H bond lengths of 1.00 Å and all bond angles equal to

120°. The XYZ geometry specification used is as follows, with distances measured in angstroms:

```
18

H          0.0          2.42   0.0
H     2.45951214675     2.42   0.0
H    -2.09578147716     1.21   0.0
H     4.55529362391    -1.21   0.0
C     1.22975607337     0.71   0.0
C    -1.22975607337     0.71   0.0
C     1.22975607337    -0.71   0.0
C          0.0          1.42   0.0
C    -1.22975607337    -0.71   0.0
C     3.68926822012     0.71   0.0
C     3.68926822012    -0.71   0.0
C     2.45951214675     1.42   0.0
C          0.0         -1.42   0.0
H     4.55529362391     1.21   0.0
C     2.45951214675    -1.42   0.0
H     2.45951214675    -2.42   0.0
H    -2.09578147716    -1.21   0.0
H          0.0         -2.42   0.0
```

BIBLIOGRAPHY

[1] S. Obara and A. Saika, J. Chem. Phys. **84**, 3963 (1986).

[2] T. Helgaker, W. Klopper, and D. P. Tew, Mol. Phys. **106**, 2107 (2008).

[3] E. Schrödinger, *Collected Papers on Wave Mechanics: Third Edition*, American Mathematical Soc., 1982.

[4] E. Schrödinger, Phys. Rev. **28**, 1049 (1926).

[5] P. Atkins and R. Friedman, *Molecular Quantum Mechanics*, Oxford University Press, 4th edition, 2005.

[6] D. A. McQuarrie, *Quantum Chemistry*, University Science Books, 2nd edition, 2008.

[7] A. Szabo and N. S. Ostlund, *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory*, Dover, 1st rev. edition, 1996.

[8] H. A. Bethe and E. E. Salpeter, *Quantum mechanics of one- and two-electron atoms*, Springer, 1957.

[9] P. A. M. Dirac, Math. Proc. Cambridge **35**, 416 (1939).

[10] T. Helgaker, P. Jørgensen, and J. Olsen, *Molecular Electronic-Structure Theory*, Wiley, 2000.

[11] J. C. Slater, Phys. Rev. **34**, 1293 (1929).

[12] F. Jensen, *Introduction to Computational Chemistry*, Wiley, 2nd edition, 2007.

[13] F. Seitz, *The Modern Theory of Solids*, McGraw-Hill Book Company, Incorporated, 1940.

[14] T. Koopmans, Physica **1**, 104 (1934).

[15] D. R. Hartree, Math. Proc. Cambridge **24**, 89 (1928).

[16] D. R. Hartree, Math. Proc. Cambridge **24**, 111 (1928).

[17] D. R. Hartree and W. Hartree, Proc. R. Soc. Lon. Ser. A **150**, 9 (1935).

[18] V. Fock, Z. Phys. **61**, 126 (1930).

[19] V. Fock, Z. Phys. **62**, 795 (1930).

[20] J. C. Slater, Phys. Rev. **35**, 210 (1930).

[21] C. C. J. Roothaan, Rev. Mod. Phys. **23**, 69 (1951).

[22] G. G. Hall, Proc. R. Soc. Lon. Ser. A **205**, 541 (1951).

[23] P. Pulay, Chem. Phys. Lett. **73**, 393 (1980).

[24] P. Pulay, J. Comput. Chem. **3**, 556 (1982).

[25] C. Hättig, W. Klopper, A. Köhn, and D. P. Tew, Chem. Rev. **112**, 4 (2012).

[26] T. Kato, Commun. Pure Appl. Math. **10**, 151 (1957).

[27] R. T. Pack and W. B. Brown, J. Chem. Phys. **45**, 556 (1966).

[28] D. P. Tew, J. Chem. Phys. **129**, 014104 (2008).

[29] P.-O. Löwdin, Phys. Rev. **97**, 1474 (1955).

[30] B. Roos, Chem. Phys. Lett. **15**, 153 (1972).

[31] I. Shavitt, Mol. Phys. **94**, 3 (1998).

[32] G. H. Booth, A. J. W. Thom, and A. Alavi, J. Chem. Phys. **131**, 054106 (2009).

[33] J. A. Pople, J. S. Binkley, and R. Seeger, Int. J. Quantum Chem. **10**, 1 (1976).

[34] I. Shavitt and R. J. Bartlett, *Many-Body Methods in Chemistry and Physics: MBPT and Coupled-Cluster Theory*, Cambridge University Press, 2009.

[35] R. J. Bartlett, Annu. Rev. Phys. Chem. **32**, 359 (1981).

[36] F. Coester, Nucl. Phys. **7**, 421 (1958).

[37] F. Coester and H. Kümmel, Nucl. Phys. **17**, 477 (1960).

[38] J. Čížek, J. Chem. Phys. **45**, 4256 (1966).

[39] R. J. Bartlett, J. Phys. Chem. **93**, 1697 (1989).

[40] T. D. Crawford and H. F. Schaefer, An Introduction to Coupled Cluster Theory for Computational Chemists, in *Reviews in Computational Chemistry*, volume 14, pages 33–136, John Wiley & Sons, Inc., Hoboken, NJ, USA, 2000.

[41] K. Raghavachari, G. W. Trucks, J. A. Pople, and M. Head-Gordon, Chem. Phys. Lett. **157**, 479 (1989).

[42] J. D. Watts, J. Gauss, and R. J. Bartlett, J. Chem. Phys. **98**, 8718 (1993).

[43] D. I. Lyakh, M. Musiał, V. F. Lotrich, and R. J. Bartlett, Chem. Rev. **112**, 182 (2012).

[44] R. J. Bartlett and M. Musiał, Rev. Mod. Phys. **79**, 291 (2007).

[45] C. Møller and M. Plesset, Phys. Rev. **46**, 618 (1934).

[46] D. Cremer, WIREs Comput. Mol. Sci. **1**, 509 (2011).

[47] E. A. Hylleraas, Z. Phys. **65**, 209 (1930).

[48] H. Hettema, *Quantum chemistry: classic scientific papers*, World Scientific series in 20th century chemistry, World Scientific, 2001.

[49] O. Sinanoğlu, J. Chem. Phys. **36**, 706 (1962).

[50] W. Klopper and W. Kutzelnigg, Chem. Phys. Lett. **134**, 17 (1987).

[51] T. H. Dunning, J. Chem. Phys. **90**, 1007 (1989).

[52] D. E. Woon and T. H. Dunning, J. Chem. Phys. **98**, 1358 (1993).

[53] A. K. Wilson, T. van Mourik, and T. H. Dunning Jr., J. Mol. Struct. THEOCHEM **388**, 339 (1996).

[54] T. Helgaker, W. Klopper, H. Koch, and J. Noga, J. Chem. Phys. **106**, 9639 (1997).

[55] A. Halkier, T. Helgaker, P. Jørgensen, W. Klopper, H. Koch, J. Olsen, and A. K. Wilson, Chem. Phys. Lett. **286**, 243 (1998).

[56] W. Klopper, F. R. Manby, S. Ten-No, and E. F. Valeev, Int. Rev. Phys. Chem. **25**, 427 (2006).

[57] H.-J. Werner, F. R. Manby, and P. J. Knowles, J. Chem. Phys. **118**, 8149 (2003).

[58] T. L. Gilbert, Rev. Mod. Phys. **35**, 491 (1963).

[59] E. A. Hylleraas, Z. Phys. **54**, 347 (1929).

[60] J. C. Slater, Phys. Rev. **31**, 333 (1928).

[61] T. Helgaker and W. Klopper, Theor. Chem. Acc. **103**, 180 (2000).

[62] S. F. Boys and N. C. Handy, Proc. R. Soc. Lon. Ser. A **310**, 43 (1969).

[63] K. Szalewicz, B. Jeziorski, H. J. Monkhorst, and J. G. Zabolitzky, Chem. Phys. Lett. **91**, 169 (1982).

[64] J. S. Sims and S. Hagstrom, Phys. Rev. A **4**, 908 (1971).

[65] D. C. Clary and N. C. Handy, Phys. Rev. A **14**, 1607 (1976).

[66] W. Kutzelnigg, Theor. Chim. Acta **68**, 445 (1985).

[67] W. Klopper, Chem. Phys. Lett. **186**, 583 (1991).

[68] W. Kutzelnigg and W. Klopper, J. Chem. Phys. **94**, 1985 (1991).

[69] A. J. May, E. Valeev, R. Polly, and F. R. Manby, Phys. Chem. Chem. Phys. **7**, 2710 (2005).

[70] A. J. May and F. R. Manby, J. Chem. Phys. **121**, 4479 (2004).

[71] H.-J. Werner, T. B. Adler, and F. R. Manby, J. Chem. Phys. **126**, 164102 (2007).

[72] R. A. Kendall, T. H. Dunning, and R. J. Harrison, J. Chem. Phys. **96**, 6796 (1992).

[73] D. P. Tew, W. Klopper, C. Neiss, and C. Hättig, Phys. Chem. Chem. Phys. **9**, 1921 (2007).

[74] D. P. Tew, W. Klopper, C. Neiss, and C. Hättig, Phys. Chem. Chem. Phys. **10**, 6325 (2008).

[75] R. M. Martin, *Electronic Structure: Basic Theory and Practical Methods*, Cambridge University Press, 2004.

[76] A. D. Becke, J. Chem. Phys. **140**, 18A301 (2014).

[77] P. Hohenberg and W. Kohn, Phys. Rev. **136**, B864 (1964).

[78] W. Kohn and L. J. Sham, Phys. Rev. **140**, A1133 (1965).

[79] J. A. Pople, J. Chem. Phys. **43**, S229 (1965).

[80] G. Knizia, T. B. Adler, and H.-J. Werner, J. Chem. Phys. **130**, 054104 (2009).

[81] H.-J. Werner and F. R. Manby, J. Chem. Phys. **124**, 054114 (2006).

[82] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, 2009.

[83] P. Pulay, Chem. Phys. Lett. **100**, 151 (1983).

[84] S. Saebø and P. Pulay, Chem. Phys. Lett. **113**, 13 (1985).

[85] P. Pulay and S. Saebø, Theor. Chim. Acta **69**, 357 (1986).

[86] A. S. P. Gomes and C. R. Jacob, Annu. Rep. Prog. Chem., Sect. C: Phys. Chem. **108**, 222 (2012).

[87] J. C. Slater, Phys. Rev. **36**, 57 (1930).

[88] S. F. Boys, Proc. R. Soc. Lon. Ser. A **200**, 542 (1950).

[89] A. J. May, *Density Fitting in Explicitly Correlated Electronic Structure Theory*, Ph. D., University of Bristol, Bristol, 2006.

[90] J. G. Hill, Int. J. Quantum Chem. **113**, 21 (2012).

[91] W. J. Hehre, R. F. Stewart, and J. A. Pople, J. Chem. Phys. **51**, 2657 (1969).

[92] H. Taketa, S. Huzinaga, and K. O-ohata, J. Phys. Soc. Jpn. **21**, 2313 (1966).

[93] J. A. Pople and W. J. Hehre, J. Comput. Phys. **27**, 161 (1978).

[94] M. Dupuis, J. Rys, and H. F. King, J. Chem. Phys. **65**, 111 (1976).

[95] H. F. King and M. Dupuis, J. Comput. Phys. **21**, 144 (1976).

[96] J. Rys, M. Dupuis, and H. F. King, J. Comput. Chem. **4**, 154 (1983).

[97] L. E. McMurchie and E. R. Davidson, J. Comput. Phys. **26**, 218 (1978).

[98] P. M. W. Gill, Molecular Integrals Over Gaussian Basis Functions, in *Advances in Quantum Chemistry*, edited by J. R. Sabin and M. C. Zerner, volume 25, pages 141–205, Academic Press, 1994.

[99] S. Reine, T. Helgaker, and R. Lindh, WIREs Comput. Mol. Sci. **2**, 290 (2012).

[100] H. B. Schlegel, J. Chem. Phys. **77**, 3676 (1982).

[101] M. Head-Gordon and J. A. Pople, J. Chem. Phys. **89**, 5777 (1988).

[102] B. G. Johnson, P. M. W. Gill, and J. A. Pople, Chem. Phys. Lett. **206**, 229 (1993).

[103] L. L. Shipman and R. E. Christoffersen, Comput. Phys. Commun. **2**, 201 (1971).

[104] S. T. Elbert and E. R. Davidson, J. Comput. Phys. **16**, 391 (1974).

[105] F. E. Harris, Int. J. Quantum Chem. **23**, 1469 (1983).

[106] P. M. W. Gill, B. G. Johnson, and J. A. Pople, Int. J. Quantum Chem. **40**, 745 (1991).

[107] D. Hegarty and G. Van Der Velde, Int. J. Quantum Chem. **23**, 1135 (1983).

[108] P. M. W. Gill, M. Head-Gordon, and J. A. Pople, Int. J. Quantum Chem. **36**, 269 (1989).

[109] P. M. W. Gill, M. Head-Gordon, and J. A. Pople, J. Phys. Chem. **94**, 5564 (1990).

[110] P. M. W. Gill and J. A. Pople, Int. J. Quantum Chem. **40**, 753 (1991).

[111] C. A. White and M. Head-Gordon, J. Chem. Phys. **104**, 2620 (1996).

[112] Y. Miao and K. M. Merz, J. Chem. Theory Comput. **9**, 965 (2013).

[113] A. V. Titov, I. S. Ufimtsev, N. Luehr, and T. J. Martinez, J. Chem. Theory Comput. **9**, 213 (2013).

[114] K. Yasuda and H. Maruoka, Int. J. Quantum Chem. **114**, 543 (2014).

[115] Y. Miao and K. M. Merz, J. Chem. Theory Comput. **11**, 1449 (2015).

[116] O. Treutler and R. Ahlrichs, J. Chem. Phys. **102**, 346 (1995).

[117] S. Ten-no, J. Chem. Phys. **121**, 117 (2004).

[118] Y.-y. Ohnishi, K. Ishimura, and S. Ten-no, Int. J. Quantum Chem. **115**, 333 (2014).

[119] D. F. Brailsford and G. G. Hall, Int. J. Quantum Chem. **5**, 657 (1971).

[120] IEEE Computer Society, *IEEE Standard for Floating-Point Arithmetic (IEEE Std 754-2008)*, 2008, https://standards.ieee.org/findstds/standard/754-2008.html.

[121] R. Hyde, *Write Great Code, Vol. 1: Understanding the Machine*, No Starch Press, 2004.

[122] S. Oliveira and D. E. Stewart, *Writing Scientific Software: A Guide to Good Style*, Cambridge University Press, Cambridge ; New York, 1st edition, 2006.

[123] J. Almlöf, K. Faegri, and K. Korsell, J. Comput. Chem. **3**, 385 (1982).

[124] M. Schütz and R. Lindh, Theor. Chim. Acta **95**, 13 (1997).

[125] M. Schütz, R. Lindh, and H.-J. Werner, Mol. Phys. **96**, 719 (1999).

[126] H. Koch, A. S. de Merás, T. Helgaker, and O. Christiansen, J. Chem. Phys. **104**, 4157 (1996).

[127] I. S. Ufimtsev and T. J. Martínez, J. Chem. Theory Comput. **4**, 222 (2008).

[128] I. S. Ufimtsev and T. J. Martinez, J. Chem. Theory Comput. **5**, 1004 (2009).

[129] M. Häser and R. Ahlrichs, J. Comput. Chem. **10**, 104 (1989).

[130] D. S. Lambrecht and C. Ochsenfeld, J. Chem. Phys. **123**, 184101 (2005).

[131] F. R. Manby, J. Chem. Phys. **119**, 4607 (2003).

[132] N. H. F. Beebe and J. Linderberg, Int. J. Quantum Chem. **12**, 683 (1977).

[133] T. B. Pedersen, F. Aquilante, and R. Lindh, Theor. Chem. Acc. **124**, 1 (2009).

[134] I. Røeggen and T. Johansen, J. Chem. Phys. **128**, 194107 (2008).

[135] H. Koch, A. S. de Merás, and T. B. Pedersen, J. Chem. Phys. **118**, 9481 (2003).

[136] B. I. Dunlap, J. W. D. Connolly, and J. R. Sabin, J. Chem. Phys. **71**, 3396 (1979).

[137] S. Boys and I. Shavitt, University of Wisconsin, National Technical Information Service , Document code AD212985 (1959).

[138] E. Baerends, D. Ellis, and P. Ros, Chem. Phys. **2**, 41 (1973).

[139] J. L. Whitten, J. Chem. Phys. **58**, 4496 (1973).

[140] O. Vahtras, J. Almlöf, and M. Feyereisen, Chem. Phys. Lett. **213**, 514 (1993).

[141] S. Ten-no and S. Iwata, Chem. Phys. Lett. **240**, 578 (1995).

[142] F. Weigend, Phys. Chem. Chem. Phys. **4**, 4285 (2002).

[143] R. Polly, H.-J. Werner, F. R. Manby, and P. J. Knowles, Mol. Phys. **102**, 2311 (2004).

[144] F. Weigend and M. Häser, Theor. Chem. Acc. **97**, 331 (1997).

[145] F. Weigend, M. Häser, H. Patzelt, and R. Ahlrichs, Chem. Phys. Lett. **294**, 143 (1998).

[146] A. P. Rendell and T. J. Lee, J. Chem. Phys. **101**, 400 (1994).

[147] C. Hättig and F. Weigend, J. Chem. Phys. **113**, 5154 (2000).

[148] B. I. Dunlap, Phys. Chem. Chem. Phys. **2**, 2113 (2000).

[149] B. I. Dunlap, J. Mol. Struct. THEOCHEM **529**, 37 (2000).

[150] J. C. Womack and F. R. Manby, J. Chem. Phys. **140**, 044118 (2014).

[151] H.-J. Werner, P. J. Knowles, G. Knizia, F. R. Manby, M. Schütz, et al., *MOLPRO, version 2012.1, a package of ab initio programs*, Cardiff, UK, 2012, See http://www.molpro.net.

[152] H.-J. Werner, P. J. Knowles, G. Knizia, F. R. Manby, and M. Schütz, WIREs Comput. Mol. Sci. **2**, 242 (2012).

[153] S. F. Boys, Proc. R. Soc. Lon. Ser. A **258**, 402 (1960).

[154] K. Singer, Proc. R. Soc. Lon. Ser. A **258**, 412 (1960).

[155] K. Szalewicz, B. Jeziorski, H. J. Monkhorst, and J. G. Zabolitzky, J. Chem. Phys. **78**, 1420 (1983).

[156] S. Ten-no, Chem. Phys. Lett. **330**, 175 (2000).

[157] M. M. Mehine, S. A. Losilla, and D. Sundholm, Mol. Phys. **111**, 2536 (2013).

[158] *TURBOMOLE, V6.6 2014*, University of Karlsruhe and Forschungszentrum Karlsruhe GmbH, 1989-2007, TURBOMOLE GmbH since 2007, 2014, See http://www.turbomole.com.

[159] C. C. Samson, W. Klopper, and T. Helgaker, Comput. Phys. Commun. **149**, 1 (2002).

[160] S. Ten-no, Chem. Phys. Lett. **398**, 56 (2004).

[161] E. F. Valeev, J. Chem. Phys. **125**, 244106 (2006).

[162] D. P. Tew and W. Klopper, J. Chem. Phys. **123**, 074101 (2005).

[163] F. R. Manby, H.-J. Werner, T. B. Adler, and A. J. May, J. Chem. Phys. **124**, 094103 (2006).

[164] H.-J. Werner, J. Chem. Phys. **129**, 101103 (2008).

[165] T. B. Adler, H.-J. Werner, and F. R. Manby, J. Chem. Phys. **130**, 054106 (2009).

[166] T. B. Adler and H.-J. Werner, J. Chem. Phys. **130**, 241101 (2009).

[167] D. P. Tew, B. Helmich, and C. Hättig, J. Chem. Phys. **135**, 074107 (2011).

[168] C. Krause and H.-J. Werner, Phys. Chem. Chem. Phys. **14**, 7591 (2012).

[169] T. B. Adler, G. Knizia, and H.-J. Werner, J. Chem. Phys. **127**, 221106 (2007).

[170] G. Knizia and H.-J. Werner, J. Chem. Phys. **128**, 154103 (2008).

[171] D. P. Tew, C. Hättig, R. A. Bachorz, and W. Klopper, Explicitly Correlated Coupled-Cluster Theory, in *Recent Progress in Coupled Cluster Methods*, number 11 in Challenges and Advances in Computational Chemistry and Physics, pages 535–572, Springer Netherlands, 2010.

[172] H.-J. Werner, T. B. Adler, G. Knizia, and F. R. Manby, Efficient Explicitly Correlated Coupled-Cluster Approximations, in *Recent Progress in Coupled Cluster Methods*, number 11 in Challenges and Advances in Computational Chemistry and Physics, pages 573–619, Springer Netherlands, 2010.

[173] W. Klopper and C. C. M. Samson, J. Chem. Phys. **116**, 6397 (2002).

[174] E. F. Valeev, Chem. Phys. Lett. **395**, 190 (2004).

[175] S. Ten-no and F. R. Manby, J. Chem. Phys. **119**, 5358 (2003).

[176] F. R. Manby, Explicitly correlated electronic structure theory, in *Solving the Schrödinger equation: has everything been tried?*, edited by P. Popelier, pages 25–42, Imperial College Press, London, 2011.

[177] O. Sinanoğlu, Proc. R. Soc. Lon. Ser. A **260**, 379 (1961).

[178] O. Sinanoğlu, J. Chem. Phys. **36**, 3198 (1962).

[179] C. S. Lin and F. W. Birss, Theor. Chim. Acta **5**, 373 (1966).

[180] P. Wind, W. Klopper, and T. Helgaker, Theor. Chem. Acc. **107**, 173 (2002).

[181] S. Ten-no and J. Noga, WIREs Comput. Mol. Sci. **2**, 114 (2012).

[182] R. A. Bachorz, F. A. Bischoff, A. Glöß, C. Hättig, S. Höfener, W. Klopper, and D. P. Tew, J. Comput. Chem. **32**, 2492 (2011).

[183] V. Termath, W. Klopper, and W. Kutzelnigg, J. Chem. Phys. **94**, 2002 (1991).

[184] W. Klopper and R. Röhse, Theor. Chim. Acta **83**, 441 (1992).

[185] C. Hättig, Phys. Chem. Chem. Phys. **7**, 59 (2005).

[186] K. E. Yousaf and K. A. Peterson, Chem. Phys. Lett. **476**, 303 (2009).

[187] K. E. Yousaf and K. A. Peterson, J. Chem. Phys. **129**, 184108 (2008).

[188] A. Hellweg, C. Hättig, S. Höfener, and W. Klopper, Theor. Chem. Acc. **117**, 587 (2007).

[189] F. Weigend and R. Ahlrichs, Phys. Chem. Chem. Phys. **7**, 3297 (2005).

[190] F. Weigend, J. Comput. Chem. **29**, 167 (2008).

[191] C. Villani and W. Klopper, J. Phys. B: At. Mol. Opt. Phys. **38**, 2555 (2005).

[192] F. Weigend, F. Furche, and R. Ahlrichs, J. Chem. Phys. **119**, 12753 (2003).

[193] W. Klopper, J. Chem. Phys. **120**, 10890 (2004).

[194] Python Software Foundation, *Python 3.4.3 documentation*, 2015, https://docs.python.org/3/.

[195] T. P. Hamilton and H. F. Schaefer III, Chem. Phys. **150**, 163 (1991).

[196] J. T. Fermann, *Efficient Implementation of Vertical Recursion Relations for the Generation of Electron Repulsion Integrals*, Ph. D., University of Georgia, Athens, Georgia, 1996.

[197] U. Ryu, Y. S. Lee, and R. Lindh, Chem. Phys. Lett. **185**, 562 (1991).

[198] R. Ahlrichs, Phys. Chem. Chem. Phys. **6**, 5119 (2004).

[199] Message Passing Interface Forum, *Message Passing Interface Standard, version 3.1*, 2015, http://www.mpi-forum.org/docs/docs.html.

[200] The OpenMP Architecture Review Board, *OpenMP 4.0 complete specifications*, 2013, http://openmp.org/wp/openmp-specifications/.

[201] NVIDIA Corporation, *CUDA Toolkit Documentation v7.0*, 2015, http://docs.nvidia.com/cuda/index.html.

[202] Khronos OpenCL Working Group, *OpenCL 2.1 API, C language and extensions specifications*, 2014, https://www.khronos.org/registry/cl/.

[203] K. Yasuda, J. Comput. Chem. **29**, 334 (2008).

[204] A. Asadchev and M. S. Gordon, J. Chem. Theory Comput. **8**, 4166 (2012).

[205] Basic Linear Algebra Subprograms Technical (BLAST) Forum, *Basic Linear Algebra Subprograms Technical (BLAST) Forum Standard*, 2001, http://www.netlib.org/blas/blast-forum/.

[206] R. C. Whaley, A. Petitet, et al., *Automatically Tuned Linear Algebra Software (ATLAS), version 3.10.1*, 2013, http://math-atlas.sourceforge.net/.

[207] Intel Corporation, *Intel Math Kernel Library (MKL) 11.2*, 2015, https://software.intel.com/en-us/intel-mkl.

[208] M. A. L. Marques, M. J. T. Oliveira, and T. Burnus, Comput. Phys. Commun. **183**, 2272 (2012).

[209] P. Eastman, M. S. Friedrichs, J. D. Chodera, R. J. Radmer, C. M. Bruns, J. P. Ku, K. A. Beauchamp, T. J. Lane, L.-P. Wang, D. Shukla, T. Tye, M. Houston, T. Stich, C. Klein, M. R. Shirts, and V. S. Pande, J. Chem. Theory Comput. **9**, 461 (2013).

[210] D. E. Bernholdt, B. A. Allan, R. Armstrong, F. Bertrand, K. Chiu, T. L. Dahlgren, K. Damevski, W. R. Elwasif, T. G. W. Epperly, M. Govindaraju, D. S. Katz, J. A. Kohl, M. Krishnan, G. Kumfert, J. W. Larson, S. Lefantzi, M. J. Lewis, A. D. Malony, L. C. Mclnnes, J. Nieplocha, B. Norris, S. G. Parker, J. Ray, S. Shende, T. L. Windus, and S. Zhou, Int. J. High. Perform. C. **20**, 163 (2006).

[211] J. P. Kenny, S. J. Benson, Y. Alexeev, J. Sarich, C. L. Janssen, L. C. McInnes, M. Krishnan, J. Nieplocha, E. Jurrus, C. Fahlstrom, and T. L. Windus, J. Comput. Chem. **25**, 1717 (2004).

[212] C. L. Janssen, J. P. Kenny, I. M. B. Nielsen, M. Krishnan, V. Gurumoorthi, E. F. Valeev, and T. L. Windus, J. Phys. Conf. Ser. **46**, 220 (2006).

[213] J. P. Kenny, C. L. Janssen, E. F. Valeev, and T. L. Windus, J. Comput. Chem. **29**, 562 (2008).

[214] J. Backus, Ann. Hist. Comput. **1**, 21 (1979).

[215] S. F. Boys, G. B. Cook, C. M. Reeves, and I. Shavitt, Nature **178**, 1207 (1956).

[216] B. F. Gray, H. O. Pritchard, and F. H. Sumner, J. Chem. Soc. , 2631 (1956).

[217] H. D. Wactlar and M. P. Barnett, Commun. ACM **7**, 704 (1964).

[218] S. Hirata, J. Phys. Chem. A **107**, 9887 (2003).

[219] S. Hirata, J. Chem. Phys. **121**, 51 (2004).

[220] A. A. Auer, G. Baumgartner, D. E. Bernholdt, A. Bibireata, V. Choppella, D. Cociorva, X. Gao, R. Harrison, S. Krishnamoorthy, S. Krishnan, C.-C. Lam, Q. Lu, M. Nooijen, R. Pitzer, J. Ramanujam, P. Sadayappan, and A. Sibiryakov, Mol. Phys. **104**, 211 (2006).

[221] C. L. Janssen and H. F. Schaefer III, Theor. Chim. Acta **79**, 1 (1991).

[222] X. Li and J. Paldus, J. Chem. Phys. **101**, 8812 (1994).

[223] M. Nooijen and V. Lotrich, J. Mol. Struct. THEOCHEM **547**, 253 (2001).

[224] M. Nooijen, Int. J. Mol. Sci. **3**, 656 (2002).

[225] S. Hirata, Theor. Chem. Acc. **116**, 2 (2006).

[226] R. Strange, F. R. Manby, and P. J. Knowles, Comput. Phys. Commun. **136**, 310 (2001).

[227] P. Sałek and A. Hesselmann, J. Comput. Chem. **28**, 2569 (2007).

[228] A. Asadchev, V. Allada, J. Felder, B. M. Bode, M. S. Gordon, and T. L. Windus, J. Chem. Theory Comput. **6**, 696 (2010).

[229] Q. Sun, J. Comput. Chem. **36**, 1664 (2015).

[230] E. F. Valeev, *A library for the evaluation of molecular integrals of many-body operators over Gaussian functions*, 2014, http://libint.valeyev.net/.

[231] D. Kats and F. R. Manby, J. Chem. Phys. **138**, 144101 (2013).

[232] V. Lotrich, N. Flocke, M. Ponton, A. D. Yau, A. Perera, E. Deumens, and R. J. Bartlett, J. Chem. Phys. **128**, 194104 (2008).

[233] E. Deumens, V. F. Lotrich, A. Perera, M. J. Ponton, B. A. Sanders, and R. J. Bartlett, WIREs Comput. Mol. Sci. **1**, 895 (2011).

[234] Python Software Foundation, *The Python programming language, version 3.4*, 2014, https://www.python.org/.

[235] E. Jones, T. Oliphant, P. Peterson, and others, *SciPy: Open source scientific tools for Python*, 2015, http://www.scipy.org/.

[236] S. van der Walt, S. Colbert, and G. Varoquaux, Comput. Sci. Eng. **13**, 22 (2011).

[237] J. Hunter, Comput. Sci. Eng. **9**, 90 (2007).

[238] ISO/IEC 9899:1999, *C99 language standard*, 1999, http://www.iso.org/iso/catalogue_detail.htm?csnumber=29237.

[239] ISO/IEC 1539-1:2004, *Fortran 2003 language standard*, 2004, http://www.iso.org/iso/catalogue_detail.htm?csnumber=39691.

[240] W3C Recommendation 26 November 2008, *Extensible Markup Language (XML) 1.0 (Fifth Edition)*, 2008, http://www.w3.org/TR/2008/REC-xml-20081126/.

[241] ECMA-404, *The JSON Data Interchange Format*, 2013, http://www.ecma-international.org/publications/standards/Ecma-404.htm.

[242] RFC 7159, *The JavaScript Object Notation (JSON) Data Interchange Format*, 2014, https://tools.ietf.org/html/rfc7159.

[243] A. van Deursen, P. Klint, and J. Visser, ACM SIGPLAN Notices **35**, 26 (2000).

[244] Free Software Foundation, Inc., *GNU Compiler Collection, version 4.9.2*, 2014, https://gcc.gnu.org/.

[245] LLVM Team, *clang: a C language family frontend for LLVM, version 3.5.0*, 2014, http://clang.llvm.org/.

[246] Intel Corporation, *Intel C++ Compiler 14.0*, 2014, https://software.intel.com/en-us/compiler_14.0_ug_c.

[247] Free Software Foundation, Inc., *GNU make, version 3.81*, 2006, https://www.gnu.org/software/make/.

[248] R. C. Whaley, A. Petitet, and J. J. Dongarra, Parallel Comput. **27**, 3 (2001).

[249] C. Sanderson, R. Curtin, et al., *Armadillo C++ linear algebra library, verion 5.200.2*, 2015, http://arma.sourceforge.net/.

[250] C. Sanderson, Technical report, NICTA (2010), http://arma.sourceforge.net/armadillo_nicta_2010.pdf.

[251] K. Eichkorn, O. Treutler, H. Öhm, M. Häser, and R. Ahlrichs, Chem. Phys. Lett. **240**, 283 (1995).

[252] K. Eichkorn, F. Weigend, O. Treutler, and R. Ahlrichs, Theor. Chem. Acc. **97**, 119 (1997).

[253] F. Weigend, A. Köhn, and C. Hättig, J. Chem. Phys. **116**, 3175 (2002).

[254] D. E. Woon and T. H. Dunning, J. Chem. Phys. **100**, 2975 (1994).

[255] B. Morgan, *Building an Optimizing Compiler*, Digital Press, Boston, 1997.

[256] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*, Pearson, Harlow, 2nd edition, 2013.

[257] T. Rumsey, *Optimisation and Parallelisation of Generated Code*, M. Eng., University of Bristol, Bristol, 2015.

[258] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann, ACM T. Math. Software **33** (2007).

[259] Free Software Foundation, Inc., *GNU MPFR Library, version 3.1.3*, 2015, http://www.mpfr.org/.

[260] B. A. Mamedov, J. Math. Chem. **36**, 301 (2004).

[261] O. Matsuoka, Comput. Phys. Commun. **3**, 130 (1972).

[262] T. W. Chaundy, *The differential calculus*, Clarendon Press, 1935.

[263] R. D. Johnson III, editor, *NIST Computational Chemistry Comparison and Benchmark Database, Release 16a*, 2013, http://cccbdb.nist.gov/.

[264] A. R. Hoy and P. R. Bunker, J Mol. Spectrosc. **74**, 1 (1979).

[265] F. J. Lovas, E. Tiemann, J. S. Coursey, S. A. Kotochigova, J. Chang, K. Olsen, and R. A. Dragoset, *NIST Diatomic Spectral Database (NIST Standard Reference Database 114)*, 2005, http://www.nist.gov/pml/data/msd-di/index.cfm.