Closed-Shell Hartree-Fock: an Efficient Implementation Based on the Contraction of Integrals in the Primitive Basis

Inaugural-Dissertation

zur
Erlangung des Doktorgrades
der Mathematisch-Naturwissenschaftlichen Fakultät
der Universität zu Köln

vorgelegt von

Joseph Held

aus München

Köln 2017

Berichterstatter:	Prof. Dr. M. Dolg
	PD Dr. M. Hanrath

Tag der letzten mündlichen Prüfung:	05. Juli 2017

Meinen Eltern

Abstract

In this work a highly efficient, closed-shell restricted Hartree-Fock self-consistent-field (SCF) program has been developed. It has been written as a part of the "Quantum Objects Library" (QOL), which is developed at the Institute for Theoretical Chemistry at the University of Cologne.

In the implementation presented here, the explicit transformation of the two-electron integrals from the primitive to the radial contracted, angularly transformed basis is avoided. The density matrix is obtained by diagonalizing the Fock matrix in the radial-angular-transformed basis. It is transformed to the primitive basis, and contracted with the integrals to give the Fock matrix, also in the primitive basis. To assure that solutions of the Fock equations are obtained in the transformed basis, the Fock matrix is transformed back to the radial contracted, angularly transformed basis after the density contraction step.

Both one- and two-electron integrals are calculated using the ansatz of Obara-Saika (OS). For the evaluation of the two-electron integrals a code-generating ansatz is used. At compile time optimized code is generated, to be called at runtime. Numerical instabilities inherent in the Obara-Saika scheme were analyzed and eliminated. The code-generating ansatz was also used for the implementation of the density contraction. Contraction codes for both primitive and radial-angular-transformed bases were developed. Both two-electron integral evaluation and density contraction implementations have been optimized by explicit use of streaming single instruction multiple data (SIMD) extensions (SSE).

Integral prescreening has been implemented on basis of the Schwarz inequality, and the differential density scheme is used. Convergence acceleration has been realized by implementing the direct inversion of the iterative subspace (DIIS) method. The developed SCF program has been parallelized using the open multi-processing (OMP) application programming interface (API).

In final performance comparisons to commercially available programs competitive results were obtained.

Kurzzusammenfassung

In dieser Arbeit wurde ein hocheffizientes, closed-shell restricted Hartree-Fock self-consistent-field (SCF) Programm entwickelt. Es wurde in die "'Quantum Objects Library"' (QOL) integriert, die am Institut für Theoretische Chemie der Universität zu Köln entwickelt wird.

In der hier vorgestellten Implementierung wird die explizite Transformation der Zwei-Elektronen Integrale aus der primitiven in die radial kontrahierte, sphärisch transformierte Basis umgangen. Durch Diagonalisierung der Fock Matrix in der radial-sphärisch transformierten Basis wird die Dichte Matrix erhalten. Diese wird in die primitive Basis transformiert und mit den Integralen kontrahiert, wodurch die Fock Matrix ebenfalls in ihrer primitiven Darstellung vorliegt. Um sicherzustellen dass die Lösungen der Fock Gleichungen in der transformierten Basis erhalten werden, wird die Fock Matrix nach der Dichte Kontraktion zurück in die radial-sphärische Basis transformiert.

Sowohl Ein- als auch Zwei-Elektronen Integrale werden mit dem Verfahren von Obara-Saika (OS) berechnet, wobei für die Zwei-Elektronen Integrale ein Code generierender Ansatz gewählt wurde. Dabei wird zum Zeitpunkt der Kompilierung optimierter Code generiert, der zur Laufzeit ausgeführt wird. Die numerischen Instabilitäten des Obara-Saika Verfahrens wurden analysiert und beseitigt. Auch auf die Implementierung der Kontraktion der Dichte wurde der Code generierende Ansatz angewandt, wobei Spezialisierungen sowohl für primitive als auch für sphärisch transformierte Basen erzeugt wurden. beide Implementierungen, Integral Berechnung und Dichte Kontraktion, wurden optimiert durch die explizite Berücksichtigung der streaming single instruction multiple data (SIMD) extensions (SSE).

Eine Abschätzung der Integral Größenordnungen wurde auf Basis der Schwarz'schen Ungleichung umgesetzt, und das differential density scheme wird benutzt. Zur Konvergenz Beschleunigung kommt das direct inversion of the iterative subspace (DIIS) Verfahren zum Einsatz. Das entwickelte SCF Programm wurde mittels der open multi-processing (OMP) Programmierschnittstelle (API) parallelisiert.

In finalen Vergleichen zu kommerziell erhältlichen Programmen wurden konkurrenzfähige Resultate im Hinblick auf die Performanz erreicht.

Contents

1. Introduction

The main objective of quantum chemistry is the mathematical description of atomic and molecular systems. Such systems are characterized by the Schrödinger equation [1], the required information can be obtained from the associated wave function. As closed solutions of this equation only exist for the most simple problems, approximations have to be introduced for the treatment of more complicated systems. One of the most central approximations is the restriction of the wave function to the electronic problem [2]. Two other widely used approximations are the treatment in the non-relativistic framework, and the neglection of time dependence. These three approximations are used in this work.

Even in this approximation however, except for one-electron systems, closed solutions are not available. A first, and central, ansatz to find approximate solutions for the many-electron problem has been given by Hartree and Fock [3–7]. In Hartree-Fock theory, the many-electron problem is transformed to effective one-electron problems, in which the influence of the remaining electrons is treated in an average way. However, the obtained integro-differential equations are still complicated to solve. Only further approximations made these equations applicable to a wider range of problems. By projection to a finite space, i.e. by introducing finite basis sets, the complicated integro-differential equations can be formulated as rather simple matrix problems. In the restricted (spin-free), closed-shell formulation, which this work is concerned about, the matrix eigenvalue equation of Roothaan and Hall [8, 9] is obtained. As the Fock matrix to be diagonalized in this equation depends on its solution, the Roothaan-Hall equation has to be solved iteratively. Starting with an guess for the solutions, these equations are solved iteratively, until the corresponding field becomes self-consistent.

Even neglecting electron correlation, the ground state wave function obtained from the Hartree-Fock equations is not sufficient to describe all systems accurately. Nevertheless, Hartree-Fock theory still plays a central role in quantum chemistry. Despite the age of the theory, the wave functions obtained from Hartree-Fock calculations are widely used. They constitute the entry point for a multitude of advanced ab-initio calculations.

With increasing computational resources and advances in theory, programs were made available to treat a variety of chemical problems in an standardized way. Still, vast resources are needed to address even systems of moderate size. Over the years, a lot of development [10–12] has been invested in optimizing the algorithms involved in the calculation of the Hartree-Fock ground state wave function. Target of most of these optimizations is the efficient treatment of the vast number of two-electron integrals arising. They scale with the fourth power of the system size and constitute a bottleneck of Hartree-Fock calculations. Various compromises have therefore been introduced, to computationally simplify the evaluation of those integrals.

Among the first of these compromises has been the development of direct SCF calculations [13]. Theoretically, the two-electron integrals stay constant in each SCF iteration. In the first implementations, they were therefore calculated once, and stored to disk. In these conventional SCF calculations however, the treatable system size is limited by the physically available disk space. In direct implementations, the integrals are recalculated in each iteration, and processed immediately. By removing the disk size limitations, larger systems became accessible, although at the cost of increased computational time. In the long run, the direct implementation became state of the art. This is also due to the advent of advanced prescreening [14, 15] methods, by which a large number of integrals can be neglected. Especially in the context of the differential density scheme [13, 14, 16], the calculation of a certain integral can be dynamically determined in each iteration.

One of the major breakthroughs however has been the introduction of primitive Cartesian Gaussians [17] as basis functions. Using these functions constitutes another compromise of theory and technical realization. Although rather poorly suited to describe the physical reality, they are now used almost exclusively in electronic structure calculations. This is due to the fact that they greatly simplify the calculation of the costly two-electron integrals. Their lack of physical description is accounted for by increasing the number of functions included, thus spanning the space required for a certain accuracy. In practice, the correct radial description is achieved by linear combinations of several primitive Cartesian Gaussians of differing exponents. Another set of linear combinations transforms these functions to spherical harmonics. This has implications to the practical realization, as quantities have to be transformed from and to this different basis sets. Nevertheless the ease of integration of primitive Cartesian Gaussians justifies these implications.

The restricted, closed-shell SCF implementation presented in this work now focuses on these implications. In common implementations, the transformation to the radial contracted and angularly-transformed basis is applied to the primitive integrals. These transformed integrals are then contracted with the transformed density. In this work a different ansatz is implemented. The contraction of the density with the integrals is performed in the primitive basis. After diagonalizing the Fock matrix, in the transformed basis, the resulting density is transformed to the primitive basis. After contraction with the primitive integrals the resulting Fock matrix is then transformed back to the radial-angular basis. Thus it is assured that the actual solution of the Fock equations is obtained in the transformed basis.

The restricted, closed-shell SCF program presented here has been written as part of the Quantum Objects Library (QOL) [18] program package of the Institute for Theoretical Chemistry at the University of Cologne. Most of the techniques used in commercially available programs were included. Thus the results presented here constitute not a proof of concept study, but a fully comparable production code. Among the techniques implemented are integral prescreening based on the Schwarz inequality, in the context of the differential density scheme. Convergence acceleration is used with the direct inversion of the iterative subspace (DIIS) [19] ansatz. One- and two-electron integrals are calculated using the Obara-Saika scheme [20, 21]. For the two-electron integrals a highly efficient code-generated ansatz has been chosen, in which the integration code is automatically

generated at compile time, for arbitrary integrals. The density contraction also has been implemented using the code-generating ansatz. At compile time, efficient code is generated to be called at runtime. This has been done for contractions in primitive bases, as well as for contractions in spherical-harmonic basis sets. Both integration and contraction have been optimized by explicit use of streaming single instruction multiple data (SIMD) extensions (SSE).

Part I.

Theory

2. Hartree-Fock Theory

In this chapter, the key steps of the derivation of the restricted closed-shell Hartree-Fock equations will be shown. The discussion given here follows the books of Szabo and Ostlund [22], Levine [23] and Helgaker, Jørgensen and Olsen [24], to which the reader is referred to for more detailed informations.

2.1. Electronic Solutions for Restricted Closed-Shell Systems

Quantum chemistry is concerned with finding approximate solutions for the non-relativistic, time independent Schrödinger equation.

$$\hat{H}\Psi = E\Psi \tag{2.1}$$

The Hamilton operator

$$\hat{H} = -\sum_{i=1}^{N}\frac{1}{2}\nabla_i^2 - \sum_{A=1}^{M}\frac{1}{2M_A}\nabla_A^2 - \sum_{i=1}^{N}\sum_{A=1}^{M}\frac{Z_A}{r_{iA}} + \sum_{i=1}^{N}\sum_{j>i}^{N}\frac{1}{r_{ij}} + \sum_{A=1}^{M}\sum_{B>A}^{M}\frac{Z_A Z_B}{r_{AB}} \tag{2.2}$$

describes the interactions of the M nuclei and N electrons of the system of interest, using the indices A and B to indicate nuclei, and i and j for the electrons. The first two terms describe the kinetic energies of electrons and nuclei. The third their Coulomb attraction. The fourth and fifth the repulsion of electrons and nuclei, respectively. M_A is the mass ratio of nucleus A to an electron, Z_A the charge of that nucleus. The distances of nuclei and electrons are depicted by r.

In the Born-Oppenheimer [2] approximation the fact is used that electrons move much faster than nuclei. They can be considered to move in the field of fixed nuclei. Within this approximation, the second term of (2.2), the kinetic energy of the nuclei, can be neglected and the nuclear repulsion is just a constant. In this ansatz, the electronic problem is solved using the electronic hamiltonian.

$$\hat{H}_{elec} = -\sum_{i=1}^{N}\frac{1}{2}\nabla_i^2 - \sum_{i=1}^{N}\sum_{A=1}^{M}\frac{Z_A}{r_{iA}} + \sum_{i=1}^{N}\sum_{j>i}^{N}\frac{1}{r_{ij}} \tag{2.3}$$

Finding approximate solutions for the electronic Schrödinger equation is the concern of Hartree-Fock theory.

$$\hat{H}_{elec}\Phi_{elec} = E_{elec}\Phi_{elec} \tag{2.4}$$

The solutions of this equation depend parametrically on the coordinates of the nuclei, giving different wave functions and energies for different nuclear arrangements.

To obtain the total energy of the system, the constant nuclear repulsion has to be taken into account.

$$E_{tot} = E_{elec} + \sum_{A=1}^{M} \sum_{B>A}^{M} \frac{Z_A Z_B}{r_{AB}} \tag{2.5}$$

As this work is only concerned with solutions of the electronic Schrödinger equation (2.4), the subscript $_{elec}$ to the hamiltonian, energy and wave function will be dropped in the following.

The electronic wave function will be set up from linear combinations of products of spin orbitals. In a spin orbital χ, one electron is described by the product of its spatial distribution $\psi(r)$ and one of the two spin functions $\alpha(\omega)$ or $\beta(\omega)$.

$$\chi(x) = \psi(r)\alpha(\omega) \quad \text{or} \quad \chi(x) = \psi(r)\beta(\omega) \tag{2.6}$$

Here the spatial and spin coordinates r and ω are collected to x. The spatial orbitals are assumed to form an orthonormal set, from which orthonormality in the set of the spin orbitals follows.

$$\int \chi_i^*(r)\chi_j(r)dr = \langle \chi_i | \chi_j \rangle = \delta_{ij} \tag{2.7}$$

To account for the indistinguishability of the electrons and the antisymmetry of the wave function, Slater determinants [25] are used to set up the wave function.

$$\Phi(x_1, x_2, \ldots, x_N) = (N!)^{-1/2} \begin{vmatrix} \chi_i(x_1) & \chi_j(x_1) & \cdots & \chi_k(x_1) \\ \chi_i(x_2) & \chi_j(x_2) & \cdots & \chi_k(x_2) \\ \vdots & \vdots & & \vdots \\ \chi_i(x_N) & \chi_j(x_N) & \cdots & \chi_k(x_N) \end{vmatrix} \equiv |\chi_i \chi_j \cdots \chi_k\rangle \tag{2.8}$$

In this Slater determinant, N electrons occupy N spin orbitals. No specification is made as to which electron occupies which spin orbital, thus ensuring indistinguishability. Antisymmetry is met by the mathematical properties of the determinant [26]. Interchange of two rows changes the sign of the determinant. This corresponds to an interchange of two electrons. The Pauli exclusion principle [27] is met by another property of the determinant. Two electrons occupying the same spin orbital results in two equal columns, for which the determinant is zero.

In Hartree-Fock theory, a single determinant ansatz is used to describe the ground state of an N-electron system.

$$|\Phi_0\rangle = |\chi_1 \chi_2 \cdots \chi_N\rangle \tag{2.9}$$

Specifically, this work is concerned with solutions for closed-shell systems, using a set

of restricted spin orbitals. In a restricted set of spin orbitals, two spin orbitals $\chi(x)$ are built from one spatial orbital $\psi(r)$, one with α- and the other with β-spin, respectively. In a closed-shell system with N electrons, the ground state is then built from $N/2$ spatial orbitals, each of which is doubly occupied.

$$|\Phi_0\rangle = |\psi_1\alpha\ \psi_1\beta\ \cdots\ \psi_{N/2}\alpha\ \psi_{N/2}\beta\rangle \tag{2.10}$$

The expectation value for the ground state energy can be expressed in terms of spatial orbitals. Application of the Slater-Condon [28, 29] rules and integrating out spin gives the Hartree-Fock ground state energy E_0.

$$E_0 = \langle\Phi_0|\hat{H}|\Phi_0\rangle = \sum_{i}^{N/2}\langle i|\hat{h}|i\rangle + \frac{1}{2}\sum_{ij}^{N/2}\left(2\langle ij|ij\rangle - \langle ii|jj\rangle\right) \tag{2.11}$$

Here, the one-electron part of the hamiltonian is given by

$$\langle i|\hat{h}|j\rangle = \int \psi_i^*(r_1)\left(-\frac{1}{2}\nabla^2 - \sum_{A}^{M}\frac{Z_A}{r_{1A}}\right)\psi_j(r_1)dr_1 \tag{2.12}$$

and the two-electron part by

$$\langle ij|kl\rangle = \iint \psi_i^*(r_1)\psi_j^*(r_2)\frac{1}{r_{12}}\psi_k(r_1)\psi_l(r_2)dr_1dr_2 \tag{2.13}$$

Note that the physicists notation will be used for all two-electron integrals over spatial functions. The specific nature of those functions will be implied by the context.

Now the variational principle states that the expectation value of the trial function $|\Phi_0\rangle$ is always greater than or equal to the exact electronic energy.

$$E \leq \langle\Phi_0|\hat{H}|\Phi_0\rangle = E_0 \tag{2.14}$$

The strategy is then to minimize E_0 by varying the spatial orbitals ψ. This can be done using Lagranges' method of undetermined multipliers, under the constraint of orthonormal spatial orbitals. Doing so leads to the restricted, canonical closed-shell Fock equation.

$$\hat{f}\,|\psi_a\rangle = \epsilon_a\,|\psi_a\rangle \tag{2.15}$$

It is of the eigenvalue form, determining the spatial orbitals and their energy ϵ. The restricted, closed-shell Fock operator is defined as

$$\hat{f}(1) = \hat{h}(1) + \sum_{b}^{N/2}\left(2\hat{J}_b(1) - \hat{K}_b(1)\right) \tag{2.16}$$

It is an effective one-electron operator. Similar to (2.12), operators depending only on

the coordinate of one electron are collected in the one-electron operator $\hat{h}$.

$$\hat{h}(1) = -\frac{1}{2}\nabla_1^2 - \sum_A \frac{Z_A}{r_{1A}} \tag{2.17}$$

The effects of the other electrons are included in an averaged way by the summation over the two-electron operators $\hat{J}$ and $\hat{K}$. The local Coulomb operator

$$\hat{J}_b(1)\psi_a(1) = \left[\int \frac{\psi_b^*(2)\psi_b(2)}{r_{12}} dx_2\right] \psi_a(1) \tag{2.18}$$

can be interpreted as a classical potential. The non-local exchange operator

$$\hat{K}_b(1)\psi_a(1) = \left[\int \frac{\psi_b^*(2)\psi_a(2)}{r_{12}} dx_2\right] \psi_b(1) \tag{2.19}$$

has no classical interpretation.

2.2. The Roothaan-Hall Equations

The integro-differential equation (2.15) can be turned into a set of algebraic equations, which can be solved by standard matrix techniques. After Roothaan, this is accomplished by introducing a set of k known basis functions ϕ, in which the molecular orbital (MO) ψ are linearly expanded.

$$\psi_l = \sum_j^k C_{jl}\phi_j \tag{2.20}$$

Using this expansion in (2.15), multiplying with ϕ_m^* on the left and integrating, the Roothaan equations are obtained.

$$\sum_j^k C_{jl} \int \phi_m^*(1)\hat{f}(1)\phi_j(1)dr_1 = \epsilon_l \sum_j^k C_{jl} \int \phi_m^*(1)\phi_j(1)dr_1 \tag{2.21}$$

Introducing the Fock

$$F_{mj} = \int \phi_m^*(1)\hat{f}(1)\phi_j(1)dr_1 \tag{2.22}$$

and the overlap matrix

$$S_{mj} = \int \phi_m^*(1)\phi_j(1)dr_1 \tag{2.23}$$

the Roothaan Hall equations can be written in the generalized eigenvalue form.

$$\mathbf{FC} = \mathbf{SC}\boldsymbol{\epsilon} \tag{2.24}$$

The Fock matrix is

$$\mathbf{F} = \mathbf{H}^{\mathbf{core}} + \mathbf{G} \tag{2.25}$$

where the one-electron part $\mathbf{H}^{\mathbf{core}}$ is the sum of kinetic- and nuclear attraction integrals.

$$H_{ij}^{core} = -\frac{1}{2}\int \phi_i^*(1)\nabla^2\phi_j(1)dr_1 - \sum_A \int \phi_i^*(1)\frac{Z_A}{r_{1A}}\phi_j(1)dr_1 \tag{2.26}$$

$$= T_{ij} + V_{ij} \tag{2.27}$$

For the two-electron part $\mathbf{G}$, in terms of two-electron integrals over the basis functions $|i\rangle = \phi_i(r)$, the following is obtained.

$$G_{ij} = \sum_{kl} D_{kl}\left[\langle ik|jl\rangle - \frac{1}{2}\langle ik|lj\rangle\right] \tag{2.28}$$

Here the density matrix $\mathbf{D}$ has been introduced as the product of the 'occupied' columns of the expansion coefficients.

$$D_{ij} = 2\sum_k^{N/2} C_{ik}C_{jk}^* \equiv \mathbf{C_{occ}} \cdot \mathbf{C_{occ}^*} \tag{2.29}$$

3. Integral Evaluation

Integrals over basis functions are of central importance in electronic structure theory, both from a theoretical and technical point of view. In technical terms, this is due to the sheer number of integrals arising already for calculations of small systems. Especially the two-electron integrals, scaling with the fourth power of the number of basis functions, turn out to be rather demanding in terms of computational resources. An optimal implementation is therefore mandatory for highly efficient production codes.

Several schemes exist to evaluate integrals over basis functions. This work focuses on the implementation of the recurrence relations of Obara and Saika [20, 21]. Other evaluation schemes, such as the Gaussian quadrature [30, 31], the McMurchie-Davidson scheme [32] and the method of Pople et al. [33, 34] will not be discussed.

It shall be noted that most of the discussion given in this chapter follows the one given in the book of Helgaker, Jørgensen and Olsen [24].

3.1. Primitive Cartesian Gaussians

Of great importance is the choice of basis functions to be integrated over. In the long run, primitive Cartesian Gaussian (PCG) [17] functions became the state of art, trading their lack of correct physical description for ease of integration [24]. They are specified by an exponent a, a center A and have a total angular momentum of $l = i + j + k$,

$$G_{\substack{i\\j\\k}}(r, a, A) = x_A^i y_A^j z_A^k \exp(-a r_A^2) = G_a(r). \tag{3.1}$$

The vector $r_A = r - A$ is the distance relative to the center. Note that in the abbreviated form $G_a(r)$, exponent, center and angular momentum information are collected to the single index a.

Their main advantage is that they are separable with respect to the three Cartesian directions:

$$\exp\left(-a\left(\sqrt{x_A^2 + y_A^2 + z_A^2}\right)^2\right) = \exp(-a x_A^2)\exp(-a y_A^2)\exp(-a z_A^2) \tag{3.2}$$

This drastically simplifies the evaluation of integrals. The physically better suited Slater type functions lack this separability.

$$\exp(-a r_A) = \exp\left(-a\sqrt{x_A^2 + y_A^2 + z_A^2}\right) \neq f(x)f(y)f(z) \tag{3.3}$$

Another property is that the product of two (or any number of) PCG's can be written in terms of a different, single PCG. This is the Gaussian product rule [17]. Separability can be used, giving the product of two spherical ($l = 0$) Gaussians in x-direction as

$$\exp(-ax_A^2) \cdot \exp(-bx_B^2) = K_{ab}^x \exp(-px_P^2) \tag{3.4}$$

Where

$$p = a + b \tag{3.5}$$
$$P_x = \frac{aA_x + bB_x}{p} \tag{3.6}$$
$$X_{AB} = A_x - B \tag{3.7}$$
$$\mu = \frac{ab}{a+b} \tag{3.8}$$
$$K_{ab}^x = \exp(-\mu X_{AB}^2) \tag{3.9}$$

For arbitrary l, the one-dimensional Gaussian overlap distribution is just

$$\Omega_{ij}^x(x) = x_A^i x_B^j K_{ab}^x \cdot \exp(-px_p^2) \tag{3.10}$$

Applying this to all Cartesian direction, the general, three-dimensional Gaussian overlap distribution $\Omega_{ab}(r) = G_a(r) \cdot G_b(r)$ can be split up into one-dimensional parts

$$\Omega_{ab}(r) = \Omega_{\substack{ij \\ kl \\ mn}}(r) = \Omega_{ij}^x(x)\Omega_{kl}^y(y)\Omega_{mn}^z(z) \tag{3.11}$$

The trivial relationships

$$\Omega_{i+1,j}^x = x_A \Omega_{ij}^x \tag{3.12}$$
$$\Omega_{i,j+1}^x = x_B \Omega_{ij}^x \tag{3.13}$$

lead to the recurrence relation

$$\Omega_{i,j+1}^x - \Omega_{i+1,j}^x = X_{AB}\Omega_{ij}^x. \tag{3.14}$$

Finally, the derivatives of the Gaussian overlap distributions with respect to the center coordinates are

$$\frac{\partial \Omega_{ij}^x}{\partial A_x} = 2a\Omega_{i+1,j}^x - i\Omega_{i-1,j}^x \tag{3.15}$$
$$\frac{\partial \Omega_{ij}^x}{\partial B_x} = 2b\Omega_{i,j+1}^x - i\Omega_{i,j-1}^x \tag{3.16}$$

3.1.1. Shells and Shell Batches

Central to all integration schemes presented in this work is the concept of grouping basis functions into shells. A shell is defined as the set of functions located on the same center and having the same angular momentum and exponent. For $l = 1$, some center A and some exponent a, this particular p-shell consists of the three functions

$$p^A_{a,x} = x_A \exp(-ar^2_A) \tag{3.17}$$
$$p^A_{a,y} = y_A \exp(-ar^2_A) \tag{3.18}$$
$$p^A_{a,z} = z_A \exp(-ar^2_A) \tag{3.19}$$

Mostly, shells will be addressed solely by their angular momentum, i.e. p, f and so on. If a distinction for shells of equal angular momenta has to be made, a combined index will be used, i.e. a p_i-shell is supposed to have a different center and or exponent than a p_j-shell.

Generally, the number of PCG's in a given shell of angular momentum l is

$$N^{pcg}_l = \frac{(l+1)(l+2)}{2} \tag{3.20}$$

A shell batch is the set of integrals arising from the combination of two (one-electron integrals) or four (two-electron integrals) shells. Again, shells will be denoted by their angular momenta, using indices when necessary. The two-electron $dpss$-batch for example can have two distinctive (dps_is_j), or equal (dps_is_i) s-shells.

Consider for example the shell batch $p_ip_js_ks_l$, with arbitrary labeled functions.

$$p_ip_js_ks_l \equiv \left\{ \langle \begin{pmatrix} a \\ b \\ c \end{pmatrix} \begin{pmatrix} d \\ e \\ f \end{pmatrix} | (g) (h) \rangle \right\}$$

Combining the functions in the shells leads to the following nine integrals. The order of the functions is in accordance to the order of the shells. Permutations of the shells leading to an according permutation of the functions in the integrals.

$$\begin{array}{ccc} \langle ad|gh\rangle & \langle bd|gh\rangle & \langle cd|gh\rangle \\ \langle ae|gh\rangle & \langle be|gh\rangle & \langle ce|gh\rangle \\ \langle af|gh\rangle & \langle bf|gh\rangle & \langle cf|gh\rangle \end{array}$$

The generation of the integrals of a batch is just the Cartesian product of the four sets of functions (shells): $\{a, b, c\}$ x $\{d, e, f\}$ x $\{g\}$ x $\{h\}$. In this case the Cartesian product is associative.

Neglecting redundancies, the number of integrals present in a batch is then just the product of the shell sizes.

The reason to impose this grouping of integrals into batches is technical, based on the reusage of shared quantities within the calculation of integrals of a given batch. For

example, all prefactors (3.5) to (3.9) of a Gaussian overlap distribution (3.11) depend only on the exponents (a and b) and centers (A and B). Thus, computing them once prior to the evaluation of all $N_{l,i} \cdot N_{l,j}$ Gaussian overlap distributions, and reusing them when needed, clearly saves computational cost compared to repeated recomputation. Additionally, the integration schemes used for this work are recursive, involving a lot of intermediates. Again, most of these intermediates are needed for the calculation of different integrals in a given batch. Calculation of one batch as a whole exploits this by reusing those intermediates, compared to a costly recalculation in an individual integral evaluation scheme.

3.2. The Obara-Saika Scheme

In this work, the recurrence relations derived by Obara et al. [20, 21] will be used to evaluate all integrals necessary for a Hartree-Fock SCF calculation. It will be used in the modified version introduced by Head-Gordon and Pople [35], Hamilton and Schäfer [36] and Lindh, Ryu and Liu [37].

Generally, integrals over PCG's of specific angular momenta are expressed in terms of integrals of in- or decreased angular momenta. Starting with integrals over spherical PCG's ($l = 0$), integrals for higher angular momenta can be obtained by application of the recurrence relations. For the spherical recursion seeds analytical expressions are known.

Each PCG has three dimensions of angular momentum $l = i + j + k$. To specify a one-electron integral (two PCG's), six indices are needed, 12 for a two-electron integral (four PCG's), respectively. However, based on the separability of the PCG's, there is no mixing of directions in the Obara-Saika schemes. To increment some angular momentum in x-direction, only in- or decrements in the x-direction are needed. This allows for some simplification in the notation, by omitting the unneeded directions. The actual direction the recurrence relation is applied to is given implicitly by the direction of the associated center coordinate.

Explicit derivations will be shown for the one-electron overlap and kinetic-energy integrals. For Coulomb integrals, both one- and two-electron, the derivation will be skipped, an extensive derivation can be found in the book of Helgaker et al. [24].

3.2.1. Overlap- and Kinetic-Energy Integrals

Overlap Integrals

The overlap integrals S_{ab} are of the form

$$S_{ab} = \int_{-\infty}^{\infty} G_a^*(r) G_b(r)\, dr = \langle a|b\rangle \tag{3.21}$$

As the PCG's used in this work are real, complex conjugation has no effect and will be omitted from now on (this concerns all integrals over PCG's). Thus, the overlap

integrals are just integrals over a Gaussian overlap distribution and can be separated in the three Cartesian directions.

$$\begin{aligned} S_{ab} &= \int_{-\infty}^{\infty} \Omega_{ab}\, dr \\ &= \int_{-\infty}^{\infty} \Omega_{ij}^{x}\, dx \int_{-\infty}^{\infty} \Omega_{kl}^{y}\, dy \int_{-\infty}^{\infty} \Omega_{mn}^{z}\, dz \\ &= S_{ij}^{x}\, S_{kl}^{y}\, S_{mn}^{z} \end{aligned} \tag{3.22}$$

The horizontal recurrence relation follows directly from integrating (3.14):

$$S_{i,j+1}^{x} - S_{i+1,j}^{x} = X_{AB} S_{ij}^{x} \tag{3.23}$$

Additionally, the overlap integrals are invariant to identical translations of the centers A and B

$$\frac{\partial S_{ij}^{x}}{\partial A_x} + \frac{\partial S_{ij}^{x}}{\partial B_x} = 0 \tag{3.24}$$

Insertion of the definition (3.22) and application of the derivatives of the Gaussian overlap distributions (3.15) and (3.16) then gives the recurrence relation

$$2aS_{i+1,j}^{x} - iS_{i-1,j}^{x} + 2bS_{i,j+1}^{x} - jS_{i,j-1}^{x} = 0 \tag{3.25}$$

Finally, using the horizontal recursion (3.23), the Obara-Saika recurrence relations for the overlap integrals can be obtained

$$S_{i+1,j}^{x} = X_{PA} S_{ij}^{x} + \frac{1}{2p}\left(iS_{i-1,j}^{x} + jS_{i,j-1}^{x}\right) \tag{3.26}$$

$$S_{i,j+1}^{x} = X_{PB} S_{ij}^{x} + \frac{1}{2p}\left(iS_{i-1,j}^{x} + jS_{i,j-1}^{x}\right) \tag{3.27}$$

The center of charge P and the total exponent p have their origin in the Gaussian overlap distribution, eqns. (3.6) and (3.5), respectively.

The recursion seed, S_{00}^{x} is just the integral of a spherical Gaussian distribution

$$S_{00}^{x} = K_{ab}^{x} \int_{-\infty}^{\infty} \exp(-px_P^2) = K_{ab}^{x} \sqrt{\frac{\pi}{p}} \tag{3.28}$$

The actual overlap integral S_{ab} can then be calculated from (3.22).

Kinetic-Energy Integrals

The kinetic-energy integrals

$$T_{ab} = \int_{-\infty}^{\infty} G_a(r)(-\frac{1}{2}\nabla^2)G_b(r)dr = \langle a|\hat{T}|b\rangle \tag{3.29}$$

are closely related to the overlap integrals. In a first step, the integral is separated into Cartesian directions via the Laplace-operator

$$T_{ab} = -\frac{1}{2}\left(\langle G_a|\frac{\partial^2}{\partial x^2}|G_b\rangle + \langle G_a|\frac{\partial^2}{\partial y^2}|G_b\rangle + \langle G_a|\frac{\partial^2}{\partial z^2}|G_b\rangle\right) \tag{3.30}$$

Now the separability of the PCG's shows the dependence on the overlap integrals

$$-\frac{1}{2}\langle G_a|\frac{\partial^2}{\partial x^2}|G_b\rangle = S^y_{kl}S^z_{mn}\left(-\frac{1}{2}\langle G^x_i|\frac{\partial^2}{\partial x^2}|G^x_j\rangle\right) = S^y_{kl}S^z_{mn}T^x_{ij} \tag{3.31}$$

which gives the total kinetic-energy integral as

$$T_{ab} = T^x_{ij}S^x_{kl}S^x_{mn} + S^y_{ij}T^y_{kl}S^y_{mn} + S^z_{ij}S^z_{kl}T^z_{mn} \tag{3.32}$$

Using

$$\frac{\partial G^x_i}{\partial x} = -\frac{\partial G^x_i}{\partial A_x} \tag{3.33}$$

the kinetic-energy integral can be written in terms of a differentiated overlap integral

$$T^x_{ij} = -\frac{1}{2}\langle G^x_i|\frac{\partial^2}{\partial x^2}|G^x_j\rangle = -\frac{1}{2}\frac{\partial^2}{\partial A^2_x}S^x_{ij} \tag{3.34}$$

The final recurrence relations for the one dimensional kinetic-energy integrals can then be obtained by differentiation of the overlap recurrence relations (3.26) and (3.27) with respect to A_x.

$$T^x_{i+1,j} = X_{PA}T^x_{ij} + \frac{1}{2p}\left(iT^x_{i-1,j} + jT^x_{i,j-1}\right) + \frac{b}{p}\left(2aS^x_{i+1,j} - iS^x_{i-1,j}\right) \tag{3.35}$$

$$T^x_{i,j+1} = X_{PB}T^x_{ij} + \frac{1}{2p}\left(iT^x_{i-1,j} + jT^x_{i,j-1}\right) + \frac{a}{p}\left(2bS^x_{i,j+1} - iS^x_{i,j-1}\right) \tag{3.36}$$

The recursion seed is

$$T^x_{00} = \left[a - 2a^2\left(X^2_{PA} + \frac{1}{2p}\right)\right]S^x_{00} \tag{3.37}$$

3.2.2. Coulomb Integrals

The derivation of the Coulomb integrals

$$\int_{-\infty}^{\infty}\frac{G_a(r)G_b(r)}{r_C}dr = \langle a|\frac{1}{r_C}|b\rangle \tag{3.38}$$

and

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\frac{G_a(r_1)G_c(r_2)G_b(r_1)G_d(r_2)}{r_{12}}dr_1dr_2 = \langle ac|bd\rangle \tag{3.39}$$

shall not be repeated here, only key steps will be shown. An extensive derivation can be found in the book of Helgaker et al. [24].

The Boys Function

Central to the evaluation of the Coulomb integrals (3.38) and (3.39) is the inverse distance operator $\frac{1}{r}$. Due to its presence, the Coulomb integrals can not be factored in the Cartesian directions. However, the inverse distance operator can be expressed as an one-dimensional integral over a spherical PCG.

$$\frac{1}{r} = \frac{1}{\sqrt{\pi}}\int_{-\infty}^{\infty}\exp(-r^2t^2)dt \tag{3.40}$$

This is the key step in the evaluation of Coulomb integral over PCG's. It leads to their expression in terms of a special class of functions, the Boys function [17]

$$F_N(x) = \int_0^1 \exp(-xt^2)t^{2N}dt \tag{3.41}$$

The integration over the whole one- or two-particle space has been reduced to a one-dimensional integration over a fixed interval.

One-Electron Integrals

The Obara-Saika recurrence relations for the one-electron Coulomb integrals are expressed in terms of auxiliary integrals $\Theta^N_{ij;kl;mn}$. The superscript N having its origin in the order of the Boys function and the subscripts specifying the angular momenta in x-, y- and z-direction. The integrals with $N = 0$ are the actual Coulomb integrals.

$$\Theta^0_{\substack{ij\\kl\\mn}} = \langle a|\frac{1}{r_C}|b\rangle \tag{3.42}$$

Similar to the overlap and kinetic-energy integrals, recurrence relations exist to increase angular momenta for the bra-index i.

$$\Theta^N_{i+1,j} = X_{PA}\Theta^N_{ij} + \frac{1}{2p}\left(i\Theta^N_{i-1,j} + j\Theta^N_{i,j-1}\right) - X_{PC}\Theta^{N+1}_{ij} - \frac{1}{2p}\left(i\Theta^{N+1}_{i-1,j} + j\Theta^{N+1}_{i,j-1}\right) \tag{3.43}$$

Again, the prefactors arising from the Gaussian overlap distribution (3.5) to (3.9). For the ket-index j the horizontal recurrence relation

$$\Theta^N_{i,j+1} = \Theta^N_{i+1,j} + X_{AB}\Theta^N_{ij} \tag{3.44}$$

is obtained.

Note that the recurrence relations operate solely within some Cartesian direction (here: x-direction), thus indices for the remaining (untouched) directions have been omitted. The direction of the recurrence is implicitly stated by the prefactors (X_{IJ}). Additionally, and contrary to the kinetic-energy and overlap integrals, the Coulomb integrals depend on the parameter N. It is introduced by the recursion seeds, which are given by the scaled Boys function of order N.

$$\Theta^{N}_{\substack{00\\00\\00}} = \frac{2\pi}{p} K^{xyz}_{ab} F_N(pR^2_{PC}) \tag{3.45}$$

Here, the one dimensional pre-exponential factors have been abbreviated to (see eqn. (3.9)).

$$K^{xyz}_{ab} = K^{x}_{ab} K^{y}_{ab} K^{z}_{ab} \tag{3.46}$$

Two-Electron Integrals

Similarly to the one-electron Coulomb integrals, the two-electron integrals are expressed in terms of the auxiliary integrals $\Theta^{N}_{ijkl;mnop;qrst}$. Again, N is connected to the Boys function and the subscripts specify the angular momenta of the integral. The zeroth order in N integrals are the two-electron Coulomb integrals

$$\Theta^{0}_{\substack{ijkl\\mnop\\qrst}} = \langle ac|bd\rangle \tag{3.47}$$

The recursion seeds are given by the scaled Boys function of order N.

$$\Theta^{N}_{\substack{0000\\0000\\0000}} = \frac{2\pi^{5/2}}{pq\sqrt{p+q}} K^{xyz}_{ab} K^{xyz}_{cd} F_N(\alpha R^2_{PQ}) \tag{3.48}$$

In principle, recursion relations for each index can be obtained from the original Obara-Saika recurrence relation

$$\begin{aligned}\Theta^{N}_{i+1,jkl} = X_{PA}\Theta^{N}_{ijkl} - \frac{\alpha}{p} X_{PQ}\Theta^{N+1}_{ijkl} + \frac{i}{2p}\left(\Theta^{N}_{i-1,jkl} - \frac{\alpha}{p}\Theta^{N+1}_{i-1,jkl}\right)\\ + \frac{j}{2(p+q)}\Theta^{N+1}_{i,j-1,kl} + \frac{k}{2p}\left(\Theta^{N}_{ij,k-1,l} - \frac{\alpha}{2p}\Theta^{N+1}_{ij,k-1,l}\right) + \frac{l}{2(p+q)}\Theta^{N+1}_{ijk,l-1}\end{aligned} \tag{3.49}$$

Here, q and Q are the equivalents of p and P, related to G_c and G_d. The reduced exponent α is

$$\alpha = \frac{pq}{p+q} \tag{3.50}$$

Using index substitution and renaming variables leads to the recurrence relation for the j index.

$$\Theta^N_{i,j+1,kl} = X_{QC}\Theta^N_{ijkl} + \frac{\alpha}{q}X_{PQ}\Theta^{N+1}_{ijkl} + \frac{j}{2q}\left(\Theta^N_{i,j-1,kl} - \frac{\alpha}{q}\Theta^{N+1}_{i,j-1,kl}\right) + \frac{i}{2(p+q)}\Theta^{N+1}_{i-1,jkl} + \frac{l}{2q}\left(\Theta^N_{ijk,l-1} - \frac{\alpha}{q}\Theta^{N+1}_{ijk,l-1}\right) + \frac{k}{2(p+q)}\Theta^{N+1}_{ij,k-1,l} \tag{3.51}$$

Similar equations can be found for the k and l indices, respectively. Using these equations leads to rather complicated expressions, as each recurrence involves up to eight contributions and recurrences in every index. Fortunately, a four step ansatz exists [35–37], using simpler recurrence relations.

By setting $j = k = l = 0$ in (3.49), the vertical recurrence relation for the i-index is obtained.

$$\Theta^N_{i+1,000} = X_{PA}\Theta^N_{i,000} - \frac{\alpha}{p}X_{PQ}\Theta^{N+1}_{i,000} + \frac{i}{2p}\left(\Theta^N_{i-1,000} - \frac{\alpha}{p}\Theta^{N+1}_{i-1,000}\right) \tag{3.52}$$

Using translational invariance (compare (3.24)), increments in the j-index can be generated using the electron transfer recursion.

$$\Theta^N_{i,j+1,00} = -\frac{bX_{AB} + dX_{CD}}{q}\Theta^N_{i,j,00} + \frac{i}{2q}\Theta^N_{i-1,j,00} + \frac{j}{2q}\Theta^N_{i,j-1,00} - \frac{p}{q}\Theta^N_{i+1,j,00} \tag{3.53}$$

Similarly to (3.52), a different electron transfer can be obtained by setting $l = k = 0$ in (3.51).

$$\Theta^N_{i,j+1,00} = X_{QC}\Theta^N_{ij00} + \frac{\alpha}{q}X_{PQ}\Theta^{N+1}_{ij00} + \frac{i}{2(p+q)}\Theta^{N+1}_{i-1,j00} + \frac{j}{2q}\left(\Theta^N_{i,j-1,00} - \frac{\alpha}{q}\Theta^{N+1}_{i,j-1,00}\right) \tag{3.54}$$

Finally, increments for the k- and l-index are generated by the horizontal recursions (compare (3.14)).

$$\Theta^N_{i,j,k+1,l} = \Theta^N_{i+1,j,k,l} + X_{AB}\Theta^N_{i,j,k,l} \tag{3.55}$$

$$\Theta^N_{i,j,k,l+1} = \Theta^N_{i,j+1,k,l} + X_{CD}\Theta^N_{i,j,k,l} \tag{3.56}$$

4. Symmetry In Two-Electron Integrals And Batches

Generally put, the bottleneck of the assembly of the Fock matrix is the number of two-electron integrals to be calculated and contracted with the density matrix. They scale with the fourth power of the basis functions present in the system of interest. One way to reduce the number of integrals to be calculated is to take symmetries into account. This reduces the number of significant integrals by about an factor of eight.

4.1. Two-Electron Integral Symmetry

The general integral over PCG's is of the form

$$\langle pr|qs\rangle = \int G_p(r_1)G_r(r_2)\frac{1}{r_{12}}G_q(r_1)G_s(r_2)dr_{12} \tag{4.1}$$

There are three symmetries leaving this integral invariant. Clearly, the interchange of the basis functions depending on r_1, $G_p(r_1)$ and $G_q(r_1)$, does not change the value of the integral.

$$\langle pr|qs\rangle = \langle qr|ps\rangle \tag{4.2}$$

This holds equivalently for the functions depending on r_2.

$$\langle pr|qs\rangle = \langle ps|qr\rangle \tag{4.3}$$

Finally, the two-electron integrals are invariant with respect to particle exchange, giving

$$\langle pr|qs\rangle = \langle rq|sq\rangle \tag{4.4}$$

Each of these symmetries only leads to equivalent integrals if certain inequalities concerning the involved functions are met. For the particle one symmetry, interchange of the bra_1 and ket_1 functions only leads to a symmetry equivalent integral if

$$p \neq q \tag{4.5}$$

In integrals where $p = q$, particle one exchange just produces the same integral again, $\langle pr|ps\rangle \rightarrow \langle pr|ps\rangle$. Obviously, this also holds for the other two symmetries as well. Particle two symmetry, or exchange of the bra_2 with the ket_2 function, only gives a

symmetry-equivalent integral if

$$r \neq s \tag{4.6}$$

Finally particle exchange symmetry is met if

$$p \neq r \quad \text{and } q \neq s \tag{4.7}$$

Each of these symmetries, if present, doubles the number of symmetry equivalent integrals. The total number n_{se} of symmetry equivalent integrals for an index combination with s symmetries present is just

$$n_{se} = 2^s \tag{4.8}$$

In the integral $\langle pp|pp\rangle$, no symmetry is present and $n_{se} = 1$. No symmetry equivalent integrals exist. On the other hand, in the integral $\langle pr|qs\rangle$, all three symmetries are present, $n_{se} = 8$ and seven symmetry equivalent integrals can be found. An example for an integral with one symmetry present is $\langle pr|pr\rangle$. Particle exchange gives the one symmetry equivalent integral as $\langle rp|rp\rangle$.

Integral 'classes' with all three symmetries present will be termed as symmetric. Asymmetric classes are then the ones having one or more symmetries broken, respectively.

The idea now is to exploit this during the integral evaluation. Instead of calculating all n_{se} symmetry-equivalent integrals, only one 'non-redundant' integral of the particular class is calculated. The exact number of non-redundant two-electron integrals in a system with n basis functions is

$$N_{nr} = \frac{1}{8}(n^4 + 2n^3 + 3n^2 + 2n) \tag{4.9}$$

Taking this into account, of all the n^4 integrals, only about one eighth have to be calculated.

4.2. Symmetry in Batches of Two-Electron Integrals

As explained in section 3.1.1, the n basis functions are grouped into shells. The integrals arising from some tuple of shells are called shell batches. Now the symmetry properties of the two-electron integrals can equally be applied to four-tuple shell batches. Introducing a p-shell to be holding the three p-functions p_x, p_y and p_z, and an s-shell to be holding one s-function, some $psss$-batch will consist of the integrals

$$p_i s_j s_k s_l \equiv \left\{ \begin{pmatrix} p_{i,x} \\ p_{i,y} \\ p_{i,z} \end{pmatrix} s_j s_k s_l \right\} \equiv \left\{ \begin{matrix} \langle p_{i,x} s_j | s_k s_l \rangle \\ \langle p_{i,y} s_j | s_k s_l \rangle \\ \langle p_{i,z} s_j | s_k s_l \rangle \end{matrix} \right\}$$

Particle exchange gives the three integrals $\langle s_j p_{i,x} | s_l s_k \rangle$, $\langle s_j p_{i,y} | s_l s_k \rangle$ and $\langle s_j p_{i,z} | s_l s_k \rangle$, which are contained in the shell batch $s_j p_x s_l s_k$.

This is generally applicable, the rules being identical to those for the two-electron integrals, replacing functions with shells. The total number of non-redundant shell batches is also given by equation (4.9), substituting the number of basis functions n with the number of shells. The set of non-redundant shell batches holds all non-redundant integrals. However, some redundant integrals are reintroduced if the tuple is asymmetric. In the $p_i p_i s_i s_i$ batch for example the particle exchange symmetry (4.7) is broken. The batch contains, among others, the integrals $\langle p_{i,x} p_{i,y} | s_i s_i \rangle$ and $\langle p_{i,y} p_{i,x} | s_i s_i \rangle$ which are symmetrically identical. There is no simple formula to calculate the number of redundancies reintroduced this way. Table 4.1 shows the percentage of redundant integrals introduced in the set of non-redundant batches for some model systems. Their number clearly decreases with increasing molecule and/or basis size to a negligible amount. Even for high percentages however, the technical benefits from calculating integrals as batches outweigh the overhead introduced (see section 12.1.1).

	cc-pVDZ	cc-pVTZ	cc-pVQZ	cc-pV5Z	cc-pV6Z
CH_4	5.19	5.24	4.42	3.46	2.76
C_2H_6	3.13	3.18	2.70	2.12	1.70
C_4H_{10}	1.74	1.78	1.52	1.20	0.96
C_6H_{14}	1.21	1.23	1.05	0.83	0.67
C_8H_{18}	0.92	0.94	0.81	0.64	0.51

Table 4.1.: Percentage of redundant integrals in the set of non-redundant batches

Concluding, the exploitation of integral symmetries by omitting the calculation of redundant integrals can be done in terms of non-redundant shell batches, the overhead reintroduced being negligible.

5. Density Contraction

In Hartree-Fock theory, the two-electron part **G** of the Fock matrix is calculated by contraction of the density matrix **D** with the two-electron integrals $\langle ij|kl\rangle$

$$G_{ij} = \sum_{kl} D_{kl} \left[\langle ik|jl\rangle - 0.5 \langle ik|lj\rangle\right] \tag{5.1}$$

This step is of the order of integrals, making an efficient implementation mandatory. As the integration is performed on the basis of integral batches, the approaches to density contraction presented in this work are also in terms of a batchwise contraction.

5.1. Integral Contributions

Central aspect of the ansatz presented here is the exploitation of symmetry equivalent integrals. As has been shown before in section 4.1, considering only non-redundant integrals reduces the number of integrals to be taken into account by about an factor of eight. Obviously, reducing the number of integrals also reduces the amount of effort to be put into contracting them with the density, given an effective strategy can be found to be applied to eqn. (5.1). Starting point in discussing the density contraction is the analysis of the contributions of symmetry-equivalent integrals to the Fock matrix. In doing so, symmetric integrals have to be distinguished from asymmetric ones.

5.1.1. Symmetric Integrals

For each integral with four different basis functions $\langle pr|qs\rangle$, eight symmetry equivalent ones exist. This corresponds to 16 contributions of the same integral to **G**. Of these, eight arise from the first integral of equation (5.1), the other eight from the second, respectively.

$$\begin{aligned}
G_{pq} &+= D_{rs} \langle pr|qs\rangle & G_{ps} &-= 0.5 D_{rq} \langle pr|qs\rangle \\
G_{pq} &+= D_{sr} \langle ps|qr\rangle & G_{pr} &-= 0.5 D_{sq} \langle ps|qr\rangle \\
G_{qp} &+= D_{rs} \langle qr|ps\rangle & G_{qs} &-= 0.5 D_{rp} \langle qr|ps\rangle \\
G_{qp} &+= D_{sr} \langle qs|pr\rangle & G_{qr} &-= 0.5 D_{sp} \langle qs|pr\rangle \\
G_{rs} &+= D_{pq} \langle rp|sq\rangle & G_{rq} &-= 0.5 D_{ps} \langle rp|sq\rangle \\
G_{rs} &+= D_{qp} \langle rq|sp\rangle & G_{rp} &-= 0.5 D_{qs} \langle rq|sp\rangle \\
G_{sr} &+= D_{pq} \langle sp|rq\rangle & G_{sq} &-= 0.5 D_{pr} \langle sp|rq\rangle \\
G_{sr} &+= D_{qp} \langle sq|rp\rangle & G_{sp} &-= 0.5 D_{qr} \langle sq|rp\rangle
\end{aligned}$$

Using the symmetry of the density matrix $D_{ij} = D_{ji}$, the eight contributions on the left can already be reduced to four, i.e. $G_{pq} \mathrel{+}= 2D_{rs} \langle pr|qs\rangle$, and so on. Additionally, again using the symmetry of $\mathbf{D}$, the contributions to some element G_{ij} are the same as to the element G_{ji}. This can be exploited by considering only one of the two contributions. By doing so for all integrals, contracting to some temporary matrix $G^{'}$, some contributions will be added to $G^{'}_{ij}$, and some to $G^{'}_{ji}$. The exact matrix element can then be retrieved by

$$G_{ij} = G_{ji} = G^{'}_{ij} + G^{'}_{ji} \tag{5.2}$$

The actual contributions of all symmetry-equivalent integrals $\{\langle pr|qs\rangle\}$ to be calculated then are

$$\begin{aligned} G_{pq} &\mathrel{+}= 2.0D_{rs} \langle pr|qs\rangle & G_{ps} &\mathrel{-}= 0.5D_{rq} \langle pr|qs\rangle \\ G_{rs} &\mathrel{+}= 2.0D_{pq} \langle pr|qs\rangle & G_{pr} &\mathrel{-}= 0.5D_{qs} \langle pr|qs\rangle \\ & & G_{rq} &\mathrel{-}= 0.5D_{ps} \langle pr|qs\rangle \\ & & G_{qs} &\mathrel{-}= 0.5D_{pr} \langle pr|qs\rangle \end{aligned} \tag{5.3}$$

The index combinations in $\mathbf{D}$ and $\mathbf{G}$ correspond to the unique two-tuples which can be built from four indices, neglecting the order. For each contribution, the indices in $\mathbf{D}$ and $\mathbf{G}$ are disjunct, respectively.

5.1.2. Integrals with Broken Symmetries

In the same way, contributions for integrals with broken symmetries can be obtained. They are different to the ones obtained for the symmetric integrals $\langle pr|qs\rangle$. The class of integrals having only the particle exchange symmetry, $\langle pr|pr\rangle = \langle rp|rp\rangle$, gives the following contributions to $\mathbf{G}$

$$\begin{aligned} G_{pp} &\mathrel{+}= D_{rr} \langle pr|pr\rangle & G_{pr} &\mathrel{-}= 0.5D_{rp} \langle pr|pr\rangle \\ G_{rr} &\mathrel{+}= D_{pp} \langle pr|pr\rangle & & \end{aligned} \tag{5.4}$$

For reasons to be made clear later in section 13.2.2, the connection of these contributions and the contributions of a symmetric integral shall be shown in detail for this particular class of integrals. Application of the symmetry breaking $q \to p$ and $s \to r$ to the contributions of the symmetric integral (5.3) gives

$$\begin{aligned} G_{pp} &\mathrel{+}= 2.0D_{rr} \langle pr|pr\rangle & G_{pr} &\mathrel{-}= 0.5D_{rp} \langle pr|pr\rangle \\ G_{rr} &\mathrel{+}= 2.0D_{pp} \langle pr|pr\rangle & G_{pr} &\mathrel{-}= 0.5D_{pr} \langle pr|pr\rangle \\ & & G_{rp} &\mathrel{-}= 0.5D_{pr} \langle pr|pr\rangle \\ & & G_{pr} &\mathrel{-}= 0.5D_{pr} \langle pr|pr\rangle \end{aligned}$$

As these contributions have to be 'corrected' using (5.2), the diagonal elements on the left are doubled, and the off-diagonal elements on the right can be factored.

$$\begin{aligned} G_{pp} &+= 4.0 D_{rr} \langle pr|pr\rangle \qquad G_{pr} -= 2.0 D_{rp} \langle pr|pr\rangle \\ G_{rr} &+= 4.0 D_{pp} \langle pr|pr\rangle \end{aligned} \tag{5.5}$$

Comparing this to the actual contributions (5.4), (5.5) gives the correct contributions with respect to the indices. The numerical value however, is off by an factor four. This factor four is connected to the symmetry of the integrals. For the $\langle pr|pr\rangle$ class of integrals the number of the symmetry equivalent integrals $n_{se}^{prpr} = 2$ and for the $\langle pr|qs\rangle$ class $n_{se}^{prqs} = 8$. The fraction of the two is just the missing factor four. In other words, the contributions of the $\langle pr|pr\rangle$ class can be emulated by the formulas obtained for the symmetric $\langle pr|qs\rangle$ class, by taking the missing symmetry into account. Here the correction factor would be $f_s = \frac{1}{4}$.

In fact, this holds for all asymmetric classes of integrals. Their contributions can be calculated via the formulas (5.3), correcting them for symmetries using

$$f_s = \frac{2^s}{n_{se}^{prqs}} = \frac{2^s}{8} \tag{5.6}$$

Here, s is the number of symmetries present in the class of interest.

5.2. Shell Batch Contributions

5.2.1. Symmetric Batches

In a symmetric batch $prqs$ all shells, and therefore also the basis functions, are different. Accordingly, the integrals appearing in this shell batch are built from different basis functions and are all symmetric. The eqns. (5.3) can be applied to all integrals of this batch.

5.2.2. Batches with Broken Symmetries

In an asymmetric batch, the broken symmetry might lead to integrals from different symmetry classes. This happens if the symmetry broken shells are of a size larger than one. Consider for example the $p_i s_j p_i s_j$-batch. In terms of batch symmetries, it is invariant only under the particle exchange symmetry. However, due to the equality of the p-shells, the functions on these indices can be equal or unequal. If they are unequal, symmetry is reintroduced, i.e. the integral $\langle p_{i,x} s_j | p_{i,y} s_j\rangle$ is invariant with respect to the interchange of functions on particle one (the p-functions), in addition to the particle exchange symmetry of the batch. On the other hand, if the two p functions are equal $\langle p_{i,x} s_j | p_{i,x} s_j\rangle$, the integral is of the symmetry of the batch.

Nevertheless, as shown in section 5.1.2, the contributions by the asymmetric integrals can be calculated in a general way by taking the correction (5.6) into account. Thus,

in a first ansatz, the strategy to calculate the contributions from an asymmetric batch is similar to the one for symmetric batches. The only change being to calculate the correction for the particular integral. This might lead to different correction factors within a certain integral batch.

Additionally, reintroduction of symmetry also reintroduces redundancies. In the above example, besides the integral $\langle p_{i,x}s_j|p_{i,y}s_j\rangle$, the symmetry equivalent $\langle p_{i,y}s_j|p_{i,x}s_j\rangle$ is present in the batch. By the above ansatz, both are treated as if 'non-redundant', and their contributions are considered twice. Therefore, in addition to the correction (5.3), a second correction has to be applied to account for redundant integrals. Generally, each reintroduced symmetry doubles the number of symmetrically equivalent integrals in a given batch. The correction to account for non-redundant integrals then is

$$f_{nr} = \frac{1}{2^r} \tag{5.7}$$

Where r is the number of reintroduced symmetries. The total correction for some integral in an asymmetric batch is then given by

$$f = f_s \cdot f_{nr} = \frac{2^s}{82^r} \tag{5.8}$$

Again, s is the number of symmetries of the integral.

Fortunately, it can be shown that the product of these two corrections is constant for a given batch. Consider some batch with s_b symmetries. Reintroducing r symmetries in an integral gives the number of symmetries s for the integral as

$$s = s_b + r \tag{5.9}$$

The general correction factor for some integral (5.8) then becomes

$$f = \frac{2^{s_b+r}}{82^r} = \frac{2^{s_b}}{8} = f_{s,batch} \tag{5.10}$$

The correction for each integral in an asymmetric batch is just the symmetry correction f_s, based on the symmetries of the batch, regardless of the symmetry and/or redundancy of the individual integral.

Putting the above together, contracting the density with the integrals can be done in a general way in terms of non-redundant integral batches. After calculation of the integrals, the symmetry of the batch is used to calculate the correction (5.10). All integrals can then be contracted in the same way using (5.3), including the appropriate correction.

6. Basis Transformations

As shown in section 2.2, by expanding the MO's in terms of a finite set of basis functions, the Hartree-Fock equation can be formulated as a rather simple matrix eigenvalue equation. Over the years, a lot of effort has been put in determining the explicit form of these basis functions. There will be no exhausting overview given on this topic, a detailed overview can be found in [24]. Various compromises lead to the ansatz used in this work. In a general sense, atomic orbitals (AO) are used, meaning that atom-centered basis functions are provided for each atom of some molecule. The nature of these atomic basis functions is based on the solutions for the one-electron central field problem. Conceptually this results in a separation of the atomic wave function in a radial and angular part.

$$\psi(r,\phi,\theta)_{AO} = R(r) \cdot A(\phi,\theta) \tag{6.1}$$

6.1. Atomic Basis Functions

6.1.1. The Angular Basis

In the one-electron central field ansatz, the angular part of (6.1) is independent of the form of the central field. It is given by the eigenfunctions of the total angular momentum operator. Several expressions for these solutions exist. In this work, the real solid harmonics S_{lm} are used. More specific, they are expressed in terms of Cartesian directions.

$$S_{lm}(x,y,z) = N_{lm} \sum_{t=0}^{(l-|m|)/2} \sum_{u=0}^{t} \sum_{v=v_m}^{[|m|/(2-v_m)]+v_m} C_{tuv}^{lm}\, x^{2t+|m|-2(u+v)}\, y^{2(u+v)}\, z^{l-2t-|m|} \tag{6.2}$$

$$C_{tuv}^{lm} = (-1)^{t+v-v_m} \left(\frac{1}{4}\right)^t \binom{l}{t} \binom{l-t}{|m|+t} \binom{t}{u} \binom{|m|}{2v} \tag{6.3}$$

$$N_{lm} = \frac{1}{2^{|m|} l!} \sqrt{\frac{2(l+|m|)!(l-|m|)!}{2^{\delta_{0m}}}} \tag{6.4}$$

$$v_m = \begin{cases} 0 & m \geq 0 \\ \frac{1}{2} & m < 0 \end{cases} \tag{6.5}$$

$$|m| \leq l \tag{6.6}$$

Similarly to the PCG's, the solid harmonics are grouped into shells, depending on their angular momentum. As $|m| \leq l$, the number of functions in a shell of solid harmonics

with angular momentum l is

$$N_l^{sh} = 2l + 1 \tag{6.7}$$

For a given l, (6.2) can be easily evaluated. Doing so for e.g. $l = 2$, using the notation $S_{lm} = l_m$ and spectroscopic symbols ($l = 2$ corresponds to a d-shell), gives

$$d_{2+} = \frac{1}{2}\sqrt{3}(x^2 - y^2) \tag{6.8}$$
$$d_{1+} = \sqrt{3}xz \tag{6.9}$$
$$d_0 = \frac{1}{2}(3z^2 - r^2) \tag{6.10}$$
$$d_{1-} = \sqrt{3}yz \tag{6.11}$$
$$d_{2-} = \sqrt{3}xy \tag{6.12}$$

The sum of the exponents of the Cartesian directions in (6.2) is equal to l. Comparing this to the definition of a shell of PCG's in section 3.1.1, the Cartesian powers in (6.2) are closely related to the set (shell) of $N_l^{pcg} = \frac{1}{2}(l+1)(l+2)$ PCG's of a given l. The transformation (6.2) can be interpreted as a transformation of a shell of PCG's to a shell of solid harmonic Gaussians and written as a matrix vector multiplication. For the d-functions (6.8) to (6.12) this gives

$$\begin{pmatrix} d_{2+} \\ d_{1+} \\ d_0 \\ d_{1-} \\ d_{2-} \end{pmatrix} \exp(-ar_A^2) = \begin{pmatrix} \frac{\sqrt{3}}{2} & 0 & -\frac{\sqrt{3}}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \sqrt{3} & 0 & 0 \\ -\frac{1}{2} & 0 & -\frac{1}{2} & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & \sqrt{3} & 0 \\ 0 & \sqrt{3} & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x^2 \\ xy \\ y^2 \\ xz \\ yz \\ z^2 \end{pmatrix} \exp(-ar_A^2) \tag{6.13}$$

The explicit formulation is arbitrary, as the representation of the coefficient matrix depends on the ordering of the source and target functions, respectively.

Having an actual basis of PCG's with different angular momenta, exponents and centers, the whole basis can be transformed to solid harmonic Gaussians using the matrix vector multiplication ansatz of (6.13). Given the PCG's are ordered in terms of shells, the corresponding transformation matrix will be block diagonal, with entries given by (6.2). Defining the solid harmonic Gaussians as $|s_n\rangle = S_{lm}^n \exp(-a_n r_{A_n}^2)$ and using $|p_n\rangle = x^{i_n} y^{j_n} z^{k_n} \exp(-a_n r_{A_n}^2)$ for a PCG, this transformation of the whole basis can be written in a general way.

$$|s_n\rangle = \sum_i^M T_{in}^a |p_i\rangle \tag{6.14}$$

Here M is the size of the primitive basis and the T_{in}^a are the angular transformation coefficients. The basis of the $|s_i\rangle$'s will be termed as the angular transformed, or short angular basis. The basis of the PCG's as the primitive basis, respectively.

6.1.2. The Radial Basis

For the radial part, a functional form of $\exp(-ar^2)$ has been chosen due to ease of integration. As it does not resemble the charge distribution of an atomic system accurately, linear combinations of those Gaussians with different exponents a_λ on the same center A are used to account for that.

$$\phi_\mu(r, A) = \sum_{\lambda}^{N_r} C_{\lambda\mu} \exp(-a_\lambda r_A^2) \tag{6.15}$$

Again, this corresponds to a matrix vector multiplication. Equivalently, using $|r_\mu\rangle$ for an arbitrary radial transformed PCG, and $T^r_{i\mu}$ for the radial transformation coefficients, the radial transformation of an whole basis of PCG's with different angular momenta, exponents and centers can be written as

$$|r_\mu\rangle = \sum_{i}^{M} T^r_{i\mu} \ |p_i\rangle \tag{6.16}$$

Again, the form of the transformation matrix $\mathbf{T^r}$ depends on the ordering of the source and target functions. Ordering those with respect to the sets to be radial contracted leads to a block-diagonal transformation matrix $\mathbf{T^r}$. The coefficients $T^r_{i\mu}$ are given by (6.15). The basis of the $|r_\mu\rangle$'s will be referred to as the radial transformed, or short radial basis.

6.1.3. Atomic Basis Functions in Terms of Primitive Cartesian Gaussians

The atomic basis functions used in this work are set up from solid harmonics of a given l and m for the angular part, which are combined with the radially contracted functions of equation (6.15). The solid harmonics are in this equation expressed in terms of PCG's. The A_{nm} and the according Cartesian powers i_n, j_n and k_n are given by equations (6.2) to (6.6) (see equation (6.13)).

$$\begin{aligned}
\psi_{lm\mu}(r, A) &= \phi_\mu(r, A) \cdot S_{lm}(r, A) \\
&= \sum_{\lambda}^{N_r} C_{\lambda\mu} \exp(-a_\lambda r_A^2) \sum_{n}^{N_{pcg}} A_{nm} x_A^{i_n} y_A^{j_n} z_A^{k_n} \\
&= \sum_{\lambda}^{N_r} C_{\lambda\mu} \sum_{n}^{N_l^{pcg}} A_{nm} \left(x_A^{i_n} y_A^{j_n} z_A^{k_n} \exp(-a_\lambda r_A^2) \right) \\
&= \sum_{\lambda}^{N_r} C_{\lambda\mu} \sum_{n}^{N_l^{pcg}} A_{nm} \, G_{\substack{i_n \\ j_n \\ k_n}}(r, a_\lambda, A)
\end{aligned} \tag{6.17}$$

The AO's $\psi_{lm\mu}(r, A)$ are expressed as linear combinations of the PCG's (3.1).

Following the definitions (6.14) and (6.16), the full set of atomic basis functions $|AO_i\rangle = \psi_{l_i m_i \mu_i}(r, A_i)$ for a given system can be expressed in terms of the primitive basis. The obtained basis will be referred to as the atomic-orbital or AO-basis. Synonymously, the term radial angular transformed basis shall be used.

$$|AO_i\rangle = \sum_j^{M'} T_{ji}^a \sum_k^M T_{kj}^r |p_k\rangle = \sum_k^M \sum_j^{M'} T_{kj}^r T_{ji}^a |p_k\rangle = \sum_k^M T_{ki}^{ra} |p_k\rangle \tag{6.18}$$

In this formulation, the radial transformation is performed 'first'. The following angular transformation transforms the M' radial basis functions to the AO basis. This defines the dimensions and structures of the involved transformation matrices $\mathbf{T^a}$ and $\mathbf{T^r}$. The order of transformations can be interchanged, i.e. performing the angular transformation first. In doing so the form of the intermediate transformation matrices changes, the overall transformation matrix $\mathbf{T^{ra}}$ however stays the same.

Of course the transformation can be applied one at a time.

$$|AO_i\rangle = \sum_j^{M'} T_{ji}^a \left(\sum_k^M T_{kj}^r |p_k\rangle \right) = \sum_j^{M'} T_{ji}^a |r_j\rangle \tag{6.19}$$

6.2. Explicit Expressions for Basis Transformations

6.2.1. Two-Electron Integrals

The two-electron integral in some transformed basis $\{|a\rangle\}$ of size n

$$\langle ab|cd\rangle = \iint a(1)b(2)\frac{1}{r_{12}}c(1)d(2)dr_1 dr_2 \tag{6.20}$$

can be expressed in terms of the untransformed basis by inserting the respective transformation

$$|a\rangle = \sum_i^m C_{ia} |i\rangle \tag{6.21}$$

for each function. Here m is the basis size of the untransformed basis, and the C_{ia} are the transformation coefficients.

$$\begin{aligned} \langle ab|cd\rangle &= \iint \sum_i^m C_{ia} i(1) \sum_j^m C_{jb} j(2) \frac{1}{r_{12}} \sum_k C_{kc} k(1) \sum_l^m C_{ld} l(2) dr_1 dr_2 \\ &= \sum_i^m C_{ia} \sum_j^m C_{jb} \sum_k^m C_{kc} \sum_l^m C_{ld} \langle ij|kl\rangle \end{aligned} \tag{6.22}$$

This holds for the transformation from the primitive basis to radial-, angular- or radial-angular-transformed bases, as well as for the transformation of the AO-basis to the MO-basis.

Complexity

Technically, this is a rather demanding operation, scaling with the fifth power of the basis functions n. The transformation (6.22) can be performed one index at a time. The transformation of the ket2 index d for example is given by

$$\langle ij|kd\rangle = \sum_{l}^{m} C_{dl} \langle ij|kl\rangle \tag{6.23}$$

There are m^3n target integrals $\langle ij|kd\rangle$. To calculate each of those, m-operations are needed, giving the total cost as $\mathscr{O}(m^4n)$. The other indices scale similarly, shifting powers from m to n. The exact scaling is $\mathscr{O}(nm^4 + n^2m^3 + n^3m^2 + n^4m)$. As $m = xn$, with x being a constant depending on the exact type of the transformation, the total scaling is of $\mathscr{O}(n^5)$ times a constant depending on x.

Additionally, all m^4 untransformed integrals $\langle ij|kl\rangle$ are needed to calculate each of the n^4 transformed integrals. Storing and handling them turns out to be rather demanding in terms of system resources, even for small systems. This especially holds for the MO-transformation, where the sums run over the size of the whole AO-basis m.

For the radial and angular transformation, on the other hand, this formal scaling is drastically reduced. As mentioned in section 6.1.3, these two transformations operate on a subset of basis functions. As a consequence, the sums in (6.22) no longer run over the whole untransformed basis size m, but rather over the size of the subset. Effectively, this reduces the formal scaling by an order of magnitude to $\mathscr{O}(n^4)$. There is no simple formula to show this. However the idea shall be sketched by transforming a hypothetical basis consisting solely of primitive d-shells to a basis of the corresponding solid harmonics. The transformation of a d-shell is explicitly given in (6.13). Generally, including coefficients equal to zero, six primitive d-functions are needed to calculate each solid harmonic. Again transforming the last index, for each integral in (6.23) only six functions are needed.

$$\langle ij|kd_{sh}\rangle = \sum_{l}^{6} c_{ld} \langle ij|kd_{cart}\rangle \tag{6.24}$$

There are still m^3n target integrals, however now each of these needs only a constant amount of operations (six). The exact scaling for this index is of $\mathscr{O}(6m^3n)$. The sums in the transformation of the other indices also scale with the constant factor six instead of the basis size m. Overall this brings down the formal scaling by one order of magnitude from $\mathscr{O}(n^5)$ to $\mathscr{O}(n^4)$. Furthermore, the transformation can be blocked, as only a subset of the untransformed integrals is needed for a particular transformed one, reducing the number of integrals to be held in memory simultaneously.

The same argument holds for a generalization for arbitrary angular or radial transformations, effectively changing the number of elements in the sums from m to a constant. As these constants depend on the actual transformation (angular momentum and radial contraction length), the exact scaling depends on the system (molecule, basis set). Nevertheless the constants do not depend on m, bringing the overall formal scaling down to m^4.

6.2.2. The Density Matrix

The charge density for a closed-shell molecule with N electrons and $N/2$ occupied MO-orbitals is [22]

$$\rho(r) = 2\sum_{k}^{N/2} MO_k(r)MO_k(r) \tag{6.25}$$

The n MO's are expressed in terms of n AO's, the expansion coefficients being the eigenvectors of the Fock matrix in AO-basis.

$$|MO_l\rangle = \sum_{i}^{n} C_{il}\,|AO_i\rangle \tag{6.26}$$

The charge density (6.25) in the AO-basis then is

$$\begin{aligned}\rho(r) &= 2\sum_{k}^{N/2}\sum_{i}^{n} C_{ik}AO_i(r)\sum_{j}^{n} C_{jk}AO_j(r) \\ &= \sum_{ij}^{n}\left[2\sum_{k}^{N/2} C_{ik}C_{jk}\right]AO_i(r)AO_j(r)\end{aligned} \tag{6.27}$$

Giving the density matrix in AO-basis as

$$D_{ij}^{AO} = 2\sum_{k}^{N/2} C_{ik}C_{jk} \tag{6.28}$$

Or, using $\mathbf{C_{occ}}$ for the first $N/2$ 'occupied' columns of $\mathbf{C}$

$$\mathbf{D^{AO}} = \mathbf{C_{occ}}\mathbf{C_{occ}^{T}} \tag{6.29}$$

Similarly, the charge density can be expressed in the primitive basis, using the expansion of the AO's in the m PCG's (6.18).

$$\begin{aligned}\rho(r) &= 2\sum_{k}^{N/2}\sum_{i}^{n} C_{ik}\sum_{\mu}^{m} T^{ra}_{\mu i}p_\mu(r)\sum_{j}^{n} C_{jk}\sum_{\nu}^{m} T^{ra}_{\nu j}p_\nu(r) \\ &= \sum_{\mu\nu}^{m}\left(\sum_{i}^{n}\sum_{j}^{n} T^{ra}_{\mu i}\left[2\sum_{k}^{N/2} C_{ik}C_{jk}\right]T^{ra}_{\nu j}\right)p_\mu(r)p_\nu(r) \end{aligned} \tag{6.30}$$

Now, using (6.28), the density in the primitive basis can be written in terms of the AO-density.

$$D^{p}_{\mu\nu} = \sum_{i}^{n}\sum_{j}^{n} T^{ra}_{\mu i}D^{AO}_{ij}T^{ra}_{\nu j} \tag{6.31}$$

Or equivalently

$$\mathbf{D^p} = \mathbf{T^{ra}}\cdot\mathbf{D^{AO}}\cdot(\mathbf{T^{ra}})^T \tag{6.32}$$

Note that, following equation (6.18), the radial and angular transformations can be applied successively and in the order desired.

6.2.3. The Two-Electron Part of the Fock Matrix

The two-electron part G of the Fock matrix in the AO-basis is [22]

$$G^{AO}_{ac} = \sum_{b}^{n}\sum_{d}^{n} D^{AO}_{bd}\left[\langle ab|cd\rangle - \frac{1}{2}\langle ab|dc\rangle\right] \tag{6.33}$$

The AO-integrals can be expressed in the primitive basis using equation (6.22), with T^{ra}_{ij} as the expansion coefficients for the radial angular transformation.

$$G^{AO}_{ac} = \sum_{b}^{n}\sum_{d}^{n} D^{AO}_{bd}\left[\sum_{i}^{m} T^{ra}_{ia}\sum_{j}^{m} T^{ra}_{jb}\sum_{k}^{m} T^{ra}_{kc}\sum_{l}^{m} T^{ra}_{ld}\left(\langle ij|kl\rangle - \frac{1}{2}\langle ab|dc\rangle\right)\right] \tag{6.34}$$

$$= \sum_{i}^{m}\sum_{k}^{m} T^{ra}_{ia}\left[\sum_{j}^{m}\sum_{l}^{m}\left(\sum_{b}^{n}\sum_{d}^{n} T^{ra}_{jb}D^{AO}_{bd}T^{ra}_{ld}\right)\left(\langle ij|kl\rangle - \frac{1}{2}\langle ij|lk\rangle\right)\right]T^{ra}_{kc} \tag{6.35}$$

$$= \sum_{i}^{m}\sum_{k}^{m} T^{ra}_{ia}\left[\sum_{j}^{m}\sum_{l}^{m} D^{p}_{jl}\left(\langle ij|kl\rangle - \frac{1}{2}\langle ij|lk\rangle\right)\right]T^{ra}_{kc} \tag{6.36}$$

$$G^{AO}_{ac} = \sum_{i}^{m}\sum_{k}^{m} T^{ra}_{ia}G^{p}_{ik}T^{ra}_{kc} \tag{6.37}$$

Here, in (6.36) the expression for the AO-density in the primitive space (6.31) has been used. Similarly, comparing to (6.33), the quantity in braces in (6.36) is the two-electron part of the Fock operator in the primitive basis. Equation (6.37) then gives the relationship of $\mathbf{G}$ in the primitive with the radial angular transformed basis.

Equivalently, (6.37) can be written in matrix form.

$$\mathbf{G^{AO}} = (\mathbf{T^{ra}})^T \cdot \mathbf{G^{p}} \cdot \mathbf{T^{ra}} \tag{6.38}$$

Again, the transformations can also be applied one at a time, in the order desired.

7. Prescreening

Bottleneck of solving the Hartree-Fock equations is the number of two-electron integrals arising, as they formally scale to the fourth power of the system size n. Now integral prescreening uses the fact that, especially for large molecules, most of the two-electron integrals are small in magnitude. Neglecting integrals smaller than some allowed error leads to a drastic reduction of integrals to be calculated, without introducing considerable errors to the calculation. It can be shown that by including only significant integrals, the scaling with respect to the system size is reduced by two orders of magnitude to $\mathscr{O}(n^2)$ [38].

7.1. The Schwarz Inequality

The value of two-electron integrals in the form

$$\langle ab|cd\rangle = \iint a(1)b(2)\frac{1}{r_{12}}c(1)d(2)dr_1dr_2$$

can be estimated using the Schwarz inequality [14]. It gives with

$$|\langle ij|kl\rangle| \leq \sqrt{\langle ii|kk\rangle}\sqrt{\langle jj|ll\rangle} = S_{ik}S_{jl} \equiv S_{int}^{max} \tag{7.1}$$

an upper bound for the two-electron integrals based on the square roots of the diagonal elements $S_{ik} = \sqrt{\langle ii|kk\rangle}$. This upper bound can then be used to identify integrals of negligible value in advance and skip their calculation accordingly. In the following, the integrals S_{ik} shall be termed Schwarz integrals, taking the square root into account.

7.1.1. Batches of Integrals

As integrals are going to be calculated in terms of batches, (7.1) is not applied on individual integrals, but has to be extended to batches. One consequence of this is a loss of granularity. If only one integral in a batch of negligible integrals is of significant value, the whole batch has to be calculated. However, individual integral evaluation is technically severely more demanding than evaluating whole batches. The additional cost introduced by calculating batches which contain some negligible integrals is clearly outweighed by the effort saved in evaluating integral batches as a whole.

On the other hand, it can be shown that the computational effort to decide if any batch is negligible is essentially identical to deciding if a single integral can be skipped.

This shall be exemplified by considering the $p_i p_j s_k s_l$ shell batch, using arbitrary indices.

$$p_i p_j s_k s_l \equiv \left\{ \langle \begin{pmatrix} a \\ b \\ c \end{pmatrix} \begin{pmatrix} d \\ e \\ f \end{pmatrix} | (g) (h) \rangle \right\}$$

This particular batch consists of nine integrals, each of which can be estimated using (7.1). Explicitly doing so gives:

$$\begin{aligned}
|\langle ad|gh\rangle| &\leq \sqrt{\langle aa|gg\rangle} \cdot \sqrt{\langle dd|hh\rangle} = S_{ag} \cdot S_{dh} \\
|\langle ae|gh\rangle| &\leq \sqrt{\langle aa|gg\rangle} \cdot \sqrt{\langle ee|hh\rangle} = S_{ag} \cdot S_{eh} \\
|\langle af|gh\rangle| &\leq \sqrt{\langle aa|gg\rangle} \cdot \sqrt{\langle ff|hh\rangle} = S_{ag} \cdot S_{fh} \\
|\langle bd|gh\rangle| &\leq \sqrt{\langle bb|gg\rangle} \cdot \sqrt{\langle dd|hh\rangle} = S_{bg} \cdot S_{dh} \\
|\langle be|gh\rangle| &\leq \sqrt{\langle bb|gg\rangle} \cdot \sqrt{\langle ee|hh\rangle} = S_{bg} \cdot S_{eh} \\
|\langle bf|gh\rangle| &\leq \sqrt{\langle bb|gg\rangle} \cdot \sqrt{\langle ff|hh\rangle} = S_{bg} \cdot S_{fh} \\
|\langle cd|gh\rangle| &\leq \sqrt{\langle cc|gg\rangle} \cdot \sqrt{\langle dd|hh\rangle} = S_{cg} \cdot S_{dh} \\
|\langle ce|gh\rangle| &\leq \sqrt{\langle cc|gg\rangle} \cdot \sqrt{\langle ee|hh\rangle} = S_{cg} \cdot S_{eh} \\
|\langle cf|gh\rangle| &\leq \sqrt{\langle cc|gg\rangle} \cdot \sqrt{\langle ff|hh\rangle} = S_{cg} \cdot S_{fh}
\end{aligned}$$

Now the Schwarz integrals on the left of the estimate arise from combinations of functions of the p_i- and s_k-shells, and the ones on the right from the functions in the p_j- and s_l-shells, respectively. Additionally, each Schwarz integral arising from the p_i, s_k-set is combined with each from the p_j, s_l set. Thus an upper bound for all integrals in this batch can be obtained by the product of the maximum of the Schwarz integrals from the p_i, s_k set with the maximum of the Schwarz integrals from the p_j, s_l set, giving the Schwarz upper-bound for the batch, S^{max}_{batch} as

$$|\{\langle p_i p_j | s_k s_l \rangle\}| \leq S^{max}_{batch} = max(\{S_{ag}, S_{bg}, S_{cg}\}) \cdot max(\{S_{dh}, S_{eh}, S_{fh}\}) \tag{7.2}$$

The integrals of a batch can be generated from the functions in the shells by the Cartesian product. Similarly, the Schwarz integrals can be generated from the Cartesian product of the functions in the corresponding shells, $\{p_i\}$ x $\{s_k\}$ and $\{p_j\}$ x $\{s_l\}$, respectively.

The above can then be generalized to any batch $\langle IJ|KL\rangle$. The upper bound of the batch is the maximum of the Schwarz integrals arising from the Cartesian product of functions in the I- and K-shell times the maximum of the ones from the J- and L-shells.

$$|\{\langle IJ|KL\rangle\}| \leq S^{max}_{batch} = max(\{S_{\{I\}\mathrm{x}\{K\}}\}) \cdot max(\{S_{\{J\}\mathrm{x}\{L\}}\}) \tag{7.3}$$

As will be shown in section 11.1.2, the Schwarz integrals are evaluated once for a specific calculation. Now instead of storing the individual Schwarz integrals, the maxima of (7.3) are determined and stored instead. Compared to the full SCF calculation, the cost of this step is negligible (see section 11.1.2). During the calculation, each batch can then be estimated by comparing two numbers, the stored maxima. This is of the same computational cost as estimating a single integral.

7.2. Density Screening

As the two-electron integrals (7.1) contribute to the Fock matrix by being contracted with the density, determination of significant integrals via (7.3) is not sufficient. On the one hand, significant contributions might be neglected. On the other hand, small density elements can tip the scale, leading to an even higher number of negligible integral contributions. This is of particular interest in the scope of the differential density scheme (see section 11.4).

To arrive at a sufficient criterion, the elements of the density to be contracted with a certain integral have to be taken into account. These density elements are given by (5.3). For some symmetric integral $\langle ij|kl\rangle$, (7.1) has therefore to be extended to

$$\begin{aligned} |G(\langle ij|kl\rangle)| &\leq S_{int}^{max} \cdot max(\{2D_{ik}, 2D_{jl}, 0.5D_{ij}, 0.5D_{il}, 0.5D_{jk}, 0.5D_{kl}\}) \\ &\equiv S_{int}^{max} \cdot D_{int}^{max} \end{aligned} \tag{7.4}$$

For unsymmetrical integrals, the correction (5.6) has to be taken into account.

7.2.1. Batches of Integrals

According to Haeser et al. [14], equation (7.4) can be extended to integral batches. The idea is to get an upper bound for a batch of integrals by combining the maximum integral contribution (7.3) with the maximum density element associated with the actual batch. Similar to 7.1.1, the density elements in (7.4) are replaced by the maxima over the elements generated by the Cartesian product of the corresponding shells. The maximum density element for the batch is then

$$\begin{aligned} D_{batch}^{max} = max(\{&2 \cdot max(\{D_{\{I\}\mathrm{x}\{K\}}\}, \{D_{\{J\}\mathrm{x}\{L\}}\}), \\ &0.5 \cdot max(\{D_{\{I\}\mathrm{x}\{J\}}\}, \{D_{\{I\}\mathrm{x}\{L\}}\}, \{D_{\{J\}\mathrm{x}\{K\}}\}, \{D_{\{K\}\mathrm{x}\{L\}}\})\}) \end{aligned} \tag{7.5}$$

The upper bound for contributions to the Fock matrix for some batch $\langle IJ|KL\rangle$ is then just

$$|G(\langle IJ|KL\rangle)| \leq S_{batch}^{max} \cdot D_{batch}^{max} \tag{7.6}$$

Again, for unsymmetrical batches the correction (5.10) has to be taken into account.

Note that batches can be overestimated by this measure, as the maximum integral contribution S_{batch}^{max} and the maximum density element D_{batch}^{max} are not necessarily linked. However, as will be shown in section 11.1.5, this is of no practical concern.

Similarly to the application of the Schwarz inequality to batches of integrals, the maxima $max(\{D_{\{I\}\mathrm{x}\{K\}}\})$ can be determined in advance and used when required. Again, the computational effort to estimate one batch of integrals is equal to estimating one individual integral.

Condition (7.6) can now be used to identify integral batches which give significant contributions to the Fock matrix. A batch has to be calculated if

$$S_{batch}^{max} \cdot D_{batch}^{max} \geq t \tag{7.7}$$

where t is some threshold representing the maximum magnitude of contributions to be included.

7.3. The Differential Density Scheme

The differential density scheme [13, 14, 16] can be used to further increase the number of negligible integral batches. The Hartree-Fock equations have to be solved iteratively. The differential density is the difference of the density of the current and previous iteration.

$$\Delta\mathbf{D} = \mathbf{D}^i - \mathbf{D}^{i-1} \tag{7.8}$$

Now by using

$$\mathbf{D}^i = \mathbf{D}^{i-1} + \Delta\mathbf{D} \tag{7.9}$$

the contraction of integrals and density to the two-electron part $\mathbf{G}$ of the Fock matrix (5.1) can be reformulated in terms of the differential density.

$$\begin{aligned} G_{ij}(D^i) &= \sum_{kl} D_{kl}^i \left[\langle ik|jl\rangle - 0.5\,\langle ik|lj\rangle\right] \\ &= \sum_{kl} (D_{kl}^{i-1} + \Delta D_{kl}) \left[\langle ik|jl\rangle - 0.5\,\langle ik|lj\rangle\right] \\ &= \sum_{kl} D_{kl}^{i-1} \left[\langle ik|jl\rangle - 0.5\,\langle ik|lj\rangle\right] + \sum_{kl} \Delta D_{kl} \left[\langle ik|jl\rangle - 0.5\,\langle ik|lj\rangle\right] \\ G_{ij}(\mathbf{D}^i) &= G_{ij}(\mathbf{D}^{i-1}) + G_{ij}(\Delta\mathbf{D}) \end{aligned} \tag{7.10}$$

Incrementing the previous $\mathbf{G}^{i-1}$ by $\mathbf{G}(\Delta\mathbf{D})$ gives the current two-electron part of the Fock matrix.

Using this ansatz in combination with prescreening, an even higher amount of integrals can be neglected. The Schwarz inequality (7.1) gives a constant result for the magnitudes of the integrals of a certain system. Equivalently, especially near convergence, the density is almost constant, giving a constant amount of vanishing contributions. The differential density ansatz now exploits just this. As convergence is approached, more and more elements of the differential density tend to zero, thus increasing the amount of integrals which can be neglected.

Part II.

Implementation

8. Technical Details

8.1. Programs Used

The MOLPRO [10] program package has been used, in the version 2012.1.7. It has been compiled from the sources. The compiler used has been the GNU Compiler Collection, in version 4.9.2. MOLPRO has been configured using the default options, with the exception of the LAPACK/BLAS libraries. For these the libraries provided by INTELs Math Kernel Library [39] have been used, in version 11.0. In particular, the integral-direct Hartree-Fock SCF [37, 40] program was used. Only the basis sets of section 8.2.1 were used.

TURBOMOLE has been used in version 6.6 [11].

The QOL has been compiled with the GNU Compiler Collection, in version 4.9.2. For LAPACK/BLAS, the libraries provided by INTELs Math Kernel Library, in version 11.0 have been used. Additionally, the BOOST library [41], version 1.59 has been used.

Debian Wheezy has been used as operating system as well as all associated packages.

8.2. Basis Sets and Geometries

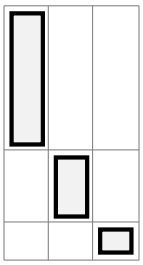
Great care has been taken to use identical inputs for all programs used. In the case of this work, the most sensitive inputs are basis sets and geometries.

8.2.1. Basis Sets

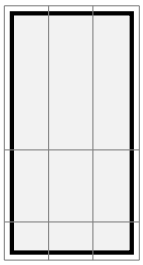
To assure the use of identical basis sets in different programs, a unified ansatz has been chosen for their generation. Source for the basis sets used has been the EMSL basis set exchange library [42, 43]. From this site, all basis sets used in this work have been obtained, in the DALTON [12] file format. This format has been chosen due to its ease of parsability. From these source files, basis set input files have then be generated, in the MOLPRO, TURBOMOLE and QOL formats, in an automated way. The naming conventions have been chosen according to the name given the basis set in the EMSL basis set exchange library.

Four different types of basis sets have been used, differentiated by their radial contraction pattern. The patterns included are depicted in figure 8.1. In segmented contractions (figure 8.1a), each primitive Gaussian contributes to only one contracted function. The Def2 [44–47] basis set were used, of double-, triple- and quadruple-zeta quality. The restrictions of the segmented basis sets are completely dropped in the generalized contractions (figure 8.1b), in which each primitive contributes to each contracted function. For

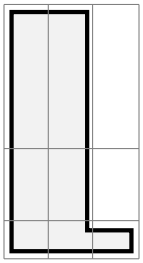
these bases atomic natural orbital basis sets have been used, in particular the roos-aug-dz-ano, roos-aug-tz-ano and ano-rcc [48–50] basis sets. Finally the correlation consistent basis sets cc-pV*Z have been used, from double- to hextuple-zeta quality [51–54]. They constitute a compromise of the segmented and general contraction patterns (figure 8.1c and figure 8.1d). Two flavours of these basis sets have been included in this work. In MOLPRO, the cc-pV*Z basis sets are included as depicted in figure 8.1c. In TURBOMOLE on the other hand, the contraction pattern of figure 8.1d is used [55,56]. In both programs, they are selected using the same keyword, i.e. cc-pvtz, although this results in the use of slightly different basis sets. In the EMSL basis set exchange library, the latter are available by selecting the use of 'optimized general contractions'. To distinguish between these two flavours, the correlation consistent basis sets using the pattern of figure 8.1c are referred to as cc-pV*Z, and the ones of figure 8.1d by cc-pV*Z-ogc.



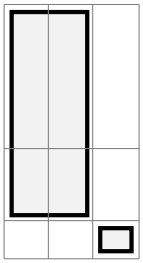
(a) Segmented.



(b) General.



(c) Correlation consistent.



(d) Correlation consistent, using optimized general contractions.

Figure 8.1.: Basis set types used, with respect to their radial contraction pattern.

8.2.2. Input Geometries

All geometries used in this work have been obtained with the TURBOMOLE program package. Geometry optimizations have been performed, using DFT with the functional BP86 [57–61] in the RI [62] approximation. The basis set used was TURBOMOLEs TZVP [46] basis.

Small deviations are introduced to SCF results by slightly different input geometries. Such slight changes are easily introduced by unit conversions, using differing constants

for the angstrom to Bohr conversion. To prevent this, the following route has been taken to generate identical input geometries for TURBOMOLE, MOLPRO and the QOL. The final geometries have been obtained from the TURBOMOLE calculations in xyz-format using `t2x -c`. These were then retransformed to Bohr using `x2t`. From this file, the final inputs for MOLPRO and the QOL were generated, resulting in identical geometry inputs for all programs investigated.

8.3. Computational Details

8.3.1. Computational Facilities

All calculations have been performed on hosts which are part of the computational facilities of the Institute for Theoretical Chemistry, University of Cologne. In each of those hosts, two sockets hold four Intel(R) Xeon(R) E5520 CPU's each, with a clock rate of 2.27 GHz. Figure 8.2 shows the CPU/cache topology of one of these hosts. All used hosts share this topology, although different amounts of RAM are installed. All hosts operate under the Debian/Wheezy operating system.

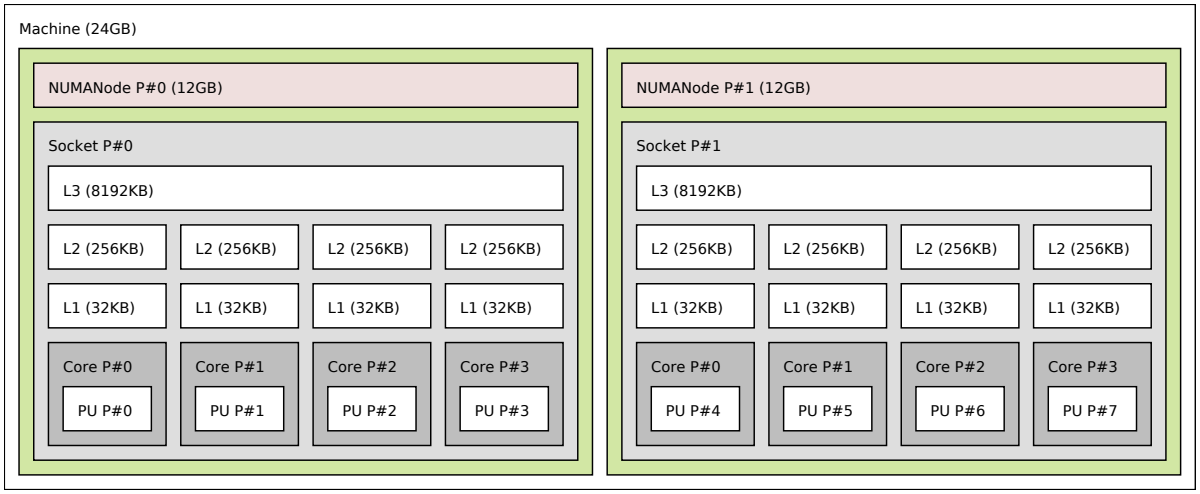


Figure 8.2.: Topology of one of the hosts used in this work.

8.3.2. Calculational Details

If not stated otherwise, all timings presented in this work have been generated in the same way. They were run on one of the hosts above, sequentially. For each calculation, the absence of interfering jobs has been assured.

9. Numerics

9.1. Numerical Errors

The source of numerical errors in computational calculations is the finite representation of floating point numbers. Generally, a floating point number is expressed in the form

$$x = (-1)^s \cdot m \cdot b^e \tag{9.1}$$

Beside the sign s, the first non-zero digits are stored in the mantissa m and the magnitude of the number is given by the exponential part b^e, with respect to some basis b. In principal, any floating point number can be expressed this way. In practice however, infinite mantissas can not be handled and have to be truncated. In modern computers, two main types of floating point numbers are available, of single (`float`) and double (`double`) precision. Of course the basis in which they are represented is base-2, i.e. $b = 2$. In the IEEE norm 754 [63], 64-bit are reserved for a `double`. One bit is needed for the sign, eleven bits are used for the exponent and the remaining 52 bits for the mantissa. In decimal numbers, the range spanned by such a number is from 10^{-308} to 10^{+308}, with about 15 to 17 significant digits for the mantissa.

Of the several sources of numerical errors, only the one arising from subtraction and addition of numbers of vastly differing magnitudes shall be discussed in detail. The general mechanism shall be exemplified using decimal numbers, and five digits for the mantissa. Before adding two numbers a and b, they have to be normalized to a common exponent.

$$\begin{aligned} a &= 1.2345 \cdot 10^5 \\ b &= 1.2345 \cdot 10^{-4} = 0.0000000012345 \cdot 10^5 \end{aligned}$$

Of course, the exact outcome of the addition is

$$a + b = \underline{1.2345}000012345 \cdot 10^5$$

However, as only five digits are available for the mantissa, the result is truncated. In this five digits representation, the value of b is completely lost, giving the result of $a + b$ as exactly a. If this would be the final result, the actual error is negligible, compared to the magnitudes involved. If used as an intermediate though, rather substantial errors might arise. Suppose the following equation:

$$-a + (a + b) \tag{9.2}$$

In the five digit representation, $(a+b)$ equates to exactly a, giving the total result as zero. Algebraically however, the result obviously is b. Contrarily, computing equation (9.2) as $(-a + a) + b$ gives, even in the five digit representation, the correct result b. Thus, addition and subtraction might break, if large magnitude differences are involved, the associativity law.

9.1.1. Absolute and Relative Errors

In analyzing numerical instabilities, the correct interpretation of arising deviations is mandatory. As it turns out, the obvious absolute error $abs = |x_0 - x|$ is not sufficient in such a classification. This is due to the fact that the magnitude of the numbers is not taken into account, which is shown in the following three examples, again using five significant digits.

$$|1.2345 \cdot 10^{+10} - 1.234\underline{6} \cdot 10^{+10}| = 1.11 \cdot 10^{8}$$
$$|1.2345 \cdot 10^{-10} - 1.234\underline{6} \cdot 10^{-10}| = 1.11 \cdot 10^{-12}$$
$$|1.2345 \cdot 10^{-10} - 1.\underline{3456} \cdot 10^{-10}| = 1.1111 \cdot 10^{-11}$$

Although the error in the mantissa is identical for the first two examples, the obtained absolute errors differ vastly, based on the magnitudes. In the third example, only one significant digit is identical in the two numbers, making them clearly more 'different' than the pairs above. The absolute error however appears to be comparable to that of example two and negligible compared to example one.

To account for that, the relative error $rel = \frac{|x_0 - x|}{x_0}$ is introduced. Application to the above example gives:

$$|1.2345 \cdot 10^{+10} - 1.234\underline{6} \cdot 10^{+10}|/1.2345 \cdot 10^{+10} = 8.1 \cdot 10^{-5}$$
$$|1.2345 \cdot 10^{-10} - 1.234\underline{6} \cdot 10^{-10}|/1.2345 \cdot 10^{-10} = 8.1 \cdot 10^{-5}$$
$$|1.2345 \cdot 10^{-10} - 1.\underline{3456} \cdot 10^{-10}|/1.2345 \cdot 10^{-10} = 8.1 \cdot 10^{-2}$$

Now the relative error for the first two examples is identical, which is in accord to their identical mantissas. The relative error of the third example is magnitudes higher, classifying the corresponding numbers as more different.

The negative decadic logarithm of the relative error can be interpreted as the first differing digit in the operands.

9.1.2. Numerical Analysis

Analysis and interpretation of numerical instabilities is not trivial. The tools and techniques used in this work shall be introduced here, using a real life example from the Obara-Saika context. As mentioned previously, severe errors might arise by the addition of numbers with largely different magnitudes. An example is the calculation of the

prefactor X_{PA} of the vertical recurrence relation (equation (3.52)).

$$X_{PA} = P_x - A_x \tag{9.3}$$

The center-of-charge coordinate P_x of two PCG's centered on A and B, with exponents a and b, is just

$$P_x = \frac{aA_x + bB_x}{a+b} \tag{9.4}$$

Now as P_x is needed not only to calculate X_{PA}, an approach would be to calculate P_x once, thus saving unnecessary computational resources. However, this gives rise to numerical noise in the calculation of X_{PA}, when using equation (9.3). Reducing equation (9.3) to the common denominator gives

$$X_{PA} = \frac{\boxed{aA_x} + bB_x\boxed{-aA_x} - bA_x}{a+b} = -\frac{b}{a+b}(A_x - B_x) \tag{9.5}$$

The term aA_x of equation (9.4) gets canceled. For large a, in calculating P_x singularly, this term however becomes of leading magnitude, throwing away significant digits of bB_x. For sufficiently large differences in magnitudes, the numerical information of bB_x is completely lost, giving an error in the calculation of X_{PA}.

This is summarized in table 9.1. For varying a, the results for X_{PA} using equation (9.3) and equation (9.5) are shown, to ten significant digits, in addition to the respective absolute and relative errors. B_x and b have been set to 1, and A_x to 2.

a	X_{PA}(9.3)	X_{PA}(9.5)	abs((9.5),(9.3))	rel((9.5),(9.3))
10^{-17}	$-1.00000000 \cdot 10^{+00}$	$-1.00000000 \cdot 10^{+00}$	$0.0 \cdot 10^{+00}$	$0.0 \cdot 10^{+00}$
10^{-01}	$-9.09090909 \cdot 10^{-01}$	$-9.09090909 \cdot 10^{-01}$	$1.1 \cdot 10^{-16}$	$1.2 \cdot 10^{-16}$
10^{+01}	$-9.09090909 \cdot 10^{-02}$	$-9.09090909 \cdot 10^{-02}$	$2.8 \cdot 10^{-17}$	$3.1 \cdot 10^{-16}$
10^{+02}	$-9.90099009 \cdot 10^{-03}$	$-9.90099009 \cdot 10^{-03}$	$1.6 \cdot 10^{-17}$	$1.6 \cdot 10^{-15}$
10^{+05}	$-9.99990000 \cdot 10^{-06}$	$-9.99990000 \cdot 10^{-06}$	$5.4 \cdot 10^{-17}$	$5.4 \cdot 10^{-12}$
10^{+07}	$-9.99999900 \cdot 10^{-08}$	$-9.99999900 \cdot 10^{-08}$	$6.7 \cdot 10^{-17}$	$6.7 \cdot 10^{-10}$
10^{+13}	$-9.99\underline{200722} \cdot 10^{-14}$	$-9.99\underline{999999} \cdot 10^{-14}$	$8.0 \cdot 10^{-17}$	$8.0 \cdot 10^{-04}$
10^{+16}	$\underline{0.00000000} \cdot 10^{+00}$	$\underline{-9.99999999} \cdot 10^{-17}$	$1.0 \cdot 10^{-16}$	$1.0 \cdot 10^{+00}$

Table 9.1.: Calculation of X_{PA} using the algebraically identical equations (9.3) and (9.5). Shown are explicit values, and their absolute and relative errors, for varying a.

Clearly, the two algebraically identical equations produce different results for increasing a. It shall be noted that the absolute error stays about constant, and relatively small, for all a. The relative error however increases with increasing a. Starting with $a = 10^2$, less and less significant digits are identical, giving a totally different result for $a = 10^{16}$.

Although the absolute errors are small, it shall be pointed out that in the Obara-Saika scheme, due to the massive number of appearing intermediates, even small errors

could propagate to substantially influence the final result. This is shown in detail in section 12.3.

The `boost::multiprecision` Library

For the calculation of X_{PA} two algebraically identical equations can be given, producing different numerical results. In this case, based on the analysis given in the text, equation (9.5) appears to give the correct result. Nevertheless, at this point, this can not be put to the test.

To analyze numerical issues properly, the `boost::multiprecision` library [41] has been used in this work. It provides with `mpfr_float_backend` an floating point type with an adjustable number of significant decimal digits. Using this type, calculations can be performed with the desired degree of accuracy, and easily compared to the results obtained with the standard floating point types. Due to `C++`'s powerful `template` mechanism, code maintenance is reduced to a minimum. Listing 9.1 exemplifies this for the above mentioned calculation of $-a + (a + b)$ versus $(-a + a) + b$. In this example the number of significant digits for the `mpfr_float_backend` type is set to 500, and the result is compared to the one obtained with the standard `double` type.

```
1  template <class T> void f() {
2    T a = 555e10, b = 555e-10;
3    cout << -a + (a + b) << endl;
4    cout << (-a + a) + b << endl;
5  }
6
7  int main (int argc, char* argv[]) {
8    f<number<mpfr_float_backend<500> > >();
9    f<double>();
10 }
11
12 // output:
13 // number<mpfr_float_backend<500> >
14 5.55e-08
15 5.55e-08
16 // double
17 0
18 5.55e-08
```

Listing 9.1: Use of the `boost::mpfr_float_backend` high accuracy type.

By templating the functions relevant to some calculation, using the high accuracy `boost` type becomes identical to the use of standard types, giving a powerful means to analyze possible numerical issues. In this case, using the `double` type, $a + b$ equates to a, due to the differences in magnitudes. Therefore $-a + (a + b)$ equates to zero, giving the algebraically wrong result. The `boost` type however produces the correct result.

Now this high accuracy type can be used to further analyze the two calculation pathways for the prefactor X_{PA}. In table 9.2 the relative error for the X_{PA} values obtained with the high accuracy type using 500 digits is shown, for equations (9.3) and (9.5),

respectively. For the purpose of comparison, the results obtained with the standard `double` type are repeated.

a	rel(boost<500>)((9.5),(9.3))	rel(double)((9.5),(9.3))
$1.0 \cdot 10^{-17}$	$2.4 \cdot 10^{-501}$	0.0
$1.0 \cdot 10^{-1}$	$2.7 \cdot 10^{-501}$	$1.2 \cdot 10^{-16}$
$1.0 \cdot 10^{2}$	$8.5 \cdot 10^{-500}$	$1.6 \cdot 10^{-15}$
$1.0 \cdot 10^{5}$	$7.5 \cdot 10^{-497}$	$5.4 \cdot 10^{-12}$
$1.0 \cdot 10^{7}$	$9.8 \cdot 10^{-495}$	$6.7 \cdot 10^{-10}$
$1.0 \cdot 10^{13}$	$2.4 \cdot 10^{-488}$	$8.0 \cdot 10^{-4}$
$1.0 \cdot 10^{16}$	$4.2 \cdot 10^{-486}$	1.0

Table 9.2.: Relative errors of X_{PA} obtained using the algebraically identical equations (9.3) and (9.5). Compared are the values of X_{PA} calculated with the high accuracy type using 500 digits and the standard `double`.

The relative error from the calculations using the high accuracy type shows the same increase as observed for the `double` type. With increasing a, more and more significant digits are lost. For $a = 10^{16}$, the accuracy of the `double` type is reduced to zero significant digits, loosing 16 digits compared to $a = 10^{-17}$. Using the high accuracy type, again 15 significant digits are lost in the same range.

However, as the high accuracy type still reproduces over 480 significant digits of X_{PA}, regardless of the equation used, it is well suited to determine which of the equations (9.3) or (9.5) gives the correct result. The result of this comparison is summarized in table 9.3, stating the relative errors of X_{PA} calculated with the high accuracy type using equation (9.3) and the results obtained with the standard `double` type, for both equation (9.3) and equation (9.5).

a	rel(boost<500>(9.3),double(9.3))	rel(boost<500>(9.3),double(9.5))
$1.0 \cdot 10^{-17}$	0.0	0.0
$1.0 \cdot 10^{-1}$	$1.2 \cdot 10^{-16}$	0.0
$1.0 \cdot 10^{2}$	$1.6 \cdot 10^{-15}$	0.0
$1.0 \cdot 10^{5}$	$5.5 \cdot 10^{-12}$	0.0
$1.0 \cdot 10^{7}$	$6.6 \cdot 10^{-10}$	0.0
$1.0 \cdot 10^{13}$	$8.0 \cdot 10^{-4}$	0.0
$1.0 \cdot 10^{16}$	1.0	$1.2 \cdot 10^{-16}$

Table 9.3.: Relative errors of X_{PA} obtained using the algebraically identical equations (9.3) and (9.5). Results for the standard `double` type are compared to the X_{PA} values obtained with the high accuracy type, using 500 digits and equation (9.3).

Obviously, and as obtained by manual inspection, equation (9.5) gives the correct result for X_{PA}, whereas the evaluation from the precalculated P_x (9.3) suffers from numerical instabilities.

10. Basis Sets and Iteration

To represent molecules, basis sets and the iteration over associated quantities, the class structures already present in the QOL have been used for most parts of the implementations presented in this work. They are part of the QOL [18] and will be not discussed in detail, only a short overview of how they are used will be given.

10.1. Molecules and Basis Sets

10.1.1. Input Format

Molecules and basis sets are initialized from text files. The format to specify the molecular geometry follows MOLPROs input specifications.

```
1 geometry = {
2 C     -0.865463     -1.35994      0
3 H     -0.174658     -3.31368      0
4 H     -0.174491     -0.382957     1.6921
5 H     -0.174491     -0.382957     -1.6921
6 H     -2.93784      -1.35995      0
7 };
```

Listing 10.1: Geometry specification for CH_4.

Coordinates are expected to be given in atomic units.

Basis sets are read from XML files, using a format specified within the QOL. They are generated from DALTON basis set files as described in section 8.2.1. Listing 10.2 shows the cc-pVDZ basis for hydrogen as an example.

```
1  <CartesianGaussianBasisEntry>
2    <Comment>
3      !  cc-pVDZ  EMSL  Basis Set Exchange Library   23/01/15 3:23
4      ! HYDROGEN       (4s,1p) -> [2s,1p]
5    </Comment>
6
7    <AngularMomentumGroup l="0">
8      <ZetaContraction
9        zeta="13.0100000"><Coefficients> 0.0196850 0.0000000
           </Coefficients>
10     </ZetaContraction>
11     <ZetaContraction
12       zeta="1.9620000"><Coefficients> 0.1379770 0.0000000
           </Coefficients>
13     </ZetaContraction>
14     <ZetaContraction
```

```
15        zeta="0.4446000"><Coefficients> 0.4781480 0.0000000
            </Coefficients>
16      </ZetaContraction>
17      <ZetaContraction
18        zeta="0.1220000"><Coefficients> 0.5012400 1.0000000
            </Coefficients>
19      </ZetaContraction>
20    </AngularMomentumGroup>
21
22    <AngularMomentumGroup l="1">
23      <ZetaContraction
24        zeta="0.7270000"><Coefficients> 1.0000000 </Coefficients>
25      </ZetaContraction>
26    </AngularMomentumGroup>
27  </CartesianGaussianBasisEntry>
```

Listing 10.2: cc-pVDZ basis for hydrogen

Note that there is no explicit reference (despite the comment) to the element type in the XML file. This is resolved internally by the QOL, the element type being deduced from the file name (here: H.xml).

10.1.2. Technical Representation

Within the QOL, molecules are represented by the class `Molecule`, and the system of interest by one of the `CGBTree_*` classes. Their usage is rather straightforward, the `Molecule` being constructed from the text file holding the molecular geometry, and the `CGBTree_*` classes from a string specifying the directory holding the xml files, relative to `~/QOLBasis`, and the molecule. The `CGBTree_*`'s are available in several flavours. Of interest is `CGBTree_Plain`, by which a basis of primitive Cartesian Gaussians is specified. The radial-angular-transformed bases of section 6.1.3 can be represented by `CGBTree_Contracted_AngularTransformed`, respectively. Listing 10.3 shows this for hydrogen in a cc-pVDZ basis, in the primitive- and radial-angular-transformed bases, respectively.

```
1  Molecule molecule("~/H.in");
2
3  CGBTree_Plain<double> pBasis(molecule, "emsl/cc-pvdz");
4
5  CGBTree_Contracted_AngularTransformed<double> tBasis(molecule,
     "emsl/cc-pvdz");
```

Listing 10.3: Initialization of different basis types

So far, only abstract representations of the system of interest have been generated.

10.2. Iterators

Several iterator classes are available to make use of the bases introduced above. They are templated, the basis type being the template argument, and can thus be used in a almost identical fashion for all basis types.

10.2.1. Basis Iterators

The simplest type of iterator is a basis iterator, which is implemented in the class `CGBTree_Iterator`. It loops over the basis functions present in some system. In the QOL, iterations are not implemented in a flat, but in a structured way. Instead of a single loop, four levels of hierarchy are provided. At the first level, all centers of the system are looped over. The second level then loops over the different angular momenta of the current center. The third level now differs, depending on the basis used. In a primitive basis, the loop is over the different exponents for the actual center-angular combination. If the radial angular transformed basis is chosen, the loop iterates over the contractions for the current tuple. At this point, the iteration runs over the shells of the system, in the primitive-, or AO-basis. The fourth level then gives access to the individual functions of the shell. Listing 10.4 applies this to the example given above, using a primitive basis.

```
1  Molecule molecule("~/H.in");
2
3  CGBTree_Plain<double> pBasis(molecule, "emsl/cc-pvdz");
4
5  typedef CGBTree_Iterator<CGBTree_Plain<double> > ITERATOR;
6
7  ITERATOR pIter(pBasis);
8
9  for (typename ITERATOR::SubIter11 i1(pIter); i1.valid(); ++i1) {
10   cout << "center␣" << pIter.i1()->first << endl;
11
12   for (typename ITERATOR::SubIter22 i2(pIter); i2.valid(); ++i2) {
13     cout << "␣␣angular␣" << pIter.i2()->first.l() << endl;
14
15     for (typename ITERATOR::SubIter33 i3(pIter); i3.valid(); ++i3) {
16       cout << "␣␣␣␣exponent␣" << pIter.i3()->first.zeta() << endl;
17
18       for (typename ITERATOR::SubIter44 i4(pIter); i4.valid(); ++i4)
          {
19         cout << "␣␣␣␣␣shell␣" << pIter.i4()->first <<
20         cout << "␣␣␣␣␣index␣" << pIter.i4()->second <<
21         cout << "␣␣␣␣␣␣␣ITER␣" << pIter << endl;
22       }
23     }
24   }
25 }
```

Listing 10.4: Iterating over a primitive basis.

The primitive basis can be easily changed to the radial-angular-transformed basis by changing the basis template in the `typedef` from `CGBTree_Plain` to `CGBTree_Contracted_AngularTransformed`.

The properties of the basis are accessed by the methods `i1()` to `i4()` of the iterator. In the example this is done at the beginning of the actual loops changing the quantity, but this is not restrictive. In fact properties can be queried anywhere in the loops, regardless of the actual level.

The output of the above example is given in listing 10.5. On the left, before `ITER`, the hierarchical access to the basis can be seen. On the right, the actual basis function is shown.

```
1  center [0 0 0]
2    angular 0
3      exponent 13.01
4        shell (0, 0, 0)    index 0       ITER [[0 0 0], s, (13.01),
           (0, 0, 0) (#0)]
5      exponent 1.962
6        shell (0, 0, 0)    index 1       ITER [[0 0 0], s, (1.962),
           (0, 0, 0) (#1)]
7      exponent 0.4446
8        shell (0, 0, 0)    index 2       ITER [[0 0 0], s, (0.4446),
           (0, 0, 0) (#2)]
9      exponent 0.122
10       shell (0, 0, 0)    index 3       ITER [[0 0 0], s, (0.122),
           (0, 0, 0) (#3)]
11   angular 1
12     exponent 0.727
13       shell (1, 0, 0)    index 4       ITER [[0 0 0], p, (0.727),
           (1, 0, 0) (#4)]
14       shell (0, 1, 0)    index 5       ITER [[0 0 0], p, (0.727),
           (0, 1, 0) (#5)]
15       shell (0, 0, 1)    index 6       ITER [[0 0 0], p, (0.727),
           (0, 0, 1) (#6)]
```

Listing 10.5: Output of looping over the primitive basis structure of H in a cc-pVDZ basis

10.2.2. Two- and Four-Index Iterators

Similar iterators are provided to loop over pairs and quadruples of basis functions. Two-index iterations, needed for one particle matrix elements such as overlap and kinetic-energy integrals, are provided by the `CGBTree_HermitianTupel2_Iterator` class. Loops over four-index quantities, such as the two-electron integrals, are implemented via the `CGBTree_HermitianTupel4_Iterator` class. Only the `CGBTree_HermitianTupel4_Iterator` will be discussed in some detail, as the use of the two-index iterator is essentially identical.

Both iterators are implemented as the basis iterator, resembling the four-fold hierarchy of the basis in the same way. The first level generates all center-quadruples, for each of which all angular tuples are iterated over, followed by exponents/contractions and the individual shell functions. Note that the first three levels effectively iterate over the integral batches. The fourth level, iteration over all two-electron integrals is not needed in most cases.

As indicated in the name, using these two iterators will not give all n^2 or n^4 possible combinations of basis-functions. Both are written to take integral/batch symmetries into account, iterating only over all non-redundant tuples.

An example for hydrogen in a cc-pVDZ basis is given in listing 10.6.

```
1  Molecule molecule("~/H.in");
2
3  typedef CGBTree_Plain<double> BasisType;
4  BasisType basis(molecule, basis);
5
6  typedef TwoParticleBasis<BasisType, TwoParticleFunctionPair>
     BasisType2;
7  BasisType2 basis2(basis);
8
9  typedef CartesianGaussian_UnitarySpace2<BasisType> USpace;
10 USpace uSpace(basis2);
11
12 typedef CGBTree_HermitianTupel4_Iterator<BasisType> PI4;
13 typedef typename PI4::SubIter11 I11;
14 typedef typename PI4::SubIter22 I22;
15 typedef typename PI4::SubIter33 I33;
16 typedef typename PI4::SubIter44 I44;
17 PI4 pIter4(uSpace);
18
19 for (I11 i1(pIter4); i1.valid(); ++i1) {              // center
20   for (I22 i2(pIter4); i2.valid(); ++i2) {            // angular
21     for (I33 i3(pIter4); i3.valid(); ++i3) {          // exponent
22       for (I44 i4(pIter4); i4.valid(); ++i4) {        // shell
23         int idx_bra1 = pIter4.i4().bra.i1->second;
24         int idx_bra2 = pIter4.i4().bra.i2->second;
25         int idx_ket1 = pIter4.i4().ket.i1->second;
26         int idx_ket2 = pIter4.i4().ket.i2->second;
27         cout << idx_bra1 << "␣" << idx_bra2 << "␣" << idx_ket1 << "␣
             " << idx_ket2 << endl;
28       }
29     }
30   }
31 }
```

Listing 10.6: Iterating over four-index tuples.

Despite the additional two-particle definitions, the use of the `CGBTree_HermitianTupel4_Iterator` is almost identical to the basis iterator. Similarly, the basis can be changed by simply replacing the basis class in the `BasisType` typedef. The access methods are also identical, just extended by bra/ket and i1/i2 to indicate which function of the quadruple to access.

Iteration order and naming convention are based on the physicist notation

$$\langle ij|kl\rangle = \int i(1)j(2)k(1)l(2)r_{12}^{-1}dr_1dr_2$$

referring with bra to the functions i and j, and with ket to k and l, respectively. The particle is identified by i1 and i2. The use of the two-index iterator is essentially the same, although the specification of the particle (i1 and i2) is not necessary in this case.

Note that, in the case of a four-index iteration, each of the four level loops masks a four-fold loop, one for each index, giving the four index iteration as a loop over a 16

dimensional quantity. In the two-index case, two loops are masked by each level.

Finally, listing 10.7 shows part of the output of the code of listing 10.6, in a formatted way. Shown is the fourth level loop over the *pppp*-batch with four equal *p*-shells, each of which holds the basis functions 4, 5 and 6. Clearly the neglection of redundant tuples can be seen, for example the tuples 5444, 4544 and 4454 are identical to 4445 and omitted.

```
1  4 4 4 4        4 4 4 6        5 5 5 6        4 6 6 6
2  4 4 4 5        4 4 5 6        4 5 6 6        5 6 6 6
3  4 4 5 5        5 4 5 6        5 5 6 6        6 6 6 6
4  4 5 4 5        4 4 6 6        4 6 4 6
5  4 5 5 5        4 5 4 6        4 6 5 6
6  5 5 5 5        4 5 5 6        5 6 5 6
```

Listing 10.7: Part of the output of looping over the four-index tuples of H in a cc-pVDZ basis. Index tuples are given for the *pppp*-batch.

10.2.3. Arbitrary Level and Notation Iteration

The iterators provided by the QOL lack certain functionalities, which could not be added to the existing framework. Therefore a new iteration approach has been implemented, to overcome those shortcomings. The major property to be changed in the two-particle iterator was to alter the order of iteration from physicist to chemist notation. A great deal of quantities arising are particle bound, for example the Schwarz integrals and parts of the shell batch data. Iteration in the physicist notation constantly breaks the particle correlation, leading to redundancies and the recalculation/reloading of various quantities. Closely connected is the possibility to fully access the four-fold loops at each level, allowing for a efficient factoring out of quantities at loop access. Thirdly, at some point in the implementation, the change of the fixed center-angular-exponent-shell hierarchy became desirable.

A general scheme has been devised to ensure non-redundant iteration for all six possible hierarchies, in physicists or chemist notation, using a simple piece of code. Understanding this is non-trivial and shall be explained in detail.

To do so, the angular loop of the center-angular-exponent hierarchy shall be used. To simplify manners, the exponent part is going to be neglected for now. The physicists notation $\langle ij|kl\rangle = \int i(1)j(2)r_{12}^{-1}k(1)l(2)dr_1dr_2$ is going to be used, depicting i as bra1, j as bra2 and k and l as ket1 and ket2, respectively. Suppose some centers A and B, associated with the following angular momenta.

$$A \leftarrow \{0, 1, 2\}$$
$$B \leftarrow \{0, 1\}$$

For the angular loop of the bra1 center, no restriction exists, all indices can be used.

For the bra2 angular loop, restrictions exist, if some condition is met. This condition is the equality of the bra1 and bra2 and the ket1 and ket2 centers. In this case, the bra2 loop has to start with the angular momentum of the current bra1 index. For example in the center pattern $AABB$, with the bra1 angular momentum being one, starting

the bra2 loop with angular momentum 0 leads to the iteration seed $10ij$, with i and j determined by the following ket1 and ket2 loops. As the combination $01ij$ has already be iterated over, iterating over $10ij$ will only give redundant patterns.

The ket1 loop is bound to the bra1 center, as interchange gives a symmetry equivalent integral. In the case of equal bra1 and ket1 centers, the ket1 angular iteration has to start with the current angular momentum of the bra1 loop. This prevents the generation of i.e. $1i0j$ as an iteration seed.

Similar interchange of bra2 and ket2 centers leads to the first restriction of the ket2 loop. If both centers are equal, the ket2 angular loop has to start with the angular momentum of the bra2 position. An additional restriction exists if the bra1, bra2 and ket1, ket2 centers are equal. For the center pattern $AABB$ (or $AAAA$), the angular pattern $iijk$ exist. In such a case the ket2 angular momentum has to be bound to the ket1 angular momentum. An example would be the angular seed $221k$ (for $AABB$). Starting the ket2 loop by 0 would give 2210, which has already been generated by the previous seed $220k$.

This can be generalized by thinking of the centers in the example as index containers. In the same manner as the centers of a center tuple hold sets of angular momenta, the angular momenta of an angular tuple hold sets of exponents. For these the same restrictions as derived for the angular loop of a center tuple hold. Equivalently the level assignment can be changed, i.e. starting by angular momenta with assigned centers with assigned exponents. Each level loop can be expressed in four loops over the index sets, using the restrictions above to ensure iteration over non-redundant tuples.

Listing 10.8 shows the loop for level two in the center (`Point3D`), angular (`int`), exponent (`double`) hierarchy.

```
1  typedef typename vector<pair<Point3D<double>,
2                   vector<pair<int,
3                   vector<pair<double, int> > > > > >::const_iterator
                     L1;
4
5  typedef typename vector<pair<int,
6                   vector<pair<double, int> > > >::const_iterator L2;
7
8  // L1 l1_b1, l1_b2, l1_k1, l1_k2;
9
10 for (L2 l2_b1 = l1_b1->second.begin(); l2_b1 != l1_b1->second.end();
     ++l2_b1)
11 {
12   for (L2 l2_k1 = (l1_b1 == l1_k1) ? l2_b1 : l1_k1->second.begin();
       l2_k1 != l1_k1->second.end(); ++l2_k1)
13   {
14     for (L2 l2_b2 = ((l1_b1 == l1_b2) && (l1_k1 == l1_k2)) ? l2_b1 :
        l1_b2->second.begin(); l2_b2 != l1_b2->second.end(); ++l2_b2)
15     {
16       L2 l2_k2 = l1_k2->second.begin();
17
18       if (l1_b2 == l1_k2)
19         l2_k2 = l2_b2;
```

```
20          if ( (l1_b1 == l1_b2) && (l1_k1 == l1_k2) && (*l2_b1 ==
              *l2_b2))
21            l2_k2 = l2_k1;
22          for (; l2_k2 != l1_k2->second.end(); ++l2_k2)
23          {
24          }
25        }
26      }
27  }
```

Listing 10.8: General loop structure for non-redundant hierarchical iterations. Specialisation for level two angular iteration.

The loop-structure itself is general, the specialization to a specific hierarchy is done with the `typedef`'s in the beginning.

Listing 10.8 gives the non-redundant tuples in terms of the physicists notation. The chemists notation $(ik|jl) = \int i(1)k(1)r_{12}^{-1}j(2)l(2)dr_1dr_2 = \langle ij|kl\rangle$ can be generated from the physicists notation by interchange of bra2 ($j(2)$) and ket1 ($k(1)$) functions. Using this in listing 10.8, i.e. interchanging the bra2 and ket1 loops gives the leveled, non-redundant iteration in chemist notation.

Obviously, this structure can be easily downgraded to two- and one-index iterations.

11. Prescreening

Integral prescreening has been implemented for primitive Cartesian Gaussian bases. In a first ansatz, prescreening is performed in terms of batches. As described in section 7.1.1, a batch $\langle IJ|KL\rangle$ of integrals can be neglected if its maximum contribution to the Fock matrix is less than some given threshold T.

$$|G(\langle IJ|KL\rangle)| \leq S^{max}_{batch} \cdot D^{max}_{batch} \leq T \tag{11.1}$$

The maximum integral value S^{max}_{batch} can be estimated by the product of two maxima over certain sets of Schwarz integrals $S_{ij} = \sqrt{\langle ii|jj\rangle}$. The first set consists of the Schwarz integrals arising from the Cartesian product of the functions in the I- and K-shells, the second is obtained in the same way from the J- and L-shells.

$$S^{max}_{batch} = max(\{S_{\{I\}\mathrm{x}\{K\}}\}) \cdot max(\{S_{\{J\}\mathrm{x}\{L\}}\}) \tag{11.2}$$

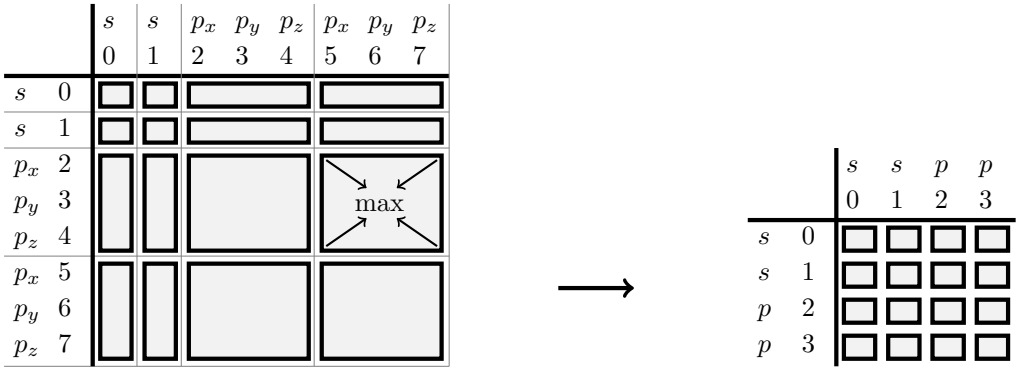
For the maximum density contribution, similar maxima over sets of density matrix elements are needed. Again, the sets are built from the Cartesian product of functions in pairs of shells.

$$\begin{aligned} D^{max}_{batch} = max(\{&2 \cdot max(\{D_{\{I\}\mathrm{x}\{K\}}\}, \{D_{\{J\}\mathrm{x}\{L\}}\}), \\ &0.5 \cdot max(\{D_{\{I\}\mathrm{x}\{J\}}\}, \{D_{\{I\}\mathrm{x}\{L\}}\}, \{D_{\{J\}\mathrm{x}\{K\}}\}, \{D_{\{K\}\mathrm{x}\{L\}}\})\}) \end{aligned} \tag{11.3}$$

11.1. General Implementation

Before delving into the optimized specializations of the implementation, the basic ideas shall be sketched. Central is the concept of generating maxima over certain elements of Schwarz integrals, or density matrix elements. Now the Schwarz integrals S_{ij} can be interpreted as a matrix having the same structure as the density matrix. By structure the order of basis function is meant. In the QOL framework, the ordering is given by the iteration, resulting in the center-angular-shell-function hierarchy introduced in section 10.2.1. The individual maxima to calculate the maximum contribution of a batch are generated from matrix elements arising from the combination of two shells. In the QOL ordering of basis functions, these matrix elements correspond to blocks in the respective matrices. The maxima have then to be built over the elements of these blocks. Figure 11.1 exemplifies this for some model system, consisting of one center, with two s- and p-shells. On the left in figure 11.1a, the full n x n matrix is shown, emphasizing the shell blocks the maxima have to be generated for. Figure 11.1b on the right then shows the corresponding matrix of the maxima, in terms of the shells of the

system.



(a) Full Schwarz integral- or density-matrix. Shell blocks over which maxima have to be generated are emphasized.

(b) The reduced block-maxima matrix.

Figure 11.1.: Reduction of the full Schwarz integral- or density-matrix to the block-maxima form. In terms of shell batches.

Note that, as both the Schwarz integral and density matrices are symmetric, i.e. $S_{ij} = S_{ji}$ and $D_{ij} = D_{ji}$, also the block maxima reduced matrices are symmetric.

11.1.1. The `MapperWithMatrix` Ansatz

This block maxima generation is identical for the Schwarz and density estimate. For both, the block maxima reduced matrices of figure 11.1b are generated prior to the calculation, accessing their elements when needed. For the Schwarz integrals, this can be done once for the entire Hartree-Fock calculation. The density maxima, on the other hand, have to be updated with changing density matrices, i.e. in each iteration.

Internally, the Schwarz and density estimates are handled by two classes, `SchwarzEstimate` and `DensityEstimate`, respectively. The logic to store, update and access the block maxima matrices is handled by the class `MapperWithMatrix`. It stores the corresponding block maxima matrix and provides a means to translate between basis function- and shell indices. The basic idea is represented by a simple `vector<int> _idxMap`, of size n. Its index corresponds to the index of the basis function, the value at a certain index being the index of the shell. The above example of one center with two s- and p-shells results in the following structure.

	s	s	p_x	p_y	p_z	p_x	p_y	p_z
`idx`	0	1	2	3	4	5	6	7
`_idxMap[idx]`	0	1	2	2	2	3	3	3

Table 11.1.: Mapping indices from basis functions to shells.

The eight basis functions correspond to four shells. The matrices holding the maxima therefore are of the size 4 x 4. The access, in terms of basis function indices, is accomplished via the `_idxMap` translator.

```
1 // matrix<double> _maxVals;
2 double & MapperWithMatrix::operator () (int i, int j)
3 { return _maxVals(_idxMap[i], _idxMap[j]); }
```

Listing 11.1: Accessing the `_maxVals` matrix.

11.1.2. Initialization

In the `DensityEstimate` class, the initialization/update of the maxima is then simply handled by the loop in listing 11.2. The density elements are stored in the matrix `_D`, and the maxima are handled by the `MapperWithMatrix _D_MWM`. First, the underlying matrix `_maxVals` is reset to zero. In the following loop over the elements of the density matrix, the shell pair maxima are updated by determining the maxima of the current shell pair maxima and the actual density matrix element.

```
1 // matrix<double> _D;
2 // MapperWithMatrix<double> _D_MVM;
3 void DensityEstimate<double>::update() {
4   _D_MVM._maxVals() *= 0.0; // reset _maxVals matrix
5   for (int row = 0; row < _D.rows(); ++row)
6     for (int col = 0; col < _D.cols(); ++col)
7       _D_MWM(row,col) = max(_D_MWM(row,col), fabs(_D(row, col)));
8 }
```

Listing 11.2: Updating the density matrix block-maxima.

Initialization of the maximum values of the Schwarz integrals is principally identical, the difference being the embedding into the calculation of the integrals. The integration is performed according to section 12.4, where it will be discussed in detail. Worth mentioning is the fact that, for the Schwarz integrals being a two-index quantity, a two index iteration ansatz is sufficient. A minor drawback is that due to the fact that integrals are evaluated in batches, a lot of unneeded integrals are calculated in addition to the Schwarz integrals. This is shown in table 11.2, for various molecules and basis sets.

		Calculated	Schwarz-Integrals	Overhead (%)
C_4H_{10}	Def2-SVP	103137	14673	85.8
C_4H_{10}	cc-pVDZ	106741	16069	84.9
C_4H_{10}	Def2-TZVP	618480	40872	93.4
C_4H_{10}	cc-pVDZ	106741	16069	84.9
C_4H_{10}	cc-pVQZ	10097685	248625	97.5
Alanine	Def2-SVP	164791	20083	87.8
Alanine	cc-pVDZ	171649	22549	86.9
Alanine	Def2-TZVP	1194768	66282	94.5
Alanine	cc-pVDZ	171649	22549	86.9
Alanine	cc-pVQZ	12915397	289039	97.8

Table 11.2.: Overhead of unneeded integrals when calculating the Schwarz integrals $\langle ii|jj\rangle$.

This has been not further optimized. The reason is summarized in table 11.3, which compares the timings, in seconds, of the initialization of the Schwarz maxima (including integration), to the full time needed to perform the Hartree-Fock calculation. For completeness sake, timings for updating the density maxima are included. Shown are results for serotonine in various basis sets.

Serotonine	Schwarz Init	Density Update	Full SCF Calculation
Def2-SVP	0.02	0.002	144.45
Def2-TZVP	0.27	0.006	1205.46
Def2-QZVP	2.04	0.028	15661.81
cc-pVDZ	0.03	0.002	172.82
cc-pVTZ	0.11	0.007	1346.57
cc-pVQZ	1.75	0.024	12328.92
cc-pV5Z	6.29	0.079	109297.09
cc-pV6Z	33.54	0.204	690142.58
roos-aug-dz-ano	0.11	0.015	7632.52
roos-aug-tz-ano	0.54	0.030	35534.73
ano-rcc	1.90	0.050	122815.99

Table 11.3.: Timing comparison of the initialization/update of the `_maxVals` matrices versus the full SCF calculation. Serotonine in various basis sets, in seconds.

Obviously, both initializing the Schwarz integral- and updating the density-maxima are negligible compared to the effort needed to perform the full SCF calculation.

11.1.3. Calculation of the Estimate

Accessing the maximum values is of course identical to updating. The `batchMax` method of the `SchwarzEstimate` class returns the estimate of the maximum integral value for some batch, having as arguments the indices of any of the basis functions of the corresponding shells. The `MapperWithMatrix _S_MWM` gives the respective maxima, their product being the estimate as given by (11.2).

```
1 const double SchwarzEstimate::batchMax(int p, int r, int q, int s)
    const {
2   return _S_MWM(p, q) * _S_MWM(r, s); }
```

Listing 11.3: Calculation of the Schwarz estimate.

Similarly, the maximum density element is generated using (11.3), implemented in `DensityEstimate`.

```
1 const double DensityEstimate::batchMax(int p, int r, int q, int s)
    const {
2   double m = 0.5 * _D_MWM(p,r);
3   m = max(m, 2.0 * _D_MWM(p,q));
4   m = max(m, 0.5 * _D_MWM(p,s));
5   m = max(m, 0.5 * _D_MWM(r,q));
6   m = max(m, 2.0 * _D_MWM(r,s));
7   m = max(m, 0.5 * _D_MWM(q,s));
8   return m;
9 }
```

Listing 11.4: Calculation of the density estimate.

Those two methods are then combined to calculate the maximum contribution of some batch. Recall that the effective iteration over batches is at the third level of the four-index iteration. Here the indices of the involved basis functions are queried. Those are needed to calculate the symmetry correction of equation (5.10), and to retrieve the maximum values. The product of symmetry correction and maximum values is then compared to some threshold, skipping the current batch if the criterion is met.

```
1  // int idx[4];
2  // SchwarzEstimate se;
3  // DensityEstimate de;
4  for (I33 i3(pIter4); i3.valid(); ++i3) {
5    idx[0] = pIter4.i4().bra.i1->second;
6    idx[1] = pIter4.i4().bra.i2->second;
7    idx[2] = pIter4.i4().ket.i1->second;
8    idx[3] = pIter4.i4().ket.i2->second;
9
10   symCorr = calcSymCorr(idx[0], idx[1], idx[2], idx[3]);
11
12   if ( (se.batchMax(idx[0], idx[1], idx[2], idx[3]) *
13         de.batchMax(idx[0], idx[1], idx[2], idx[3]) *
14         symCorr)    < thresh)
15     continue;
16 }
```

Listing 11.5: Combination of Schwarz and density estimate.

11.1.4. Accuracy

The accuracy of the two-electron energy obtained when activating integral prescreening is shown in table 11.4. The calculations have been performed on serotonine, in the cc-pVQZ basis. One SCF iteration has been performed, using the density obtained with TURBOMOLEs extended Hückel guess. For varying thresholds, the absolute and relative deviations to the energy obtained with disabled prescreening (threshold = 0) are listed. Additionally, the amount of integrals included (in percent) and the speedup in terms of execution time are shown.

Threshold	$\lvert E_{t=0} - E_t \rvert$	$\lvert E_{t=0} - E_t \rvert / \lvert E_{t=0} \rvert$	Calculated integrals (%)	Timing speedup
0.0	0.00	0.00	100	1.0
10^{-12}	$-4.20 \cdot 10^{-10}$	$2.10 \cdot 10^{-13}$	27	3.5
10^{-11}	$-5.20 \cdot 10^{-9}$	$2.60 \cdot 10^{-12}$	24	3.9
10^{-10}	$-6.80 \cdot 10^{-8}$	$3.40 \cdot 10^{-11}$	22	4.2
10^{-09}	$-7.50 \cdot 10^{-7}$	$3.80 \cdot 10^{-10}$	19	4.8
10^{-08}	$-5.20 \cdot 10^{-6}$	$2.60 \cdot 10^{-9}$	16	5.6
10^{-07}	$-9.20 \cdot 10^{-6}$	$4.60 \cdot 10^{-9}$	14	6.6
10^{-06}	$2.90 \cdot 10^{-4}$	$-1.50 \cdot 10^{-7}$	11	8.2
10^{-05}	$1.70 \cdot 10^{-2}$	$-8.60 \cdot 10^{-6}$	8	10.8
10^{-04}	$3.00 \cdot 10^{-1}$	$-1.50 \cdot 10^{-4}$	5	15.7
10^{-03}	3.70	$-1.90 \cdot 10^{-3}$	3	26.7

Table 11.4.: Accuracy with varying prescreening thresholds, for serotonine in a cc-pVQZ basis. Corresponding percentage of calculated integrals and resulting speedup.

Clearly, the error in the energy increases with increasing threshold. However, considering that a common convergence criterion for the energy is about 10^{-7}, an accuracy below this number can be achieved by thresholds of 10^{-10} and smaller. In this work a threshold of 10^{-11} is used, reproducing the exact energy, in this example, to eleven significant digits, corresponding to an absolute error of about 10^{-9}. This accuracy has been obtained by including only about 25% of all integrals, leading to a speedup of the calculation of about a factor of four.

11.1.5. Quality of the Estimate

The prescreening ansatz implemented here is not able to detect all negligible integrals. Three mechanisms have been identified by which non-significant integrals are included in the calculation. Before discussing them in detail, the amount of accessible vanishing contributions shall be shown. Table 11.5 lists the percentage of integrals giving contributions of less than 10^{-11}, with respect to the non-redundant integrals present in the system, as obtained by (11.2). Shown are the results obtained for the first SCF iteration, for a selected set of molecules and basis functions. The density matrices used are initial

guess densities, obtained with the TURBOMOLE program package, using the extended Hückel guess.

	cc-pVDZ	cc-pVTZ	cc-pVQZ	cc-pV5Z
C_4H_{10}	42.09	41.01	44.90	51.49
Alanine	52.64	48.37	50.12	54.24
Serotonine	75.89	73.88	75.34	77.37
$C_{36}H_{53}N_7O_9$	97.68	97.42	-	-

Table 11.5.: Percentage of negligible integral contributions as detected by the implemented prescreening ansatz. Listed are contributions of magnitudes smaller than 10^{-11}.

In this set of systems, even for small molecules in small basis sets already 40 percent of integral contributions can be neglected. This number increases only marginally with increasing basis size, but hugely with increasing molecule size. For the largest molecule $C_{36}H_{53}N_7O_9$ with 105 atoms only 2.32 percent of the 1242204008100 non-redundant integrals have to be included in the calculation.

Overestimation of the Batch Contributions

As has been pointed out in section 7.2.1, estimating a batch by (11.2) might overestimate the maximum contribution. This is due to the fact that the maximum integral value S^{max}_{batch}, and the maximum density element D^{max}_{batch} are not necessarily linked. In such cases the estimated contribution turns out to be higher than the one accessible using the individual approach (7.4).

To detect the amount of screenable batches missed by this overestimation, batches surviving (11.2) have been analyzed in detail. After passing the estimate (11.2), the maximum contribution of that batch has been determined by application of the individual estimate to each integral. Indeed batches could be identified this way which passed the estimate (11.2), but nevertheless gave a maximum contribution to the Fock matrix smaller than the threshold of 10^{-11}. The results obtained for the set of molecules and basis sets introduced in table 11.5 are shown in table 11.6. The same calculational details were used. Shown is the percentage of screenable integrals with respect to the number of surviving ones.

	cc-pVDZ	cc-pVTZ	cc-pVQZ	cc-pV5Z
C_4H_{10}	0.41	1.70	3.91	6.98
Alanine	0.73	2.41	5.11	8.49
Serotonine	1.31	2.99	6.06	9.53
$C_{36}H_{53}N_7O_9$	2.01	5.39	-	-

Table 11.6.: Amount of integrals missed by overestimation, in percent. Relative to integrals surviving the implemented screening.

The number of integrals included but giving vanishing contributions is of significant amount. It increases with increasing molecular and basis set size. For serotonine in a cc-pV5Z basis, about one tenth of the integrals calculated correspond to batches giving negligible contributions.

Unfortunately, the effort to detect such batches, i.e. using (7.4) to determine the maximum batch contributions, is of significantly higher cost compared to the overhead to calculate the not detected batches. Application of (11.2), in technical terms, requires a maximum to be built over six numbers (D_{batch}^{max}), which has to be multiplied by two numbers (S_{batch}^{max}). In the individual approach (7.4), the same number of operations has to be performed for each integral in the batch, followed by the determination of the maximum value.

Batches with Significant and Insignificant Contributions

Second source for screenable, but unnecessarily calculated integrals is the evaluation of integrals in terms of batches. Even one significant integral contribution leads to the calculation of the whole batch, including vanishing contributions.

To detect their number, all integrals surviving both the batch based screening (11.2), and the individual batch screening presented above have been analyzed individually, using (7.4). Their number is summarized in table 11.7, again using the set of molecules and basis sets of above. Shown is the percentage with respect to integrals passing (11.2).

	cc-pVDZ	cc-pVTZ	cc-pVQZ	cc-pV5Z
C_4H_{10}	8.39	14.27	19.90	24.50
Alanine	11.30	16.69	21.35	24.68
Serotonine	20.46	26.31	30.87	33.83
$C_{36}H_{53}N_7O_9$	17.53	23.36	-	-

Table 11.7.: Amount of integrals missed by the batch screening ansatz, in percent. Relative to integrals surviving the implemented screening.

An even higher amount of additionally screenable integrals can be found, again increasing with increasing molecule and basis set size. For serotonine in a cc-pV5Z basis, one third of the calculated integrals could be neglected.

However, as will be shown in detail in section 12.1.1, no really feasible way to calculate integrals individually exists in the framework of the OS scheme. As integrals of a given batch are closely bound together by the recursion intermediates, the computational cost to calculate an integral individually is close to the cost of a whole batch. Additionally, adapting the code-generating scheme of section 12.4 to individual integrals would result in a huge amount of code and access logic. Consider for example a *psss* batch. Of the three integrals in this batch, none, one, two or three might be negligible, giving a total of eight possibilities to be taken into account.

Integrals not Subject to the Schwarz Inequality

Thirdly, there are integrals giving vanishing contributions which can not be detected using the Schwarz inequality (7.1). Their number has been determined similarly to the methods described above, analyzing only the integrals which survived all three presented screening hierarchies. For each of those integrals, the exact maximum contribution (5.3) has been compared to the threshold of 10^{-11}. The percentage of negligible integrals with respect to the integrals calculated after application of (11.2) is shown in table 11.8.

	cc-pVDZ	cc-pVTZ	cc-pVQZ	cc-pV5Z
C_4H_{10}	12.75	13.40	14.34	15.43
Alanine	15.40	14.93	14.87	15.23
Serotonine	21.31	20.41	19.73	18.95
$C_{36}H_{53}N_7O_9$	25.19	23.24	-	-

Table 11.8.: Amount of integrals missed by the Schwarz inequality, in percent. Relative to integrals surviving the implemented screening.

Again, quite a substantial number of integrals passing the implemented prescreening ansatz turn out to give vanishing contributions to the Fock matrix. The reasons for integrals being missed by the Schwarz inequality shall not be discussed here, it shall only be mentioned that at least a subset of those might be accessible by the multipole screening method of Lambrecht et al. [15].

11.2. Exploiting Basis Set Hierarchies

The concept of shifting the interpretation of contributions to the Fock matrix from a singular integral ansatz to estimating batches at once can be extended to further levels of hierarchies. The center-angular-exponent hierarchy introduced with the QOL iterators is particularly well suited for such an ansatz. For example, especially in sufficiently large systems, there will be center combinations for which all integral contributions are negligible. Similarly, there are cases for which all integrals of a particular angular combination of a certain center combination can be discarded on the whole. It will be shown that these negligible sets of integrals can be determined in a very efficient manner, leading to a considerable speedup of the calculation.

In the ansatz introduced above, the estimation of the maximum contribution of a batch has been based on the determination of maxima over sets of elements arising from the combination of shell pairs. In a similar manner, maximum contributions can be estimated for larger subsets of integrals. Considering some center combination $ABCD$, the Schwarz integrals needed to estimate all integrals associated with this center tuple arise from the Cartesian product of basis functions on the centers A combined with C, and B with D, respectively. The same holds for the maximum density element associated with this center combination, which can be determined from the function combinations of the center pairs (A, B), (A, C), (A, D), (B, C), (B, D) and (C, D).

Conceptually this is identical to the block maxima generation on batch level (see section 11.1), extended to blocks of centers and angular momenta. In total, such maxima can be generated for the three hierarchies of the iteration, leading to the block maxima structure depicted in figure 11.2. Shown is a system with two centers, each of which holds two s-shells, and two p-shells. The block maxima shown correspond to the batch-, angular-, and center-hierarchies, in clock-wise fashion.

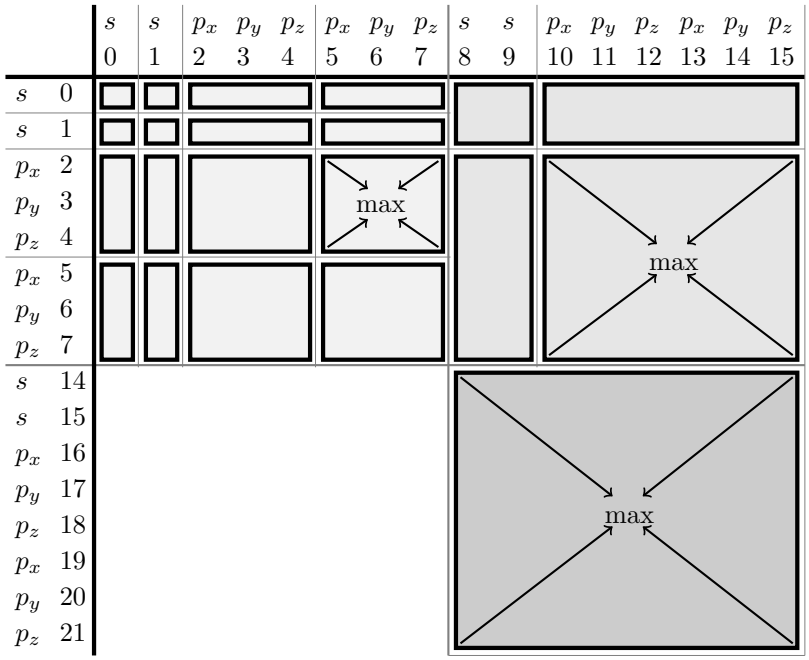


Figure 11.2.: Block maxima reduction for the three hierarchies. Batch-, angular- and center-maxima blocks are shown in clockwise fashion.

Having introduced the `MapperWithMatrix` class in section 11.1.1, the additional levels of hierarchy can be realized in a straight forward way by implementing different `_idxMap`'s, one for each hierarchy. Using the above example, the following structure is obtained:

		s	s	p_x	p_y	p_z	p_x	p_y	p_z	s	s	p_x	p_y	p_z	p_x	p_y	p_z
	`idx`	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
center	`_idxMap[idx]`	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
angular	`_idxMap[idx]`	0	0	1	1	1	1	1	1	2	2	3	3	3	3	3	3
batch	`_idxMap[idx]`	0	1	2	2	2	3	3	3	4	5	6	6	6	7	7	7

Table 11.9.: Mapping indices. Basis function indices to center-, angular- and batch-indices.

Now the `MapperWithMatrix` can represent any of the three hierarchies, depending on the index mapping used. The access method to the maximum values does not change, as the logic is hidden in the index map.

The `DensityEstimate` and `SchwarzEstimate` classes have to be extended by methods referring to the additional hierarchies. The update of the density maxima is simply changed to listing 11.6.

```
1  // matrix<double> _D;
2  // MapperWithMatrix_Center<double> _D_MVM_Center;
3  // MapperWithMatrix_Angular<double> _D_MVM_Angular;
4  // MapperWithMatrix_Batch<double> _D_MVM_Batch;
5  void DensityEstimate<double>::update() {
6    _D_MVM_Center._maxVals() *= 0.0; // reset _maxVals matrix
7    _D_MVM_Angular._maxVals() *= 0.0;
8    _D_MVM_Batch._maxVals() *= 0.0;
9
10   for (int row = 0; row < _D.rows(); ++row)
11     for (int col = 0; col < _D.cols(); ++col)
12     {
13       _D_MWM_Center(row,col) = max(_D_MWM_Center(row,col),
           fabs(_D(row, col)));
14       _D_MWM_Angular(row,col) = max(_D_MWM_Angular(row,col),
           fabs(_D(row, col)));
15       _D_MWM_Batch(row,col) = max(_D_MWM_Batch(row,col),
           fabs(_D(row, col)));
16     }
17 }
```

Listing 11.6: Updating the density matrix maxima in the three level approach.

Similar initializations have to be performed for the Schwarz estimate. Additionally to the `batchMax` methods of both estimates, the `centerMax` and `angularMax` methods are provided. The full loop for prescreening is shown in listing 11.7. At each level of hierarchy, after obtaining indices, the maximum Fock contribution for the associated integrals is calculated. If the criterion is met, the whole set of underlying integral batches is neglected.

Note that the symmetry correction is only available at the zeta level of hierarchy. However, as the symmetry correction is always smaller or equal one, the worst case is an overestimation of the estimate at the levels one and two. Additional overestimation arises, identically to the batch estimate, by the fact that the maximum Schwarz- and density-estimates are not necessarily linked.

```
1  for (I11 i1(pIter4); i1.valid(); ++i1) {
2    // obtain indices
3    if ( (se.centerMax(idx[0], idx[1], idx[2], idx[3]) *
4          de.centerMax(idx[0], idx[1], idx[2], idx[3]) *)     < thresh)
5      continue;
6
7    for (I22 i2(pIter4); i2.valid(); ++i2) {
8      // obtain indices
9      if ( (se.angularMax(idx[0], idx[1], idx[2], idx[3]) *
10           de.angularMax(idx[0], idx[1], idx[2], idx[3]) *)     <
               thresh)
11       continue;
12
```

```
13      for (I33 i3(pIter4); i3.valid(); ++i3) {
14        // obtain indices and symCorr
15        if ( (se.batchMax(idx[0], idx[1], idx[2], idx[3]) *
16              de.batchMax(idx[0], idx[1], idx[2], idx[3]) *
17              symCorr)    < thresh)
18          continue;
19      }
20    }
21  }
```

Listing 11.7: Querying the three level prescreening machinery.

Now the computational effort to determine the estimate is identical at each level. For the maximum Schwarz integral values two numbers have to be read from matrices, and for the maximum density element the maximum over six numbers has to be determined. For a given vanishing tuple (center or angular), the computational cost is thus reduced to one call to the `max` functions, compared to calls for each batch contained in the specific tuple.

11.2.1. Results

As the effects of the hierarchical screening show most prominently for large molecules, they shall be discussed in detail for the hexa-peptide $C_{36}H_{53}N_7O_9$. Results are shown for the first SCF iteration, in a cc-pVDZ basis, using a threshold of 10^{-11}. The density matrix used has been obtained with the TURBOMOLE program package, using the extended Hückel guess (see section 15.2.1). The iteration hierarchy was center-angular-batch, in physicist notation. As is shown in table 11.10, there are about 120 billion batches in this particular system, for each of which the prescreening machinery has to be queried at batch level, discarding 98 percent as non-significant.

Level	Queries	Discarded	Discarded(%)
Batch	121,781,735,481	119,835,900,493	98

Table 11.10.: Prescreening queries for $C_{36}H_{53}N_7O_9$ in a cc-pVDZ basis. Plain batch approach.

Table 11.11 now shows the results for the same calculation, with activated hierarchical screening. Shown are the queries to the prescreening machinery for the individual levels, and the number and percentage of queries leading to a neglection of the whole subset.

Level	Queries	Discarded	Discarded(%)
Center	15,487,395	9,921,863	64
Angular	215,934,075	122,458,704	57
Batch	31,393,945,305	29,448,110,317	94
Sum	31,625,366,775		

Table 11.11.: Prescreening queries for $C_{36}H_{53}N_7O_9$ in a cc-pVDZ basis. Hierarchical approach.

Already at the center level, about two thirds of 15 million non-redundant center tuple turn out to give vanishing contributions and can be neglected without detailed examination. For the surviving center tuples, another 57 percent of the angular tuples can be skipped in their entirety. On the batch level, about 31 billion batches remain to be queried individually. All in all, about 32 billion queries to the prescreening machinery have been performed, which is four times less than the 120 billion of the one level approach.

The wall clock time needed for the construction of the Fock matrix for both approaches is summarized in table 11.12. Deployment of the hierarchical approach gives a speedup of 2.9 compared to the one level approach. Both timings have been generated using the same code base, the difference being only the added screening hierarchies.

Plain	Hierarchical
4998.86	1772.25

Table 11.12.: Timings for the generation of the first Fock matrix for $C_{36}H_{53}N_7O_9$, in a cc-pVDZ basis, in seconds. Physicists notation iteration order.

As already mentioned, the effects of the hierarchical approach are only dominant in large molecules, which can be taken from table 11.13. Shown is the relationship of total queries and timings for some molecules and basis sets. Again, results are taken from the generation of the first Fock matrix, using TURBOMOLEs initial guess density.

		Plain/Hierarchical Queries	Timings
C_4H_{10}	cc-pVDZ	0.99	0.97
	cc-pVTZ	1.00	0.99
Alanine	cc-pVDZ	1.00	0.97
	cc-pVTZ	1.00	0.99
Serotonine	cc-pVDZ	1.07	1.01
	cc-pVTZ	1.07	1.01
$C_{36}H_{53}N_7O_9$	cc-pVDZ	3.85	2.94
	cc-pVTZ	3.33	2.13

Table 11.13.: Comparison of the plain and hierarchical approaches in terms of queries and timings.

Even for serotonine, having 25 atoms, the effect of the hierarchical approach does not show. For the 105-atom hexa-peptide $C_{36}H_{53}N_7O_9$, however, the effect is considerable, although slightly decreasing with increasing basis size.

11.3. Iteration in Chemists Notation. Further Levels of Hierarchy

The hierarchical screening approach introduced above can be applied equally to iteration schemes based on physicists- or chemists-notation, respectively. By using chemists notation, however, two additional levels of hierarchy can be introduced. Consider some center-angular tuple $(IJ|KL)$, at the third level of iteration. The outer two loops iterate over all shell tuples in I and J. For each given shell tuple i and j, the two inner loops iterate over the shell tuples in K and L. A further hierarchy can be introduced to generate maximum contributions for the integrals which can be built from the combination of two shells i and j, and the two sets of shells K and L. For the Schwarz estimate, on the one hand the maxima over i and j are needed, to be combined with the maxima over K and L. Now the maxima for the i and j combination correspond to the level three batch maxima, whereas the K and L maxima correspond to the level two, or angular, maxima. These are already available in the three level hierarchical approach. For the density maximum, in addition to the combinations of i and j, as well as K and L, hybrid maxima are needed. These arise from the mixed particle pair combinations (i, K), (i, L), (j, K) and (j, L). They can be generated using the block maxima approach introduced above, mixing pairs of different hierarchies. This is depicted in figure 11.3, showing the third level hybrid structure on the left. A certain shell (identified via the rows in the matrix) is combined with a set of shells of the same angular momentum, local to a center.

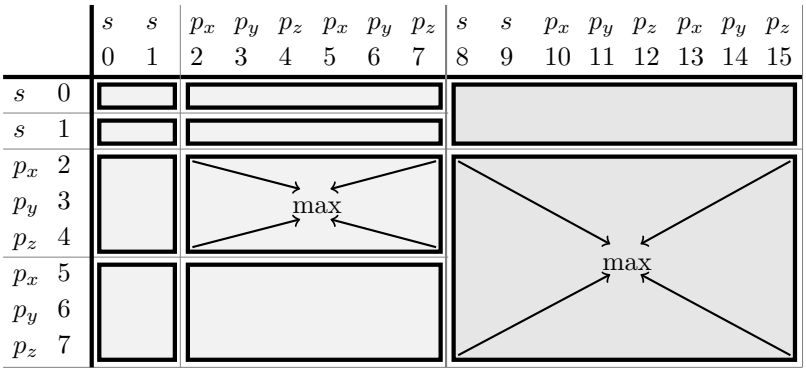


Figure 11.3.: Block maxima reduction for the hybrid levels. Batch level on the left, Angular level on the right.

In addition, the same argument leads to a hybrid hierarchy at the second, the angular level of hierarchy. At the two outer iterations, a set of angular functions is combined with all functions of the two centers at the inner loops. The particle one/two pairs can similarly be obtained from the level one/two block maxima matrices. For the mixed pairs hybrid block maxima have to be generated. Their structure is depicted in figure 11.3, on the right.

The implementation is identical to the one introduced above, the only difference being the addition of the two hybrid matrices to `DensityEstimate`. The internal `_maxVal` access is handled equivalently, using two different `_idxMaps` according to the hierarchy requested.

The effect shall be shown for the hexa-peptide $C_{36}H_{53}N_7O_9$, using the same calculational details as in the examples above. Results for the three level approach have already been shown in table 11.11, the results for the five fold hierarchical approach are summarized in table 11.14.

Level	Queries	Discarded	Discarded(%)
Center	15,487,395	9,921,863	64
$\text{Angular}_{\text{hyb}}$	37,706,614	12,294,798	33
Angular	146,665,057	53,189,686	36
$\text{Batch}_{\text{hyb}}$	1,110,714,365	781,395,831	70
Batch	8,979,069,367	7,033,234,379	78
Sum	10,289,642,798		

Table 11.14.: Prescreening queries for $C_{36}H_{53}N_7O_9$ in a cc-pVDZ basis. Five level hierarchical approach.

A considerable number of subsets can be discarded at the newly introduced hybrid levels. At the angular stage 33 percent of all queried subsets contain only negligible integrals, and on the batch stage 70 percent of the queries lead to vanishing contributions. Compared to the three level hierarchy approach, the total number of queries is reduced by a factor of three, from about 30 to ten billion queries.

Again this significantly affects the timings obtained for the Fock matrix generation, as shown in table 11.15.

Three Level	Five Level
898.81	642.09

Table 11.15.: Timings for the generation of the first Fock matrix for $C_{36}H_{53}N_7O_9$, in a cc-pVDZ basis, in seconds. Chemists notation iteration order.

Although not as prominent as in the transition from one to three hierarchies, still a speedup of about 1.4 could be achieved by introducing the two additional hierarchies. Note that the timings given in in table table 11.15 have been both generated using the chemists notation iteration ansatz, contrary to the physicists ansatz used in the introduction of the three level ansatz. The timings of table 11.12 and table 11.15 can therefore not be compared directly.

Similar to the introduction of the center- and angular-hierarchies, the effect of the two hybrid hierarchies only shows significantly for sufficiently large molecules. Table 11.16 gives an overview over the decrease of both timings and prescreening queries, comparing the three-level results with the five-level ansatz. Shown is the same set of molecules as in table 11.13, using the same computational details. Although the number of total queries decreases already for small molecules, the timings are only visibly affected for large molecules.

		3Level/5Level	
		Queries	Timings
C_4H_{10}	cc-pVDZ	1.20	1.02
	cc-pVTZ	1.18	1.01
Alanine	cc-pVDZ	1.32	1.03
	cc-pVTZ	1.27	1.01
Serotonine	cc-pVDZ	1.85	1.11
	cc-pVTZ	1.74	1.04
$C_{36}H_{53}N_7O_9$	cc-pVDZ	3.07	1.41
	cc-pVTZ	2.98	1.20

Table 11.16.: Comparison of the three- and five-level prescreening approaches in terms of queries and timings.

11.4. The Differential Density Scheme

To conclude the discussion of the implemented prescreening machinery, the differential density scheme shall be introduced here, as the amount of negligible integrals is significantly increased within this ansatz. Recalling section 7.3, the Fock matrix can be constructed incrementally, contracting the integrals not with the current density, but

with the differential density.

$$\mathbf{G}^{(i)} = \mathbf{G}^{(i-1)} + \mathbf{G}(\mathbf{D}^{(i)} - \mathbf{D}^{(i-1)}) = \mathbf{G}^{(i-1)} + \mathbf{G}(\Delta\mathbf{D}) \tag{11.4}$$

The implementation is straight forward, giving the call to the Fock matrix generator the differential density as argument instead of the actual density.

The changes in the density during an SCF calculation are moderate. Thus the values of the differential density are rather small in magnitude, especially if convergence is approached. The maximum contribution of some set of integrals depends on the maximum density element involved. Thus, in the differential density scheme, an increasing number of integrals can be neglected with increasing convergence. Using the actual density, on the other hand, leads to a rather constant amount of negligible integrals. This follows again from the moderate changes of the density matrix, giving a constant amount of small elements during the calculation. The Schwarz integrals, of course, stay constant in each iteration.

Table 11.17 summarizes this for serotonine in a cc-pVQZ basis. For each iteration the percentages of integrals to be calculated, and the required time (in seconds) are shown, with and without the differential density scheme enabled. The initial guess has been obtained from atomic densities. The prescreening threshold has been set to 10^{-11}, and the energy convergence threshold to 10^{-7}. For both cases the same code base has been used, differing only in the activation of the differential density scheme.

	Calculated integrals (%)		Required time (seconds)		
Itrn	Conventional	Incremental	Conventional	Incremental	% Of conventional
1	0.60	0.60	47	47	100
2	22.24	22.22	1522	1515	100
3	22.20	20.87	1520	1432	94
4	22.19	19.80	1518	1361	90
5	22.18	18.47	1518	1273	84
6	22.18	16.80	1518	1162	77
7	22.18	15.62	1518	1083	71
8	22.18	14.30	1518	995	66
9	22.18	13.75	1518	958	63
10	22.18	12.28	1518	860	57
11	22.18	11.52	1516	807	53
Sum			15208	11478	75

Table 11.17.: Comparison of the conventional and incremental Fock matrix generation schemes. In terms of calculated integrals and required time.

The first thing to note is the exceedingly small amount of integrals to be calculated in the first iteration. This is due to the use of the atomic densities guess, in which densities are calculated for each atom individually, and inserted in the initial guess density in a

block diagonal fashion. Therefore a huge number of density elements turn out to be zero, leading to almost all integral contributions to be negligible.

The effects of the differential density scheme then show starting with the third iteration. In each iteration, the amount of integrals to be taken into account decreases. In the conventional calculation, the number of integrals to be discarded stays about constant. In this example, by employing the differential density scheme, only half of the integrals to be included in the second iteration have to be considered in the last. Equivalently, the time needed to perform one iteration decreases accordingly. All in all, performing the calculation using the differential density scheme, only 75 percent of the time of the conventional calculation is needed.

As pointed out by others [14], constructing the Fock matrix incrementally introduces an error to the calculation. There are schemes to minimize this error [14], but they have not been implemented in this work. For the calculation presented above, serotonine in a cc-pVQZ basis, the error introduced to the energy is shown in table 11.18. Both absolute and relative error of the energy calculated within the differential density scheme, compared to the conventionally obtained, are shown, for each iteration, respectively.

Itrn	abs(E)	rel(E)
1	0.00	0.00
2	$1.81 \cdot 10^{-9}$	$3.17 \cdot 10^{-12}$
3	$6.48 \cdot 10^{-9}$	$1.14 \cdot 10^{-11}$
4	$2.10 \cdot 10^{-8}$	$3.68 \cdot 10^{-11}$
5	$2.83 \cdot 10^{-8}$	$4.96 \cdot 10^{-11}$
6	$3.09 \cdot 10^{-8}$	$5.43 \cdot 10^{-11}$
7	$3.54 \cdot 10^{-8}$	$6.22 \cdot 10^{-11}$
8	$3.90 \cdot 10^{-8}$	$6.85 \cdot 10^{-11}$
9	$3.91 \cdot 10^{-8}$	$6.87 \cdot 10^{-11}$
10	$4.23 \cdot 10^{-8}$	$7.43 \cdot 10^{-11}$
11	$4.61 \cdot 10^{-8}$	$8.09 \cdot 10^{-11}$

Table 11.18.: Comparison of the conventional and incremental Fock matrix generation. In terms of energy difference.

In this example, the absolute error stays below the energy convergence criterion of 10^{-7}. The energy of the conventional approach being reproduced by the differential density scheme to ten significant digits.

12. The Obara-Saika Integration Scheme

In the following, the evaluation of two-electron integrals will be shown in detail. One-electron integrals can be handled using the same techniques and will be briefly addressed in section 12.5.

In the Obara-Saika scheme, as has been shown in section 3.2, the two-electron integrals are expressed in terms of auxiliary integrals $\Theta^N_{i_x j_x k_x l_x; i_y j_y k_y l_y; i_z j_z k_z l_z}$. The subscripts describe the Cartesian powers of the corresponding primitive Cartesian Gaussians, and with $l_i = i_x + i_y + i_z$ their total angular momentum. Auxiliary integrals with $N = 0$ represent the final target integrals. Recursion seeds are the scaled Boys functions of order N, with all subscripts (angular momenta) equal to zero. The target integrals are built from these functions by iterative application of the Obara-Saika recurrence relations. Already introduced in section 3.2.2, they shall be repeated here. As in- and decrements are restricted to one Cartesian direction, they are given in the one-dimensional form, omitting unaltered subscripts. The direction of the transfer is implied by the prefactors.

To generate angular momenta on the first, the bra1 index, the vertical recurrence relation (V) is used.

$$\text{V: } \Theta^N_{i+1,000} = X_{PA}\Theta^N_{i,000} - \frac{\alpha}{p}X_{PQ}\Theta^{N+1}_{i,000} + \frac{i}{2p}\left(\Theta^N_{i-1,000} - \frac{\alpha}{p}\Theta^{N+1}_{i-1,000}\right) \tag{12.1}$$

Angular momenta on the second, the bra2 index are generated by the electron transfer recursion. Two different expressions have been implemented in the course of this work (E_1 and E_2).

$$\text{E}_1\text{: } \Theta^N_{i,j+1,00} = \frac{bX_{AB} + dX_{CD}}{q}\Theta^N_{i,j,00} + \frac{i}{2q}\Theta^N_{i-1,j,00} + \frac{j}{2q}\Theta^N_{i,j-1,00} - \frac{p}{q}\Theta^N_{i+1,j,00} \tag{12.2}$$

$$\begin{aligned}\text{E}_2\text{: } \Theta^N_{i,j+1,00} = {} & X_{QC}\Theta^N_{ij00} + \frac{\alpha}{q}X_{PQ}\Theta^{N+1}_{ij00} + \frac{i}{2(p+q)}\Theta^{N+1}_{i-1,j00} \\ & + \frac{j}{2q}\left(\Theta^N_{i,j-1,00} - \frac{\alpha}{q}\Theta^{N+1}_{i,j-1,00}\right)\end{aligned} \tag{12.3}$$

The two horizontal recurrence relations H1 and H2 finally generate angular momenta on the ket1 and ket2 indices, respectively.

$$\text{H1: } \Theta^N_{i,j,k+1,l} = \Theta^N_{i+1,j,k,l} + X_{AB}\Theta^N_{i,j,k,l} \tag{12.4}$$

$$\text{H2: } \Theta^N_{i,j,k,l+1} = \Theta^N_{i,j+1,k,l} + X_{CD}\Theta^N_{i,j,k,l} \tag{12.5}$$

12.1. General Considerations

12.1.1. Batchwise Evaluation

Due to the iterative application of the Obara-Saika recurrence relations, a lot of intermediate integrals are involved even in the calculation of simple (small angular momenta) integrals. These intermediates turn out to be not unique to a certain integral. Figure 12.1 exemplifies this for the nine integrals of a *ppss*-batch, using the electron transfer E_1. The target integrals are in the top, the recursion seeds in the bottom row. The numbers in square brackets correspond to N, and the four by four character matrix to the angular momenta of the auxiliary integral, in x-, y- and z-direction. The arrows indicate the intermediates needed to calculate a particular integral or intermediate.

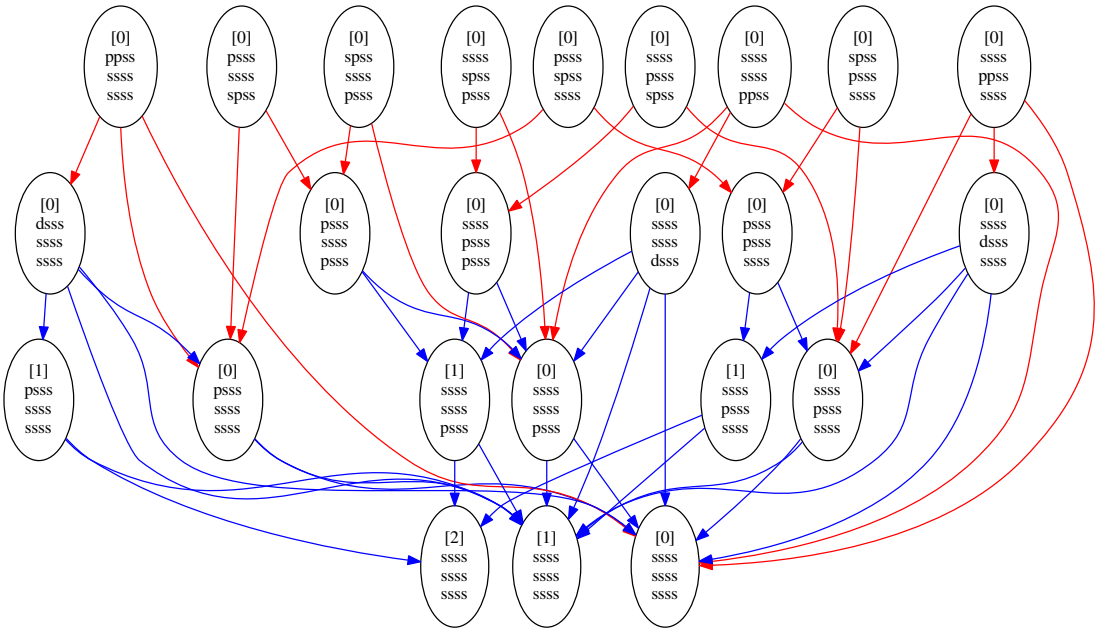


Figure 12.1.: Intermediates needed to calculate all integrals of a *ppss*-batch, and their dependencies.

The first thing to note is that almost all intermediates are shared among multiple target integrals, or intermediates (having more than one arrow pointing at them). This is the strongest argument for the calculation of integrals in terms of batches. This way the intermediates need only to be calculated once, to be used without reevaluation when necessary.

Additionally, all targets and intermediates share the same centers and exponents. The prefactors of the recurrence relations (12.1) to (12.5) are therefore constant for all intermediates of a batch. Similar to the intermediates, calculating them once and reusing

them for the whole batch saves considerable computational effort compared to repeated reevaluation.

Based on this, the integration of a batch has been split into a three-fold approach. In the first step, the prefactors for the target- and intermediate integrals are calculated. They will be referred to as `shellBatchData`. In the second step, the recursion seeds are calculated from the prefactors and the Boys function. In the final step, the intermediate- and target integrals are then calculated from the prefactors and the recursion seeds.

12.1.2. Batchwise Recursion

The Obara-Saika recurrence relations can be interpreted in terms of the total angular momenta of a batch. For all integrals of a batch, by in- or decrementing the Cartesian powers of all functions of some shell, the associated total angular momentum $l_i = i_x + i_y + i_z$ is in- or decremented accordingly.

Consider the nine target integrals of the $ppss^0$-batch of figure 12.1. The first recurrence relation to be applied is the electron transfer (12.2) E_1. The first term decrements each integral at the bra2 index, giving the three $psss$ integrals shown in figure 12.1 in the second to last row (with $N = 0$). They constitute the full $psss^0$ batch. In the second term of E_1, the bra1 index is decremented in addition to the bra2 index, resulting in the recursion seed $ssss^0$. The third term has no effect in this example, as negative bra2 indices are generated. The last term finally increments the bra1- , and decrements the bra2-index. The resulting six $dsss$ appear in the second row of figure 12.1, and again constitute the full $dsss^0$ batch.

Thus, by replacing the individual Cartesian powers of equations (12.1) to (12.5) by the total angular momenta of some batch, the recursion dependencies of whole batches can be analyzed. For the vertical transfer V (equation (12.1)), the following is obtained.

$$\mathrm{V} : \Theta^N_{l_i+1,000} \leftarrow \Theta^N_{l_i,000},\ \Theta^{N+1}_{l_i,000},\ \Theta^N_{l_i-1,000},\ \Theta^{N+1}_{l_i-1,000} \tag{12.6}$$

In doing so, the four-dimensional integral recursion (i_x, i_y, i_z, N), can be brought down to the two-dimensional description (l_i, N), and analyzed graphically.

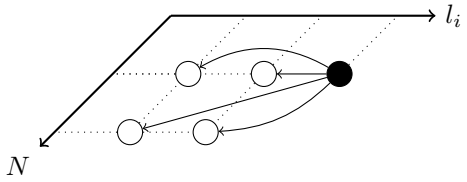


Figure 12.2.: Batch dependencies for the vertical transfer.

For the seven dimensional electron transfer E_1, similar batch dependencies can be given.

$$\mathrm{E}_1 : \Theta^N_{l_i,l_j+1,00} \leftarrow \Theta^N_{l_i,l_j,00},\ \Theta^N_{l_i-1,l_j,00},\ \Theta^N_{l_i,l_j-1,00},\ \Theta^N_{l_i+1,l_j,00} \tag{12.7}$$

The E_1 case, as N is not affected by the recurrence relation, is two-dimensional in l_i and l_j.

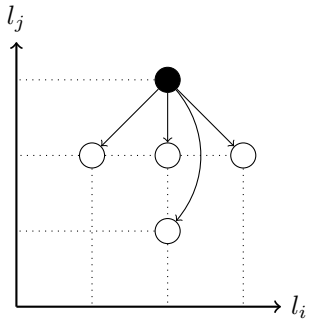


Figure 12.3.: Batch dependencies for the electron transfer E_1.

This is different in the alternative electron transfer E_2, having the batch dependencies

$$E_2 : \Theta^N_{l_i,l_j+1,00} \leftarrow \Theta^N_{l_i,l_j,00},\ \Theta^{N+1}_{l_i,l_j,00},\ \Theta^{N+1}_{l_i-1,l_j,00},\ \Theta^N_{l_i,l_j-1,00},\ \Theta^{N+1}_{l_i,l_j-1,00} \tag{12.8}$$

Here, N is part of the recurrence relation. Nevertheless, the three-dimensional dependency pattern (l_i, l_j, N) can be visualized.

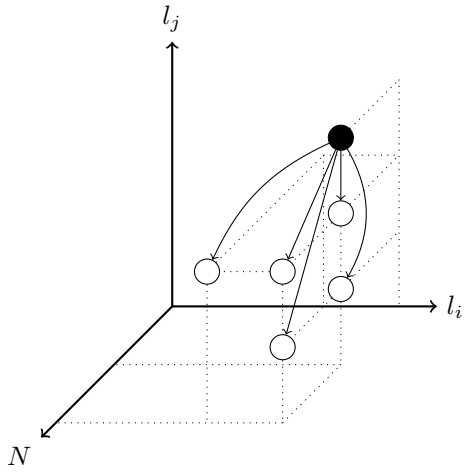


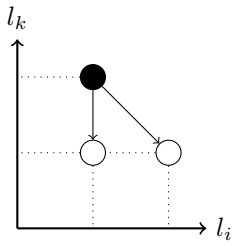
Figure 12.4.: Batch dependencies for the electron transfer E_2.

The two horizontal transfers

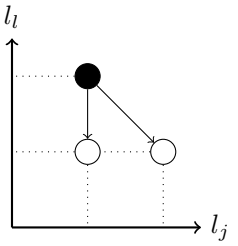
$$H1 : \Theta^N_{l_i,l_j,l_k+1,l_l} \leftarrow \Theta^N_{l_i+1,j,k,l},\ \Theta^N_{i,j,k,l} \tag{12.9}$$

$$H2 : \Theta^N_{l_i,l_j,l_k,l_l+1} \leftarrow \Theta^N_{l_i,j+1,k,l},\ \Theta^N_{i,j,k,l} \tag{12.10}$$

are two dimensional, as N is not affected.



(a) Horizontal transfer H1.



(b) Horizontal transfer H2.

Figure 12.5.: Batch dependencies for the horizontal transfers.

12.1.3. Transfer Order

Generally, the recursion pathway for a given integral is not unique, several possibilities exist. Considering the integral $\langle f_{xyz}p_x|g_{y^2z^2}s\rangle$, the electron transfer can be applied to the bra2 index, in x-direction. Another possibility is the application of the H1 transfer, in either y- or z-direction.

In this work, the type of transfer to be used will be determined by the angular pattern of the particular batch. The integral of the example above belongs to a $FPGS$ batch. The only recursion relation which can be applied to all integrals is the horizontal transfer H1, decrementing the ket1 shell from g to f. The electron transfer mentioned in the example above, for example, can not be applied to the integral $\langle f_{x^3}p_x|g_{x^4}s\rangle$.

Within some batch, given a recursion relation, the direction of the transfer is chosen for each integral individually. Transfer directions are checked for in the order x, y and z, and taken if possible. For the $\langle f_{xyz}p_x|g_{y^2z^2}s\rangle$ integral, H1 would be applied in y-direction.

Optimal Shell Permutation

Due to permutational symmetry up to eight symmetry equivalent representations of a batch can exist, of which only one is calculated. The particular pattern appearing in the calculation depends on the iteration. Using the iterators of section 10.2.2, on a center pattern of four different centers, the batches having one d and three s-shells will appear as $DSSS$, $SDSS$, $SSDS$ and $SSSD$, respectively.

Of these four patterns, the $DSSS$ ordering is optimal in terms of the Obara-Saika recurrence relations, as only the vertical transfer has to be applied. The other three, however, can be brought to match this optimal pattern by permuting them according to the symmetry-equivalence permutations of section 4.1.

Generally, as the Obara-Saika recurrence relations shift angular momenta from the left to the right, the optimal shell permutation is the one with the maximum total angular momentum as far to the left as possible. A class has been included to the QOL [18] to accomplish this, the `OptimalShellOrderingPermutation`. Using it, each batch appearing in a calculation is brought to its optimal permutation, minimizing the amount of recurrence relations to be applied.

12.2. Existing Implementations

Starting points of the implementation of the Obara-Saika integration scheme have been two approaches already implemented into the QOL framework by Michael Hanrath [18]. A careful examination will show various flaws and benefits of these implementations, depending on the approach chosen. In both cases, the electron transfer E_1 of equation (12.2) has been used.

12.2.1. Generic Implementation

In this ansatz, integral evaluation is handled by one generic function, capable of processing any shell batch given. On the one side, this has the positive effect of possessing a very small code footprint, thus not polluting the instruction cache. On the other hand, handling all shell batches in a generic way introduces a lot of redundancies. To illustrate this, listing 12.1 shows the code for the vertical transfer V (equation (12.1)).

```
1  for (int l=1; l<maxL; ++l) // batch loop
2    for (CartesianShellIterator i(l); i.valid(); ++i) // shell loop
       (i=1: p (x y z))
3      for (int N=0; N<maxL-l; ++N) // boys order loop
4      {
5        int direction = i.maxPower();
6        switch (direction)
7        {
8          case 0: // recursion in x
9            t1(i.x, i.y, i.z)[N] =
10             + this->PA.x    * t1.g(i.x-1, i.y, i.z)[N]
11             - this->alpha_p_PQ.x * t1.g(i.x-1, i.y, i.z)[N+1]
12             + (i.x.l()-1 > 0 ?
13                 (i.x.l()-1)*this->_2p * (t1.g(i.x-2, i.y, i.z)[N] -
                    this->alpha_p * t1.g(i.x-2, i.y, i.z)[N+1])
14                 : 0
15               );
16           break;
17
18         case 1: // recursion in y
19           // as above, in y
20           break;
21
22         case 2: // recursion in z
23           // as above, in z
24           break;
25       }
26     }
```

Listing 12.1: generic, vertical

In general, the information how to process a given integral has to be regenerated for each target and intermediate, at runtime. In line 5 the direction of the transfer is queried, to be used in the `switch` statement of line 6. For the actual calculation (line 9, in x-direction), the necessary integrals have to be retrieved from the container `t1`. Due to

the form of the vertical transfer, another `if` statement has to be processed (line 12), to determine if the terms three and four of the vertical transfer have to be included. All of this information solely depends on the angular momenta of the processed batch, thus being constant for all batches of the same angular pattern. For all batches of a given angular momentum, the only variables are the prefactors (i.e. `this->PA.x` in line 10). For example, in the calculation of alanine in a cc-pVTZ basis, about six million different $PSSS$ batches survive prescreening (using a threshold of 10^{-11}). Looking at the corresponding six million $p_x sss$-integrals, the transfer direction x is reevaluated six million times, followed by entering the corresponding `case`, calculation of the container indices and neglection of terms three and four. Clearly, the reevaluation of this information is redundant. Additionally, most of the redundant operations correspond to conditional statements, leading to possibly inefficient branching. Obviously, the same considerations apply to integrals involving all four recurrence relations, in an even more pronounced way.

To store the target- and intermediate integrals, the four-dimensional structure `t1` is used. Three dimensions correspond to the Cartesian powers i_x, i_y and i_z of the bra1 shell, the fourth to the order N of the auxiliary integral. To store all auxiliary integrals of a given N, a simple three dimensional array with dimensions of $(l+1)$ is used, one dimension for each direction. This form greatly simplifies element access, but is far from memory efficient. To store integrals up to d-functions, a $(3 \times 3 \times 3)$ array is needed, holding 27 elements. However, for up to d-functions only ten integrals exist (one s, three p and six d), thus most of the space allocated by the array is unused. The reason for this is that in the plain $(l+1)^3$ approach, space is allocated for unneeded integrals of higher angular momenta. In the $(3 \times 3 \times 3)$ array for up to d-functions for example the $2, 2, 2$ index is available, corresponding to the unneeded $l = 6$ integral $i_{x^2y^2z^2}sss$. Additionally, the ten needed integrals are not stored consecutively, introducing gaps in memory. These considerations hold for the four-dimensional integrals accessible solely by the vertical transfer. In the general case, thirteen dimensions are needed to identify an integral, four times three Cartesian directions, plus the order N. The above mentioned problems of unused elements and non-consecutive storage are even more dominant in such a thirteen dimensional storage container.

12.2.2. The Code-Generation Approach

These shortcomings are efficiently addressed by the code-generated ansatz. The key point is the decoupling of recursion structure and runtime execution. At compile time, the structure needed to calculate some given batch, in terms of its angular pattern, is analyzed. Based on this analysis, optimized code is generated to perform the integration for all batches of this particular angular pattern. These codes are then called at runtime. It shall be noted that a two-fold approach has been chosen. In the first step, the vertical- and electron transfer relations are applied. In the second step, if necessary, the target integrals are generated via the horizontal transfers.

As an example, a $DSSS$-batch shall be analyzed. Its recursion relations are shown in figure 12.6.

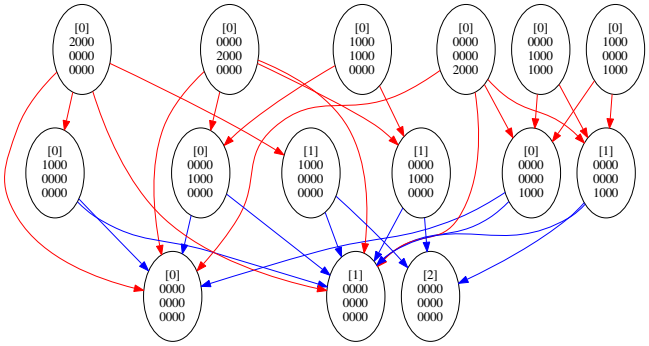


Figure 12.6.: Intermediates needed to calculate all integrals of a *dsss*-batch, and their dependencies.

The code produced for this batch using the code-generated ansatz is shown in listing 12.2.

```
1  void dsss(
2   const shellBatchData<double> & sd,
3   const double * boys,
4   double * out)
5  {
6   double tmp[2];
7   tmp[0] = sd.PAx * boys[0] - sd.apPQx * boys[1];
8   tmp[1] = sd.PAx * boys[1] - sd.apPQx * boys[2];
9   out[0] = sd.PAx * tmp[0]  - sd.apPQx * tmp[1] + sd._2p * (boys[0] -
      sd.ap * boys[1]);
10  tmp[0] = sd.PAy * boys[0] - sd.apPQy * boys[1];
11  tmp[1] = sd.PAy * boys[1] - sd.apPQy * boys[2];
12  out[1] = sd.PAx * tmp[0]  - sd.apPQx * tmp[1];
13  out[2] = sd.PAy * tmp[0]  - sd.apPQy * tmp[1] + sd._2p * (boys[0] -
      sd.ap * boys[1]);
14  tmp[0] = sd.PAz * boys[0] - sd.apPQz * boys[1];
15  tmp[1] = sd.PAz * boys[1] - sd.apPQz * boys[2];
16  out[3] = sd.PAx * tmp[0]  - sd.apPQx * tmp[1];
17  out[4] = sd.PAy * tmp[0]  - sd.apPQy * tmp[1];
18  out[5] = sd.PAz * tmp[0]  - sd.apPQz * tmp[1] + sd._2p * (boys[0] -
      sd.ap * boys[1]);
19 }
```

Listing 12.2: Generated code for the *dsss* batch.

Clearly, no conditionals have to be evaluated. Additionally, memory utilization can be optimized rather efficiently. The intermediates are saved in the array `tmp`, and overwritten if not needed anymore. For the six intermediates of the *dsss* batch, only space for two elements has to be allocated. In addition to those, three recursion seeds (stored in the array `boys`) are needed. With the six target integrals (stored in `out`), the total number of elements to store summarizes to eleven elements, all of which appear consecutively in memory. In the generic ansatz, for this batch 81 elements had to be allocated ($(3 \times 3 \times 3)$ for the Cartesian directions, times 3 for N).

The drastic increase in performance of the code-generated ansatz, compared to the generic implementation, is shown in table 12.1. Shown are the results obtained for the first SCF iteration of alanine in a cc-pVTZ basis. Prescreening has been enabled for both runs, using a threshold of 10^{-11}. The density used was the initial density obtained with TURBOMOLE.

	Generic	Code-generated	Factor
Time (seconds)	286	37	7.7
Cycles	724,599,731,069	92,883,969,691	7.8
Branches	130,540,564,523	2,996,474,218	43.6
Branch misses	2.74%	1.88%	-
L1 iCache loads	543,403,235,479	54,983,338,589	9.9
L1 iCache load-misses	0.44%	16.75%	-
L1 dCache loads	517,098,487,403	42,792,799,864	12.1
L1 dCache load-misses	2.35%	1.01%	-
L1 dCache stores	130,425,312,803	23,222,007,623	5.6
L1 dCache store-misses	3.37%	0.24%	-

Table 12.1.: Performance comparison of the generic versus the code-generated approach for alanine in a cc-pVTZ basis.

Even though the shown calculations correspond to a full SCF iteration, changing only the integration kernel from generic to code-generated results in a significant decrease of all shown performance counters, giving a total speedup of about a factor of eight from 286 seconds to 37 seconds of wall clock time. Of special interest is the almost 50-fold decrease of performed branches, clearly related to the conditionless realization of the Obara-Saika recurrence relations in the code-generated ansatz.

Of particular note however are the iCache loads. Although reduced by an order of a magnitude in the code-generated implementation, the percentage of missed iCache loads increases to almost 17 percent. This is due to the amount of code needed to calculate a batch. Generally speaking, one line of code is generated for each integral and intermediate. For the $FFSS$ batch with 100 target integrals, 554 intermediates have to be calculated, resulting in a code size of over 654 lines. The resulting byte-code has a size of about 30K, which almost fills the typical instruction cache size of 32K.

Unfortunately, the amount of code generated not only has implications on the runtime, but also on the compile time. The 120524 lines of code needed to calculate an $FFFF$-batch put, even if chunked to 1000 lines of code per file, time critical strains on the compiler. In fact, generating batches involving g-functions or higher produces an impractical amount of code. Table 12.2 shows the amount of generated code for the vertical-, electron- and horizontal transfer substructures, in dependence of the maximum angular momentum l.

l_{max}	V/E_1	H1/H2
0	222	204
1	1,394	1,160
2	18,153	30,879
3	178,857	608,804
4	1,176,162	7,124,017
5	5,755,160	57,529,321
6	22,659,255	-

Table 12.2.: Lines of generated code for vertical/electron transfer(V,E_1) and horizontal (H1 and H2) recursions as a function of l_{max}.

Clearly, the lions share corresponds to the horizontal transfers, by which most of the target integrals are generated.

12.3. Numerical Instabilities

In the existing approaches presented above, the four step Obara-Saika recurrence relations [35–37] are used, employing the electron transfer E_1 (equation (12.2)). This ansatz, in particular the E_1 transfer, suffers from numerical instabilities. The mechanism responsible is comparable to the one found in the calculation of the prefactor X_{PA}, as discussed in section 9.1.2. In particular, terms present in the vertical transfer (12.1) get canceled by terms present in the electron transfer (12.2). If those terms get large, the numerical information of the smaller ones is lost, thus introducing numerical instabilities.

The source of the problem can be analyzed by inspecting the first electron transfer level of E_1, i.e. setting j in equation (12.2) to zero.

$$\Theta^N_{i,1,00} = -\frac{b}{q} X_{AB} \Theta^N_{i,000} + \frac{i}{2q} \Theta^N_{i-1,000} - \frac{d}{q} X_{CD} \Theta^N_{i,000} - \frac{p}{q} \Theta^N_{i+1,000} \tag{12.11}$$

The last term, $-\frac{p}{q}\Theta^N_{i+1,000}$, contains the vertical transfer (12.1). Insertion gives

$$\begin{aligned} \Theta^N_{i,1,00} = & \boxed{-\tfrac{b}{q} X_{AB} \Theta^N_{i,000}} + \boxed{\tfrac{i}{2q} \Theta^N_{i-1,000}} - \frac{d}{q} X_{CD} \Theta^N_{i,000} \\ & \boxed{+\tfrac{p}{q}\tfrac{b}{p} X_{AB} \Theta^N_{i,000}} - \boxed{\tfrac{p}{q}\tfrac{i}{2p} \Theta^N_{i-1,000}} + \frac{p}{q}\frac{\alpha}{p} X_{PQ} \Theta^{N+1}_{i,000} + \frac{p}{q}\frac{i\alpha}{2p^2} \Theta^{N+1}_{i-1,000} \end{aligned} \tag{12.12}$$

The terms $\frac{b}{q} X_{AB} \Theta^N_{i,000}$ and $\frac{i}{2q}\Theta^N_{i-1,000}$ appear in both transfer relations and cancel each other. In this work, the Obara-Saika recurrence relations are solved iteratively. To calculate e.g. $\Theta^N_{i,1,00}$, first all intermediates are evaluated. Thus, similar to the calculation of X_{PA}, if the canceling terms $\frac{b}{q} X_{AB} \Theta^N_{i,000}$ and $\frac{i}{2q}\Theta^N_{i-1,000}$ are sufficiently larger in magnitude than the surviving terms numerical noise is introduced.

Unfortunately, this cancellation of terms is not only present for electron transfers with $j = 0$, but also for higher electron transfers. For such cases, repetitive insertion of the

adequate intermediate formulas finally leads to the $\Theta^N_{i+1,000}$ terms, again introducing the canceling terms of equation (12.12).

An in depth analysis of the conditions leading to numerical instabilities is non-trivial. Large values of p $(= a + b)$ compared to q $(= c + d)$ could be identified to produce numerical errors. In the case of large b's, inspection of the vertical transfer

$$\Theta^N_{i+1,000} = -\frac{b}{p}\Theta^N_{i,000} + \frac{i}{2p}\Theta^N_{i-1,000} - \frac{q}{p+q}X_{PQ}\Theta^{N+1}_{i,000} + -\frac{i}{2p}\frac{q}{p+q}\Theta^{N+1}_{i-1,000}$$

shows that the prefactor of the first term approaches one, while the prefactors of the remaining terms approach zero. Thus, at least parts of the numerical information of the remaining terms vanishes due to the significantly larger first term. As this is just the term which gets canceled, the exact result is lost.

For large a, a similar, simple interpretation is not possible. The prefactor of the last term of equation (12.3) tends more rapidly to zero than the prefactors of the remaining terms. To estimate their relative magnitudes, the prefactors alone are not sufficient, the intermediates would have to be included.

However, if p gets large compared to q, the prefactor $\frac{p}{q}$ amplifies the value of $\Theta^N_{i+1,000}$ in equation (12.12), including possible numerical errors. Additionally this term then becomes the largest (leading) contribution, further destroying numerical information.

To analyze the numerical properties of the Obara-Saika scheme, the high resolution floating point type of the `boost` library introduced in section 9.1.2 has been used. Via the `template` mechanism of `C++`, the same piece of code can be used to be executed with the desired type.

The subtlety of the numerical instabilities of the Obara-Saika scheme is shown in table 12.3. Here the numerical values of the intermediates needed to calculate a $f_{x^3}g_{x^4}ss$ integral are shown. The integrals are part of the calculation of alanine in a cc-pVTZ basis. The particular center pattern was $C_1C_3O_1C_3$, with the exponents $a = 0.761$, $b = 15330$, $c = 0.318$ and $d = 0.318$. The results obtained with the high accuracy type using 500 digits are compared to the results using the standard `double` type, showing all equal significant digits plus the first differing one.

Integral	boost<500>	double
	Vertical Transfer	
psss	$2.782\,239\,881\,132\,851\,5 \cdot 10^{-7}$	$2.782\,239\,881\,132\,851\,7 \cdot 10^{-7}$
dsss	$6.095\,248\,400\,768\,226 \cdot 10^{-7}$	$6.095\,248\,400\,768\,227 \cdot 10^{-7}$
fsss	$1.335\,337\,969\,206\,090 \cdot 10^{-6}$	$1.335\,337\,969\,206\,091 \cdot 10^{-6}$
gsss	$2.925\,458\,474\,266\,312 \cdot 10^{-6}$	$2.925\,458\,474\,266\,314 \cdot 10^{-6}$
hsss	$6.409\,138\,087\,948\,996 \cdot 10^{-6}$	$6.409\,138\,087\,948\,999 \cdot 10^{-6}$
isss	$1.404\,133\,078\,844\,736 \cdot 10^{-5}$	$1.404\,133\,078\,844\,737 \cdot 10^{-5}$
jsss	$3.076\,237\,510\,475\,794 \cdot 10^{-5}$	$3.076\,237\,510\,475\,796 \cdot 10^{-5}$
	Electron Transfer	
spss	$1.797\,123\,976\,7 \cdot 10^{-8}$	$1.797\,123\,976\,6 \cdot 10^{-8}$
ppss	$3.937\,053\,127\,8 \cdot 10^{-8}$	$3.937\,053\,127\,6 \cdot 10^{-8}$
dpss	$8.625\,166\,023\,6 \cdot 10^{-8}$	$8.625\,166\,023\,0 \cdot 10^{-8}$
fpss	$1.889\,585\,739\,5 \cdot 10^{-7}$	$1.889\,585\,739\,6 \cdot 10^{-7}$
gpss	$4.139\,698\,319\,91 \cdot 10^{-7}$	$4.139\,698\,319\,97 \cdot 10^{-7}$
hpss	$9.069\,299\,301\,1 \cdot 10^{-7}$	$9.069\,299\,301\,5 \cdot 10^{-7}$
ipss	$1.986\,926\,156\,3 \cdot 10^{-6}$	$1.986\,926\,156\,2 \cdot 10^{-6}$
pdss	$2.274\,101\,43 \cdot 10^{-7}$	$2.274\,101\,41 \cdot 10^{-7}$
ddss	$4.982\,03 \cdot 10^{-7}$	$4.982\,02 \cdot 10^{-7}$
fdss	$1.091\,456\,5 \cdot 10^{-6}$	$1.091\,456\,8 \cdot 10^{-6}$
gdss	$2.391\,162\,8 \cdot 10^{-6}$	$2.391\,162\,2 \cdot 10^{-6}$
hdss	$5.238\,595 \cdot 10^{-6}$	$5.238\,598 \cdot 10^{-6}$
dfss	$2.0 \cdot 10^{-7}$	$1.6 \cdot 10^{-7}$
ffss	$4.59 \cdot 10^{-7}$	$4.9 \cdot 10^{-7}$
gfss	$9.9 \cdot 10^{-7}$	$8.8 \cdot 10^{-7}$
fgss	$2.6 \cdot 10^{-6}$	0.0045

Table 12.3.: Values for the $f_{x^3} g_{x^4} ss$-integral and needed intermediates (all in x-direction). The values were calculated in double precision and with the `boost` 500 digits type, respectively. The accuracy obtained is indicated by the amount of digits. Shown are all equal digits and the first differing one.

In the final integral, the actual value of about 10^{-6} has been overestimated by a factor of approximately 1700, giving a totally wrong number and magnitude of 0.0045 in the `double` calculation. At the vertical transfer stage, no numerical errors can be observed. At the electron transfer stage, however, the accuracy decreases with increasing j, loosing about four to five significant digits at each level.

12.3.1. Solutions

Swapping

The first solution to the problem of numerical instabilities is based on their appearance for $p > q$. If such a case is encountered, the whole batch is permuted, swapped, to negate the condition. In particular, particle symmetry is used, and a batch $ACBD$ is permuted to $CADB$. Although this leads to numerically stable code, as shown in section 12.3.2, the implications in terms of code efficiency are not optimal. Using this approach introduces several complications to the production code, which will be discussed in section 12.4.4.

Algebraic Solution

The algebraic solution is based on equation (12.12). For $j = 0, 1$ and 2 canceling terms have been removed from the electron transfer E_1 by repeated insertion and reapplication of the appropriate recurrence relations. The following equations have been obtained.

$$\begin{aligned}
\Theta^N_{i,100} &= \frac{d}{q} X_{CD} \Theta^N_{i000} - \frac{\alpha}{q} X_{PQ} \Theta^{N+1}_{i,000} + \frac{i}{2(p+q)} \Theta^{N+1}_{i-1,000} \\
\Theta^N_{i,2,00} &= \frac{d}{q} X_{CD} \Theta^N_{i100} + \frac{\alpha}{q} X_{PQ} \Theta^{N+1}_{i100} + \frac{i}{2(p+q)} \Theta^{N+1}_{i-1,100} \\
&\quad + \frac{1}{2q} \left(\Theta^N_{i,000} - \frac{\alpha}{q} \Theta^{N+1}_{i,000} \right) \qquad (12.13) \\
\Theta^N_{i,3,00} &= \frac{d}{q} X_{CD} \Theta^N_{i200} + \frac{\alpha}{q} X_{PQ} \Theta^{N+1}_{i200} + \frac{i}{2(p+q)} \Theta^{N+1}_{i-1,200} \\
&\quad + \frac{2}{2q} \left(\Theta^N_{i,1,00} - \frac{\alpha}{q} \Theta^{N+1}_{i,1,00} \right) \qquad (12.14)
\end{aligned}$$

These are just the first solutions for the alternative electron transfer, which has been obtained from the original Obara-Saika recurrence relation.

$$\begin{aligned}
\Theta^N_{i,j+1,00} &= \frac{d}{q} X_{CD} \Theta^N_{ij00} + \frac{\alpha}{q} X_{PQ} \Theta^{N+1}_{ij00} + \frac{i}{2(p+q)} \Theta^{N+1}_{i-1,j00} \\
&\quad + \frac{j}{2q} \left(\Theta^N_{i,j-1,00} - \frac{\alpha}{q} \Theta^{N+1}_{i,j-1,00} \right) \qquad (12.15)
\end{aligned}$$

The alternative electron transfer (12.15) indeed turns out to be numerically stable (see section 12.3.2), without the need of swapping. Additionally, as will be shown in section 12.4.4, its structure is far better suited for the implementation presented there.

12.3.2. Results

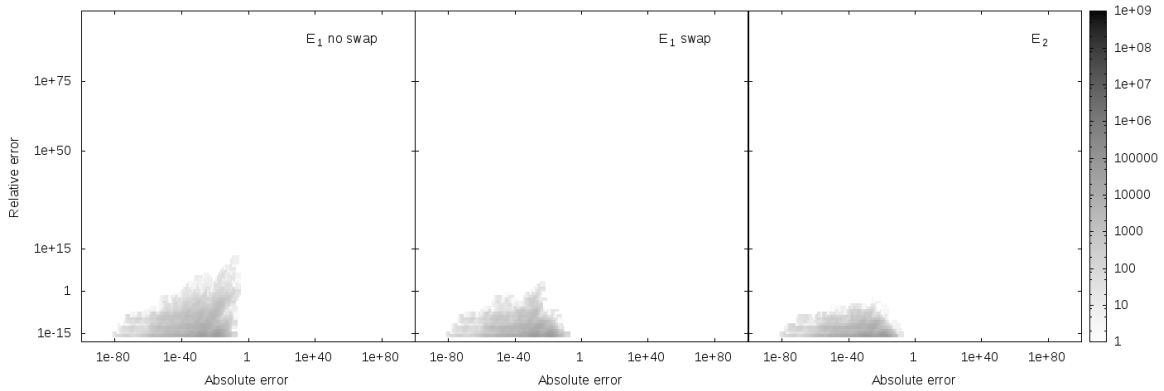
Three implementations of the integration have been analyzed with respect to their numerical stability. On the one hand the electron transfer E_1 has been examined, using the swapped and unswapped ansatz. Additionally, the numerical properties of the alternative electron transfer E_2 have been included, using the implementation of section 12.4. All three have been set up to use the standard `double` floating point type. For the comparison, the high accuracy `boost` type has been used, with 500 significant digits. It has been set up to perform the integration without swapping, using the electron transfer E_1.

The analysis has been performed on the H_4 system, using an artificial basis. This artificial basis has been chosen to examine a wide range of exponents. Starting point for the artificial basis were the minimum and maximum exponents for up to i-functions. The set of atoms they were chosen from was H, C, N, O, S and Cl and the basis sets were cc-pV{D,T,Q,5,6}Z, Def2-{S,TZ,QZ}VP, roos-aug-dz,tz-ano and ano-rcc. For l values larger than one, the maximum exponent has been scaled by a factor of ten. Additionally, the exponents 1 and 5 have been added to each angular momentum. The used exponents are summarized in table 12.4.

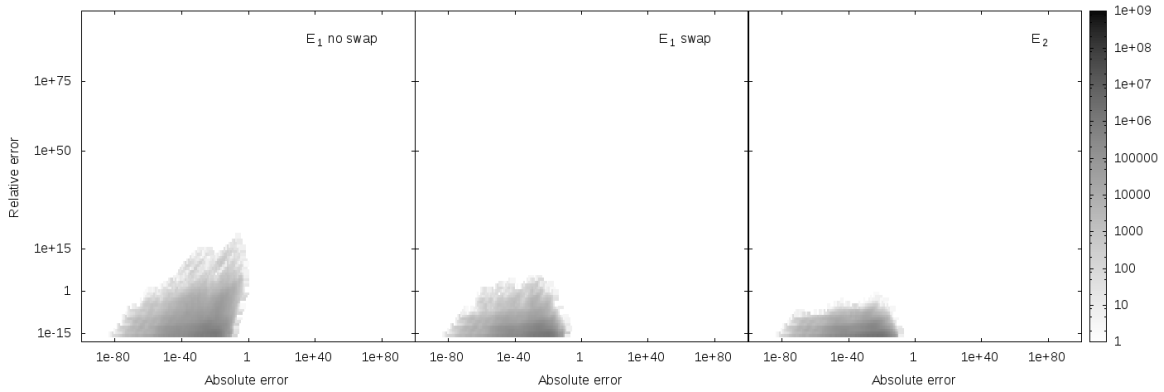
$l =$	0	1	2	3	4	5	6
	0.02526	0.0229	0.0798831	0.1010651	0.25	0.6115	1.0409000
	1	1	1	1	1	1	1
	5	5	5	5	5	5	5
	7733000	6091	82.53	54.3	52.11	38.72	27.73

Table 12.4.: Exponents used in the artificial basis.

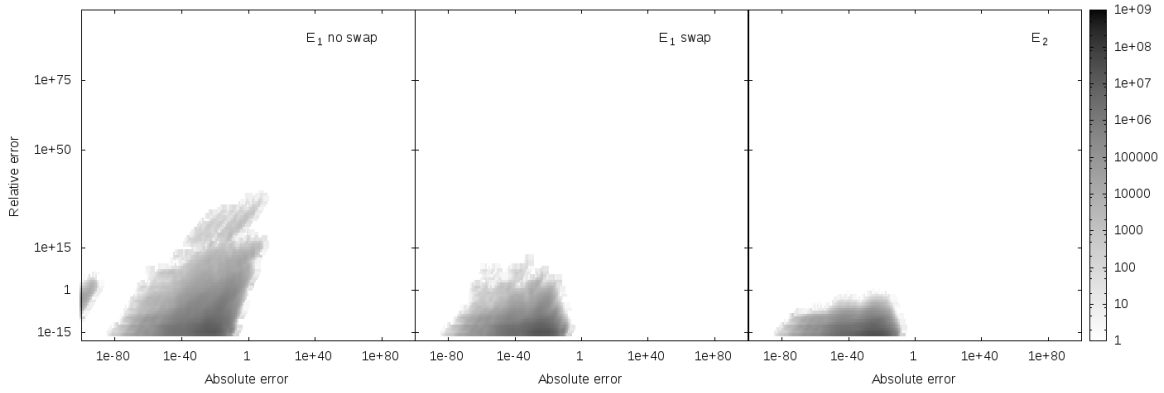
To get the full picture, for each integral calculated using the three pathways described above, the absolute and relative errors with respect to the value obtained with the `boost` high accuracy data type have been calculated. The frequency of the absolute/relative error distribution has then been determined. This has been accomplished using a grid, spanning the range of 10^{-100} to 10^{100}, in 200 logarithmic steps, respectively. Additionally, integrals have been assigned to sets of batches, depending on their angular momentum. I.e. for $l = 2$, only batches with at least one angular momentum equal to two and the rest smaller or equal to two have been taken into account. The results obtained for the H_4 molecule in the artificial basis are shown in the figures 12.7a to 12.7f. The horizontal line at $r = 1$ indicates integrals with an absolute error of the magnitude of the actual integral. The second horizontal line at $r = 10^{-8}$ indicates seven reproduced significant digits.



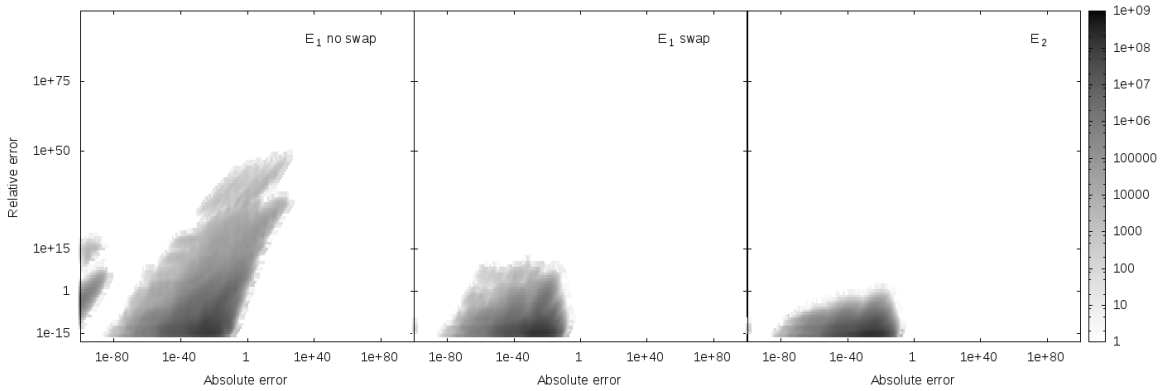
(a) Maximum angular momentum $l = 1$.



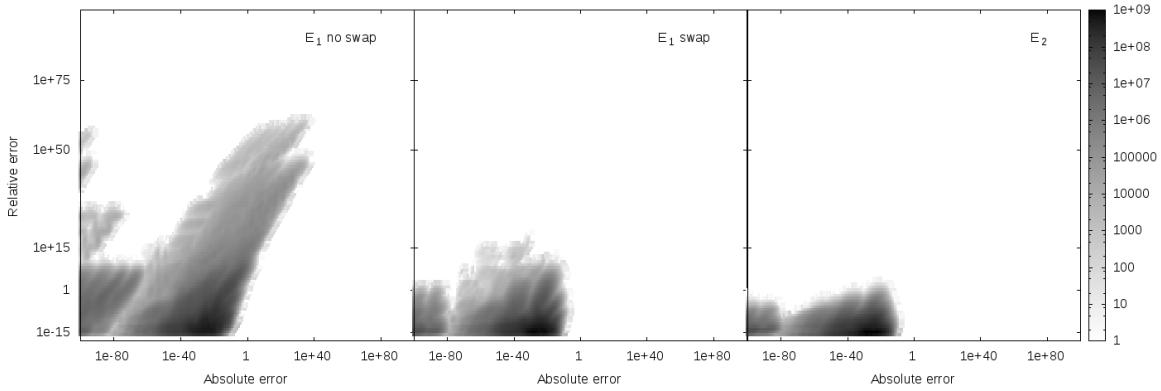
(b) Maximum angular momentum $l = 2$.



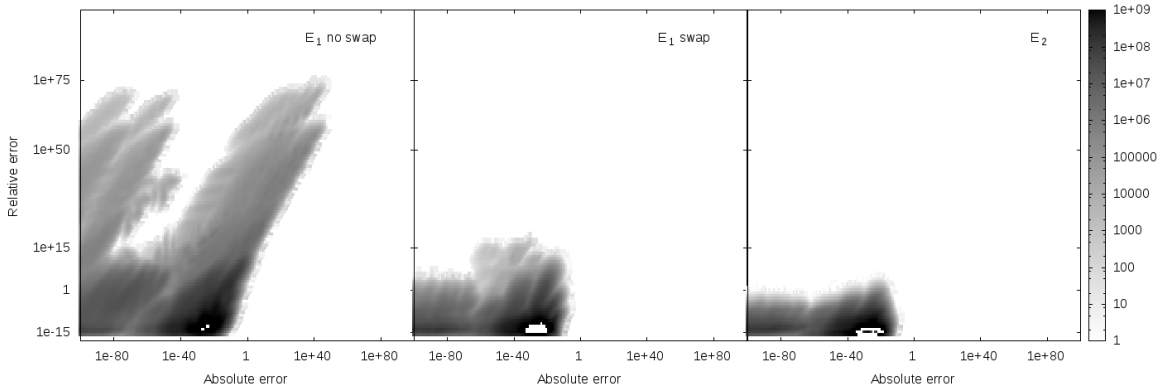
(c) Maximum angular momentum $l = 3$.



(d) Maximum angular momentum $l = 4$.



(e) Maximum angular momentum $l = 5$.



(f) Maximum angular momentum $l = 6$.

Figure 12.7.: Frequencies of absolute/relative error correlations for H_4 in the artificial basis, with respect to the maximum angular momentum of the integrals. Shown are comparisons of the `boost` 500 type to the E_1 transfer in its swapped and unswapped realization, and to the E_2 transfer, all in double precision.

Clearly, in the unswapped case, the error distributions are shifted to high absolute and relative errors, with increasing l. For $l = 6$, the absolute errors reach values of 10^{50}, with corresponding relative errors of 10^{80}. In the unswapped case, although the relative error increases with increasing l to values of about 10^{20}, the absolute error nevertheless stays smaller than one. An even better trend can be observed for the alternative electron transfer calculations. Although the relative error still increases to values larger than one, the overall distribution indicates mostly a reproduction of under seven significant digits.

To further investigate the error distribution, absolute and relative errors have been determined with respect to the magnitude of the integral. The results for H_4 in the artificial basis are shown in figure 12.8. For all integrals of a certain magnitude 10^i, the corresponding maximum absolute and relative errors are shown.

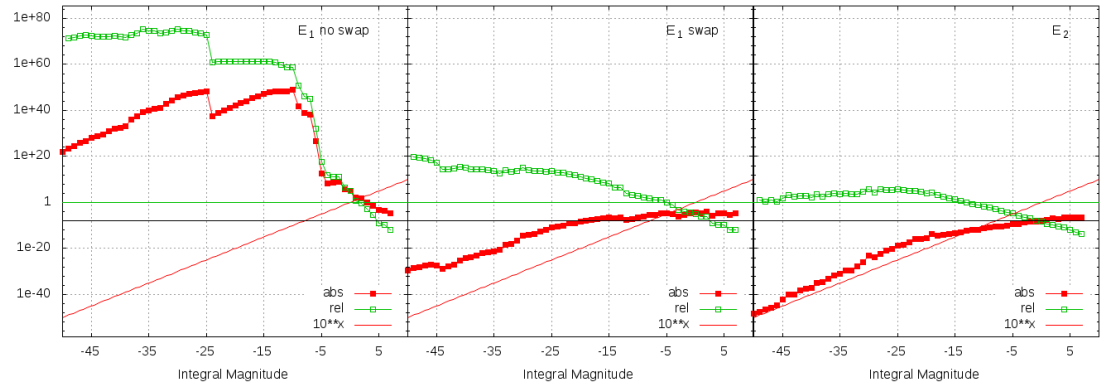


Figure 12.8.: Distribution of maximum absolute and relative errors for H_4 in the artificial basis, with respect to the magnitudes of the integrals. In all cases the integrals are compared to the BOOST<500> type. On the left the deviations of the unswapped E_1 ansatz are shown. In the middle the swapped E_1 results are shown, and on the right the E_2 approach.

In all three implementations, the maximum errors can be found for integrals of small magnitude. In the unswapped case, however, the maximum relative and absolute errors are huge. I.e. the correct integral of $4.00 \cdot 10^{-26}$ gets calculated to $1.1 \cdot 10^{48}$. Contrary, the other two implementations give absolute errors smaller than one. In the swapped implementation however, in all small integrals the maximum absolute error is larger than the integral (red line). The integral $4.6 \cdot 10^{-13}$ for example is calculated to $4.6 \cdot 10^{-7}$. In the alternative electron transfer case finally, the relative error is even smaller. The absolute error grows to the magnitude of the corresponding integral in the range of about 10^{-30} to 10^{-15}. The integral of value $-1.34 \cdot 10^{-25}$ is calculated to $-5.9 \cdot 10^{-20}$.

Still, some integrals are calculated to a significantly different number than the `boost` 500 value. The reason for this overestimation is numerical noise. The integrals are built from a huge sum, with summands of differing magnitudes. Especially if the final result is small, such sums are susceptible to numerical noise. Even two almost canceling summands of higher magnitude, appearing somewhere in the sum taint the final result.

As for certain exponent combinations the factor p/q still gets large, such noise can not be avoided.

The use of the artificial basis with exaggerated exponents amplifies this noise. This can be seen by comparing the results obtained in this artificial basis to the results of the methanolamine calculation in a unmodified cc-pV6Z basis shown in figure 12.9.

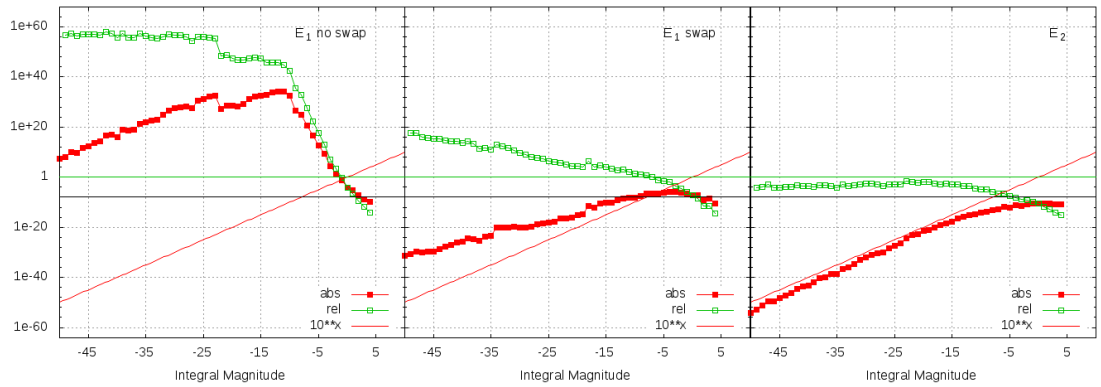


Figure 12.9.: Distribution of maximum absolute and relative errors for methanolamine CH_5NO in the cc-pV6Z basis, with respect to the magnitudes of the integrals. In all cases the integrals are compared to the BOOST<500> type. On the left the deviations of the unswapped E_1 ansatz are shown. In the middle the swapped E_1 results are shown, and on the right the E_2 approach.

Still not free of errors, the alternative electron transfer ansatz reproduces all integrals to at least one significant digit. Additionally, all potentially problematic deviations are linked to integrals of small magnitude.

The error introduced in the SCF energy has not been investigated in terms of the high accuracy type. It shall however be noted that the extensive comparisons versus TURBOMOLE and MOLPRO in section 15.1.1 and section 15.2.1 did not show any significant deviations.

12.4. Code Reduction by Using a Different Code-Generating Ansatz

Although highly efficient for batches with small angular momenta, the code-generated ansatz of section 12.2.2 becomes impractical if applied to large angular momenta. This is due to the amount of generated code. At runtime the huge bytecode size leads to costly instruction cache misses, increasing with the size (angular momenta) of the batches. At compile time, on one hand the actual generation of the recursion structure becomes rather demanding. To use, for example, basis sets of cc-pV6Z quality, $IIII$-batches have to be calculated. Having 28 functions in an I-shell, 614,656 target integrals need to be calculated for one batch. To do so, 9,285,371 intermediates are needed, giving a total of 9,900,027 integrals to be taken into account. No timings have been generated for the

analysis of such a batch. However, to analyze the 84,181 integrals of an $FFFF$ batch, about six seconds are needed, and about twelve seconds for the 122,523 integrals of an $GFFF$ batch. On the other hand, compilation of codes of the size of millions of lines of codes becomes almost impossible.

In the following, two techniques are presented, by which the amount of generated code has been drastically reduced. Still, the main objective has been a code-generated ansatz, thus evaluating the recursion structure at compile time.

12.4.1. Removing Redundancies, the Splitted Integration Ansatz

In a first step, analysis of the existing code-generated ansatz revealed that a substantial amount of the code generated is in fact redundant. This is conceptually shown in figure 12.10, which gives the full V and E_1 dependencies of a $FFSS$ batch, in terms of the batch-recursion interpretation of section 12.1.2.

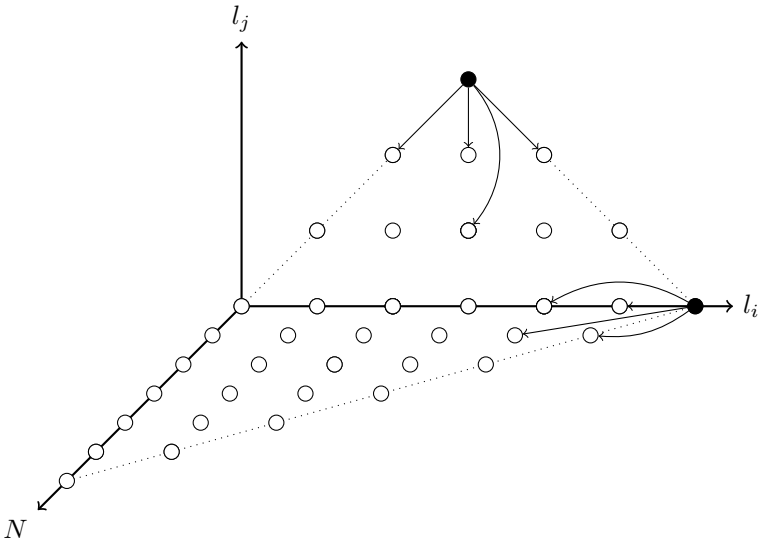


Figure 12.10.: Full recursion of a $FFSS$ batch, using E_1 and V.

In the existing code-generated ansatz, the code to calculate such an $FFSS$ batch contains code for all intermediates. The redundancies now become clear by considering that each intermediate batch needed to calculate the $FFSS$ batch can be a target batch itself. As a target batch, again all intermediates have to be included in its code. For example the code to calculate the $FDSS$ batch is contained and repeated in the code of the $FFSS$ batch. In fact the code to calculate a specific batch explicitly appears in each target batch it is an intermediate of.

To remove these redundancies, a 'splitted' ansatz has been developed to implement the Obara-Saika recurrence relations. In a first step, code is generated to calculate each batch individually, from their actual dependencies. This is shown in listing 12.3, for the $PPSS$ batch, using the electron transfer E_1. According to the batch dependencies (12.7), the $PPSS$ batch depends on the $PSSS$- and $DSSS$ batches, and the recursion seeds $SSSS$.

```
1  void PPSS(
2      double * out,
3      const shellBatchData & sbd,
4      const double * psss,
5      const double * dsss,
6      const double * ssss)
7  {
8    out[0] = sbd.b_AB_d_CD_q.x * psss[0] - sbd.p_q * dsss[0] + sbd._2q
       * ssss[0] ;
9    out[1] = sbd.b_AB_d_CD_q.y * psss[0] - sbd.p_q * dsss[1];
10   out[2] = sbd.b_AB_d_CD_q.z * psss[0] - sbd.p_q * dsss[3];
11
12   out[3] = sbd.b_AB_d_CD_q.x * psss[1] - sbd.p_q * dsss[1];
13   out[4] = sbd.b_AB_d_CD_q.y * psss[1] - sbd.p_q * dsss[2] + sbd._2q
       * ssss[0];
14   out[5] = sbd.b_AB_d_CD_q.z * psss[1] - sbd.p_q * dsss[4];
15
16   out[6] = sbd.b_AB_d_CD_q.x * psss[2] - sbd.p_q * dsss[3];
17   out[7] = sbd.b_AB_d_CD_q.y * psss[2] - sbd.p_q * dsss[4];
18   out[8] = sbd.b_AB_d_CD_q.z * psss[2] - sbd.p_q * dsss[5] + sbd._2q
       * ssss[0];
19 }
```

Listing 12.3: Calculating a $PPSS$ batch individually from its dependencies.

Again, using the code-generated approach, no conditionals appear. The information that the second term of equation (12.2) vanishes in all target integrals, and that the third term only contributes to the first, fifth and ninth integral are determined at compile time.

The indices used to access the individual integrals are determined by the particular order in which the individual functions of a shell are stored. A detailed discussion of this order will be given in section 12.4.2.

These individual snippets are then bound together to calculate the respective target batches, using a separate function. Listing 12.4 shows this for the full $PPSS$ batch.

```
1  void ppss(
2      const shellBatchData & sbd,
3      const double * boys,
4      double * out)
5  {
6    // Vertical transfer
7    double psss[6];
8    PSSS<1>(psss, sbd, boys);
9
10   double dsss[6];
11   DSSS<0>(dsss, sbd, psss, boys);
12
13   // Electron transfer
14   PPSS(out, sbd, psss, dsss, boys);
15 }
```

Listing 12.4: Calculating a $PPSS$ batch individually from its dependencies.

Prior to the call to the PPSS function (line 14) of listing 12.3, which calculates the target integrals, the dependencies are calculated via the PSSS and DSSS functions (lines eight and eleven). These are implemented similar to the PPSS function, however using the corresponding vertical transfer relations. Note that they are templated with respect to the maximum order N which is needed for the particular batch (see section 12.4.3 for details).

Splitting the implementation of the integration in this way ensures that the size intensive evaluation code is implemented only once, for a given batch. These codes then do not have to be repeated explicitly every time a specific batch needs to be calculated, but can be replaced by a call to the appropriate function.

It shall be noted that with this ansatz the effective byte size of the codes to be called is not reduced. Nevertheless, using this approach makes batches of high angular momenta partly accessible at compile time.

One reason is the decreased generation time. The full thirteen dimensional recurrence relations need only to be taken into account in the individual batch calculation, where the integrals of one target batch are evaluated from their dependencies. In each of these individual codes, exactly one recurrence relation has to be applied.

To bind these functions together to calculate the requested target integrals, the much simpler, both in terms of dimensions and required intermediates, batch recursion formulas (12.6) to (12.10) of section 12.1.2 can be used.

It shall be noted that, in terms of compilation time, the overall code sizes are decreased. Nevertheless batches of high angular momenta still lead to impractical code sizes, as the calculation of one integral requires one line of code. A $GGGG$ batch, for example, contains 50,625 integrals, leading to the same amount of lines of code, even in the individual batch approach.

12.4.2. Shell Ordering and Index Relations

Central to the generation of the actual integration code is the effective handling of integral storage and access. Unfortunately, these two contradict each other. In terms of ease of access, as the auxiliary integrals are thirteen dimensional, a multidimensional array of the same dimension would provide the simplest access structure, accessing each integral via its Cartesian powers and order N in such an array. However, as pointed out in the discussion of the general integration approach in section 12.2.1, in doing so a lot of memory is allocated but unused, introducing unwanted gaps in memory. Additionally, the structure of such a multidimensional array leads to non-locality of integrals of a given batch in terms of their memory location. Integrals of a given batch end up scattered over a sometimes huge area of memory. As they however are needed at the same time (batchwise integration), reading them from differing memory locations is far from optimal.

This ease of access multidimensional structure has therefore been abandoned in favour of a tightly packed memory alignment of integrals of a given batch. This however leads to somewhat complicated access methods.

Doing so is closely connected to the specific ordering of basis functions of a shell. There is no unique ordering. A d-shell for example consists of the six functions d_{x^2}, d_{y^2}, d_{z^2}, d_{xy}, d_{xz} and d_{yz}. These six functions can be arranged in $6! = 720$ different ways. In the general case, having n functions in a shell, $n!$ possible orderings exist. In batches, the integrals can be thought of to be generated from the Cartesian product of the functions in the shells of this batch. The N resulting integrals can again be ordered in $N!$ different ways.

The ordering for basis functions and integrals in batches used in this work is not arbitrary, but based on the ordering defined by the QOL iterators. Starting point is the ordering of basis functions in shells, which is given in table 12.5, for shells up to f-functions.

l										
1	x	y	z							
2	xx	xy	yy	xz	yz	zz				
3	xxx	xxy	xyy	yyy	xxz	xyz	yyz	xzz	yzz	zzz
idx	0	1	2	3	4	5	6	7	8	9

Table 12.5.: Shell ordering for l up to three.

Integrals in batches are then stored in one-dimensional arrays. In particular, they are ordered in a 'right to left' fashion, with respect to the shells building the batch. In a $FDSS$ batch for example, the first six integrals correspond to the f_{x^3} function, combined with the d-functions, in the order given in table 12.5. The next six integrals are built from the second f-functions f_{x^2y} of table 12.5, again combined with the corresponding d-functions, and so on. Each integral can then be identified using a unique linear index. Equivalently, the integrals can be accessed via their shell indices and shell sizes. For the $FDSS$-batch, the corresponding formula would be $idx(f) * dSize + idx(d)$.

The ordering of table 12.5 is generated using two nested `for` loops, iterating over the functions of a shell of angular momentum l, in terms of their Cartesian powers, as shown in listing 12.5.

```
1 int idx = 0;
2 for (int z = 0; z <= l; ++z) {
3   for (int y = 0; y <= l - z; ++y, ++idx) {
4     int x = l - (x + y);
5   }
6 }
```

Listing 12.5: Generator of the shell ordering of table 12.5.

At the inner loop, for each function of the shell, the linear index `idx` and the three Cartesian powers `x`, `y` and `z` are known.

Additionally, by inspection of listing 12.5, a function can be given to calculate the linear index `idx` from the Cartesian powers. For two given fixed Cartesian powers z' and y', the added loop iterations give the linear index for this particular function. If z

in listing 12.5 is equal to z', the inner y-loop has been fully run for z's in the range of $0 \ldots (z'-1)$. For a given z, this inner loop is accessed $(l-z+1)$ times. The linear index is then just the sum of all this inner loop accesses, plus y'.

$$\begin{aligned} idx(x', y', z') &= y' + \sum_{i=0}^{z'-1} (l - i + 1) \\ &= y' + lz' + z' - \frac{z'(z'-1)}{2} \end{aligned} \tag{12.16}$$

Subtracting this formula from its increment in one of the Cartesian directions gives the correlation of the linear index of some arbitrary function and its corresponding in- or decrement, with respect to the Cartesian direction (the primes have been dropped in these formulas).

$$idx(x+1, y, z) - idx(x, y, z) = z \tag{12.17}$$
$$idx(x, y+1, z) - idx(x, y, z) = z + 1 \tag{12.18}$$
$$idx(x, y, z+1) - idx(x, y, z) = l + 2 \tag{12.19}$$

The d-function d_{xz} for example has the index 3 (see table 12.5). Incrementing this function in x-direction gives the f-function f_{x^2z}. According to equation (12.17), the index of this f-function is then calculated as $3+1=4$. For the y-increment $idx(f_{xyz}) = 3+1+1=5$ is obtained, and for the z increment $idx(f_{xz^2}) = 3+2+2=7$.

12.4.3. Generation of the Individual Batch Codes

Two approaches will be discussed here, beginning with the plain ('unrolled'), one integral per line, ansatz introduced above. Afterwards it will be shown that a substantial subset of those codes can be expressed in a rather compact form, by transforming them to a looped ('rolled') structure.

The Plain Approach

Implementing the plain approach is straight forward. For a given target batch, the integrals this batch consists of are generated using the shell iterator of listing 12.5. To each of these integrals, the corresponding transfer relation is then applied. The transfer direction is deduced from the particular integral, using the first applicable direction, in the order x before y before z. For a $PSPS$-batch, using the horizontal transfer H1, the

following is obtained.

$$
\begin{aligned}
\Theta^0_{\substack{1010\\0000\\0000}} &= \Theta^0_{\substack{2000\\0000\\0000}} + X_{AB}\Theta^0_{\substack{1000\\0000\\0000}} \\
\Theta^0_{\substack{1000\\0010\\0000}} &= \Theta^0_{\substack{1000\\1000\\0000}} + Y_{AB}\Theta^0_{\substack{1000\\0000\\0000}} \\
\Theta^0_{\substack{1000\\0000\\0010}} &= \Theta^0_{\substack{1000\\0000\\1000}} + Z_{AB}\Theta^0_{\substack{1000\\0000\\0000}} \\
\Theta^0_{\substack{0010\\1000\\0000}} &= \Theta^0_{\substack{1000\\1000\\0000}} + X_{AB}\Theta^0_{\substack{0000\\1000\\0000}} \\
\Theta^0_{\substack{0000\\1010\\0000}} &= \Theta^0_{\substack{0000\\2000\\0000}} + Y_{AB}\Theta^0_{\substack{0000\\1000\\0000}} \\
\Theta^0_{\substack{0000\\1000\\0010}} &= \Theta^0_{\substack{0000\\1000\\1000}} + Z_{AB}\Theta^0_{\substack{0000\\1000\\0000}} \\
\Theta^0_{\substack{0010\\0000\\1000}} &= \Theta^0_{\substack{1000\\0000\\1000}} + X_{AB}\Theta^0_{\substack{0000\\0000\\1000}} \\
\Theta^0_{\substack{0000\\0010\\1000}} &= \Theta^0_{\substack{0000\\1000\\1000}} + Y_{AB}\Theta^0_{\substack{0000\\0000\\1000}} \\
\Theta^0_{\substack{0000\\0000\\1010}} &= \Theta^0_{\substack{0000\\0000\\2000}} + Z_{AB}\Theta^0_{\substack{0000\\0000\\1000}}
\end{aligned}
$$

Note that the transfer directions for the integrals of this batch are unique, i.e. for each integral only transfers in one direction can be applied.

From these equations, the corresponding code is then generated by determining the indices of the participating integrals and intermediates using the ordering definitions given in section 12.4.2. For the $PSPS$-batch, the code shown in listing 12.6 is obtained.

```
1  void PSPS(
2      double * out,
3      const shellBatchData & sbd,
4      const double * dsss,
5      const double * psss) {
6    out[0] = dsss[0] + sbd.AB.x * psss[0];
7    out[1] = dsss[1] + sbd.AB.y * psss[0];
8    out[2] = dsss[3] + sbd.AB.z * psss[0];
9    out[3] = dsss[1] + sbd.AB.x * psss[1];
10   out[4] = dsss[2] + sbd.AB.y * psss[1];
11   out[5] = dsss[4] + sbd.AB.z * psss[1];
12   out[6] = dsss[3] + sbd.AB.x * psss[2];
13   out[7] = dsss[4] + sbd.AB.y * psss[2];
14   out[8] = dsss[5] + sbd.AB.z * psss[2];
15 }
```

Listing 12.6: Unrolled version of the calculation of a $PSPS$-batch.

Of special interest here is the linearity of the indices. For the target- and the $PSSS$-batch, the elements are accessed consecutively. The $DSSS$-intermediate integrals however are accessed non-consecutively. In this example, this is of no concern in terms of performance. With increasing batch-sizes, such jumps span a wider region of memory,

leading to inefficient memory access patterns.

The origin of these jumps can be found in the specific ordering of functions in a shell as defined by listing 12.5. The determination of an optimal shell- or batch ordering, with respect to index linearity is rather complex, due to the vast number of possible orderings. No optimizations with respect to the ordering have been performed.

For batches to be generated by the other recurrence relations, the same strategy as shown for the H1 relation has been used.

'Rolling' the Code by Introducing Loops

As mentioned above, the plain approach leads to rather huge code- and byte-code sizes, as each integral to be calculated results in one line of code. This size however can be drastically reduced by transforming the plain integration codes to a looped, 'rolled', structure. This has been done for the horizontal- and vertical transfer relations.

The Horizontal Transfers The uniformity of the code resulting from application of the horizontal transfers (see listing 12.6) allows for a reformulation in looped form. In the $PSPS$ example of listing 12.6, two loops of the form of listing 12.5 can be used, one for the bra1 p-shell, and one for the ket1 p-shell, respectively. The indices of the target `out` array can then easily generated from the total loop index. The indices of the $PSSS$-intermediates are determined by the bra1 loop, and the corresponding transfer directions by the ket1 loop. Unfortunately, the indices of the $DSSS$-intermediates can not be deduced in a similar simple manner. They can, however, be calculated using the index relations of equation (12.17) to equation (12.19). In this example, a particular $DSSS$ integral is generated by increasing the bra1 p-function of the target $PSPS$ integral in one of the Cartesian directions. As both the index of the p-function and the transfer direction are known for every target integral, the index of the corresponding d-function can be calculated using the appropriate index relation. The index of the p_y function in a p-shell for example is one. Incrementing this function in z-direction gives the d function d_{yz}. According to equation (12.19), the index of this function in the d-shell can then be calculated as $idx(d_{yz}) = 1 + 1 + 2 = 4$. This integral is calculated in listing 12.6 in line eleven.

Putting the above together, the $PSPS$ integration can be performed in a looped form, as is shown in listing 12.7.

```
1  void PSPS(
2          double * out,
3          const shellBatchData & sbd,
4          const double * psss,
5          const double * dsss) {
6    int uI[3];
7    int bra1Idx = 0;
8    for (int bra1z = 0; bra1z <= 1; ++bra1z) {
9      for (int bra1y = 0; bra1y <= 1 - bra1z; ++bra1y, ++bra1Idx) {
10       uI[0] = bra1Idx + bra1z;
11       uI[1] = bra1Idx + bra1z + 1;
12       uI[2] = bra1Idx + 1 + 2;
```

```
13
14        *(out++) = dsss[uI[0]] + sdb.AB.x * *(psss);
15        *(out++) = dsss[uI[1]] + sdb.AB.y * *(psss);
16        *(out++) = dsss[uI[2]] + sdb.AB.z * *(psss++);
17      }
18    }
19  }
```

Listing 12.7: Rolled version of the $PSPS$-batch.

In lines eight and nine, the loop over the bra1 p-shell can be found. The index of the according p-function, `bra1Idx`, is then used in lines ten to twelve to calculate the incremented index `uI`, for each transfer direction. This index is then used in lines 14 to 16 to access the correct $DSSS$-intermediate. The inner ket1 loop mentioned above is not introduced, but kept unrolled.

In this example, not much is gained in terms of code size compared to the plain approach of listing 12.6. For batches of higher angular momentum however, similar compact formulations exist, drastically reducing the needed code size. Listing 12.8 for example shows the code to calculate an $FFFS$ batch.

```
1  void FFFS(
2      double * out,
3      const shellBatchData & sbd,
4      const double * ffds,
5      const double * gfds) {
6
7    int uI[3];
8    int bra1Idx = 0;
9
10   for (int bra1z = 0; bra1z <= 3; ++bra1z) {
11     for (int bra1y = 0; bra1y <= 3 - bra1z; ++bra1y, ++bra1Idx) {
12
13       uI[0] = 6 * 10 * (bra1Idx + bra1z);
14       uI[1] = 6 * 10 * (bra1Idx + bra1z + 1);
15       uI[2] = 6 * 10 * (bra1Idx + 3 + 2);
16
17       for (int b2I = 0; b2I < 10; ++b2I) {
18         *(out++) = gfds[uI[0]+b2I*6+0] + sbd.AB.x[0] * *(ffds++);
19         *(out++) = gfds[uI[0]+b2I*6+1] + sbd.AB.x[0] * *(ffds++);
20         *(out++) = gfds[uI[0]+b2I*6+2] + sbd.AB.x[0] * *(ffds);
21         *(out++) = gfds[uI[1]+b2I*6+2] + sbd.AB.y[1] * *(ffds++);
22         *(out++) = gfds[uI[0]+b2I*6+3] + sbd.AB.x[0] * *(ffds++);
23         *(out++) = gfds[uI[0]+b2I*6+4] + sbd.AB.x[0] * *(ffds);
24         *(out++) = gfds[uI[1]+b2I*6+4] + sbd.AB.y[1] * *(ffds++);
25         *(out++) = gfds[uI[0]+b2I*6+5] + sbd.AB.x[0] * *(ffds);
26         *(out++) = gfds[uI[1]+b2I*6+5] + sbd.AB.y[1] * *(ffds);
27         *(out++) = gfds[uI[2]+b2I*6+5] + sbd.AB.z[2] * *(ffds++);
28       }
29     }
30   }
31 }
```

Listing 12.8: Rolled version of the $FFFS$-batch.

Almost identical to the code of the $PSPS$-batch, a plain loop for the bra2 f-shell had to be added (line 17). Additionally, the incremented g-index `uI` needs to be offset by the size of the f- and d-shells present in the intermediate $GFDS$-batch ($10 * 6$ in lines 13 to 15). Again, the inner f-loop has been kept unrolled (lines 18 to 27), due to its unnecessarily complicated looped form. Its particular structure is generated at compile time, taking care of which transfer relation to be applied to a given integral, and the explicit offsets and increments in accessing the intermediate $GFDS$- and $FFDS$-integrals.

The about 30 lines of code needed in the rolled approach correspond to a saving of code size of about a factor of 30, compared to the 1000 lines needed in the plain approach. In terms of byte-code size, for the horizontal part of calculating the $FFFS$-batch using the original code-generated approach, about 216 kByte are obtained. In the rolled approach the size of the resulting code amounts to only about 5 kByte.

Additionally, the factor which determines the size of these rolled codes is the size of the unrolled shell. The $IIIS$-batch for example has the same structure as the $FFFS$-batch, the main difference being that the unrolled i-shell corresponds to 28 functions, compared to the ten of the f-shell.

It shall be noted that the code in listing 12.8 could be generalized to all batches of the form $XYFS$, with arbitrary angular momenta X and Y. For different angular momenta at the bra1 and bra2 shells, only the loop boundaries and size offsets change. These could easily be given as arguments to a generalized `XYFS` function. Nevertheless, explicit code has been generated for each $XYFS$-batch, thus allowing for the compiler to include specific optimizations, based on the actual loop and offset sizes.

The above has been introduced for the H1 transfer, the similar H2 transfer can be implemented in an almost identical manner.

The Vertical Transfer The code reduction ansatz of rolling has been applied to the vertical transfer relation as well. The repeating pattern for this relation is the order N of the batch. The actual implementation follows the ideas introduced for the horizontal transfers, and will not be repeated. Exemplary, the code for the $DSSS$-batch is given in listing 12.9.

```
1  template <int N> void DSSS(
2      double * out,
3      const shellBatchData & sbd,
4      const double * psss,
5      const double * ssss) {
6    for (int i = 0; i <= N; ++i) {
7      double x  = sbd._2p * (ssss[(i * 1) + 0] - sbd.alpha_p *
         ssss[((i + 1) * 1) + 0]);
8
9      *(out++) = sbd.PA.x * psss[(i * 3) + 0] - sbd.alpha_p_PQ.x *
         psss[((i+1) * 3) + 0] + x;
10     *(out++) = sbd.PA.x * psss[(i * 3) + 1] - sbd.alpha_p_PQ.x *
         psss[((i+1) * 3) + 1];
11     *(out++) = sbd.PA.y * psss[(i * 3) + 1] - sbd.alpha_p_PQ.y *
         psss[((i+1) * 3) + 1] + x;
```

```
12      *(out++) = sbd.PA.x * psss[(i * 3) + 2] - sbd.alpha_p_PQ.x *
          psss[((i+1) * 3) + 2];
13      *(out++) = sbd.PA.y * psss[(i * 3) + 2] - sbd.alpha_p_PQ.y *
          psss[((i+1) * 3) + 2];
14      *(out++) = sbd.PA.z * psss[(i * 3) + 2] - sbd.alpha_p_PQ.z *
          psss[((i+1) * 3) + 2] + x;
15    }
16  }
```

Listing 12.9: Rolled version of the $DSSS$-batch.

Similar to the horizontal transfers, the loop over the decremented shell is kept unrolled (lines nine to 14). Rolling is done with respect to the maximal order N for which integrals or intermediates are needed. This number depends on the context in which the batch is requested. For a final target, only the zeroth order has to be calculated. For intermediate batches, higher values of N are needed, depending on the actual target batch. To allow for compiler optimizations, the order has been implemented as a template argument. By explicit instantiations, the compiler can then generate optimized code for that particular N-value. Again the actual code size is determined by the size of the unrolled shell.

The Electron Transfers No simple rolled expression could be found for the electron transfer relations. This is due to the fact that, in both transfers, terms appear which have to be multiplied by the Cartesian power of the bra1 shell, in the requested transfer direction. For the E_1 transfer, the term of interest is $\frac{i}{2q}\Theta_{i-1,j,00}$. Taking the $PPSS$-batch as an example, the inner loop runs over the three p-functions of the bra2 p-shell. For each of these functions, the transfer direction is unique, x for the p_x function, and y and z for the remaining, respectively. The outer loop again runs over three p-functions. For the bra1 p_x function, the $\frac{i}{2q}$ term only appears for the transfer in x-direction (the 'first' bra2 loop integral: $p_x s p_x s$). For the bra1 p_y function it appears if the transfer is in y-direction (the 'second' bra2 loop integral $p_y s p_y s$) and for the bra1 p_z function for z-transfers (the 'third' bra2 loop integral). In each bra1 loop iteration, the $\frac{i}{2q}$ term is needed at a different position in the bra2 loop. For shells of higher angular momenta, this gets even more complicated, as also different values of i are possible, depending on the actual function at the bra1 index. Nevertheless, the electron transfers could be brought to a rolled formulation, but only for the cost of a conditional `if` query, needed to decide the presence of the $\frac{i}{2q}$ term, for each integral. As this reintroduces possibly costly branching, the plain approach of section 12.4.3 has been chosen to implement the electron transfers. For these transfers, the code size is determined by the product of the shell sizes of the participating shells. To address basis sets of cc-pV6Z quality, the maximum electron transfer to be taken into account is the $OOSS$-batch, of angular momenta $l_i = l_j = 12$. For this batch about 8500 lines of code are needed in the plain approach.

12.4.4. Comparison of the Electron Transfers E_1 and E_2

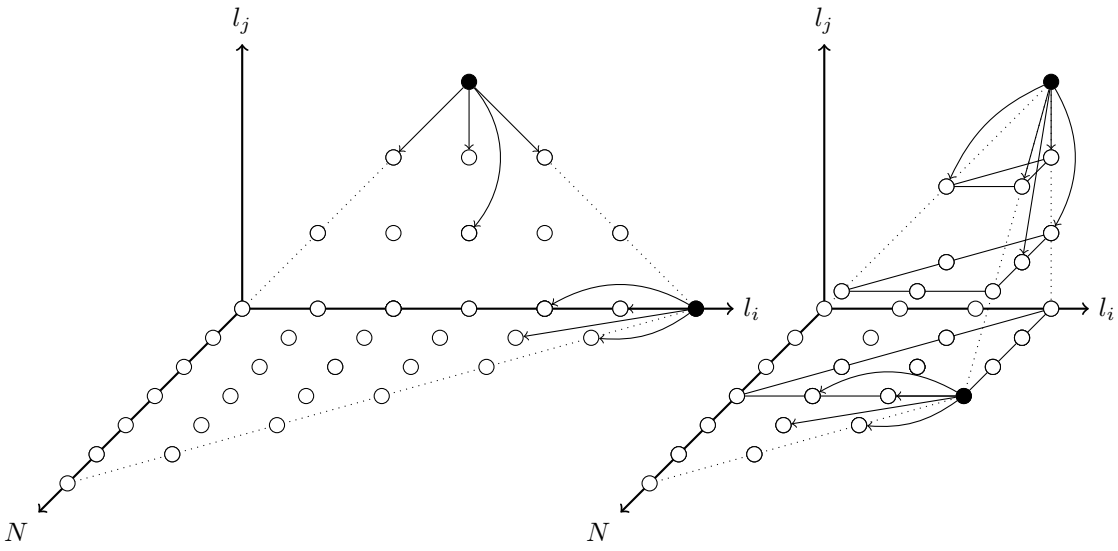
From a strict analysis of the electron recursion formulas (12.2) and (12.3), the E_1 transfer appears to be superior to the E_2 formulation. Only four terms are present in the E_1 transfer, compared to the five of the E_2 relation. Additionally, only in- and decrements of the bra1 and bra2 indices are required for the E_1 recurrence. In the E_2 transfer, additional increments of N are needed.

However, the E_2 relation lacks the numerical problems inherent in the E_1 transfer (see section 12.3). Although these numerical issues can be avoided in the E_1 transfer by swapping (interchanging) particle one and two indices, this has to be accounted for, at both the code-generation and runtime execution stages, respectively.

At the code-generation stage, code for more angular patterns has to be generated. The swapped form of the $FSFS$ batch for example is the $SFSF$ batch. As it is not of the optimal shell ordered form, it would not be included in the standard implementation. To treat swapped batches however, it has to be included. There is a non-negligible number of such angular patterns to be included, thus significantly more code has to be generated to account for swapping.

Also at runtime, swapping puts strains on the integration engine. For each batch it has to be determined if swapping has to be applied, which introduces additional logic. The criterion for swapping is based on the relative magnitudes of the exponents of a particular batch: $a + b > c + d$. In the iteration hierarchy center - angular - exponent, this criterion has to be evaluated at the innermost level of iteration. A consequence of this is an inefficient instruction cache utilization. Without swapping, all exponents (third level) connected to a certain center/angular pattern (levels one and two) can be handled by one integration code. With swapping enabled, these codes have to be sometimes replaced by their swapped versions, in an irregular manner. As the integration codes, even of the optimized code reduced versions, tend to be large for high angular momenta, frequent code changes are non-optimal.

Finally, even if more complicated (N is involved in the recurrences), the E_2 is better suited for the splitted integration implementation presented in this work. This can be shown by comparing the full recursion pathways arising from both transfers. This is done in figure 12.11, for the $FFSS$ batch.



(a) Electron Transfer E_1. (b) Electron Transfer E_2.

Figure 12.11.: Recursion pathways for the $FFSS$-batch, using the E_1 and E_2 transfers.

In this example, 24 intermediate batches are needed in the E_2 case, compared to 29 for the E_1 recurrence. Additionally, in the E_1 route intermediates of larger size (which increases with l_i and l_j) have to be calculated ($l_i = 4, 5, 6$).

The main point however is that in the context of the splitted integration ansatz, electron transfers of the same l_i and l_j, but different N can be handled by the same code. For the E_2 path, ten batches have to be calculated using the electron transfer recursion. However, only six different codes have to be used. The $FDSS^0$ and $FDSS^1$ batches, for example, can be calculated using the same function, with different arguments for N. Using the E_1 path, nine batches have to be calculated using the electron transfer. In this case, also nine codes have to be called, one for each batch, respectively.

All in all, the E_2 transfer route turns out to be superior to the E_1 ansatz. Mostly this is due to the fact that swapping is not needed for the E_2 recurrence relation.

12.5. One-Electron Integrals

The one-electron integral evaluation has been reimplemented by the author. In the original implementation [18], each integral was evaluated individually, by recursively applying the corresponding recurrence relations (equations (3.26), (3.27), (3.35), (3.36), (3.43) and (3.44)).

In the improved implementation, integrals are evaluated in terms of batches. Secondly, the expensive recursion ansatz was replaced by a generic implementation, using the approaches presented in section 12.4. Finally, overlap- and kinetic integrals were evaluated in one go, based on the equations (3.35) and (3.36).

The improvements made are shown by the wall clock time needed for the calculation of

the one-electron integrals. Table 12.6 lists the timings obtained for serotonine in the ano-rcc basis and the resulting speedups, for existing- and reimplementation, respectively.

	Wallclock time		Speedup
	Previous Version	Reimplementation	
Overlap	4.56	0.19	88
Kinetic	12.01		
Nuclear Potential	474.32	4.97	95
Sum	490.89	5.16	95

Table 12.6.: Wall clock times (in seconds) for the generation of the one-electron integrals and corresponding speedups.

The generation of the one-electron integrals has been significantly improved. In this example, the overall calculation has been sped up by a factor of about 100. In general, the speedups depend on basis set and molecule size, as is shown in table 12.7, in which the total (all three one-electron integrals) speedup of the reimplementation versus the previous implementation is summarized. Speedups of up to about 2000 are observed.

Basis	C_4H_{10}	Serotonine	$C_{36}H_{53}N_7O_9$
cc-pVDZ	3	5	4
roos-aug-dz-ano	6	8	6
cc-pVTZ	11	16	14
roos-aug-tz-ano	18	24	21
cc-pVQZ	52	72	68
ano-rcc	64	95	91
cc-pV5Z	248	361	343
cc-pV6Z	1297	1960	-

Table 12.7.: Speedups of the reimplementation of the one-electron integral calculation routines versus the previous implementation.

13. Density Contraction

Conceptually, the implementation of the density contraction follows the pathway taken for the implementation of the Obara-Saika recurrence relations. Starting point has been an already existing, generic implementation by Michael Hanrath [18]. Basically, this ansatz had similar shortcomings as were present in the generic integration code. In this work, they have been addressed by a code-generated approach. A first, plain implementation lead again to impractical code sizes. In a second implementation, practical code sizes have been achieved by rolling the density contraction codes.

13.1. Existing Implementation

The previous implementation was based on treating non-redundant integrals. In an approach similar to the one presented in section 5.1, integral contributions to the Fock matrix have been determined, based on their symmetry class. This leads to the generic implementation shown in listing 13.1.

```
1  // matrix J; // coulomb part of G
2  // matrix K; // exchange part of G
3
4  // calculate the integral batch
5
6  // loop over the non-redundant integrals of that batch
7  for (typename Iterator::SubIter44 i4(iPrim); i4.valid(); ++i4) {
8    // int a,b,c,d;   // indices of current integral
9
10   // double intVal; // value of integral <ab|cd>
11   double intValHalf = 0.5 * intVal;
12
13   int symClass = (b == d) |
14                  ((a == c) << 1) |
15                  ((a == b && c == d) << 2);
16
17   switch(symClass) {
18     case 3: // <ab|ab>
19       J(a,c) +=   intVal*D(d,b);
20       J(b,d) +=   intVal*D(c,a);
21
22       K(a,d) -=   intValHalf*D(c,b);
23       K(b,c) -=   intValHalf*D(d,a);
24       break;
25
26     case 7: // <aa|aa>
```

```
27          J(a,c) +=    intVal*D(d,b);
28
29          K(a,d) -=    intValHalf*D(c,b);
30          break;
31
32          // similar for the remaining cases: 0 1 2 and 4
33
34        default: throw;
35      }
36  }
```

Listing 13.1: Generic contraction

After each batch is calculated, all of its non-redundant integrals are accessed sequentially (loop in line seven). After retrieving its value and indices (lines eight and ten, no actual code shown), the symmetry class `symClass` of the particular integral is calculated (line thirteen). Based on this symmetry class, the integral is then contracted with the density (lines 17 to 35). Explicit code is shown only for integrals of type $\langle ab|ab\rangle$ and $\langle aa|aa\rangle$. The code for the remaining classes is similar, although different for each class, respectively. With $\mathbf{G} = \mathbf{J} + \mathbf{K}$, a distinction of Coulomb- and exchange-contributions has been introduced.

As mentioned, several shortcomings exist in this ansatz. On one hand, a lot of possibly inefficient branching is introduced by the `switch` / `case` conditional of line 17. Additionally, for a particular batch- (or angular-) pattern, the logic needed to contract that batch is independent of the actual integral- and density-matrix values. Nevertheless, it is regenerated during runtime for each appearing batch.

13.2. Code-Generated Implementation

Identically to the integration, these shortcomings have been addressed by a code-generated ansatz. Again, the logic of the contraction is analyzed at compile time, and optimized code is automatically generated for each requested angular pattern. These optimized code fragments are then called at runtime.

13.2.1. Plain Approach

Starting point of the code-generated density contraction has been a brute force plain approach. In this ansatz, the sole angular pattern is not sufficient to treat the contraction of the density with the integrals. This is due to different symmetry properties of batches with identical angular patterns. For the angular pattern $DPSP$ for example, two different contraction routes appear. One is connected to batches with different p-shells. In these DP_iSP_j-batches, only integrals with four different functions appear. In batches with equal p-shells DP_iSP_i, on the other hand, integrals with four different functions are mixed with integrals in which the two p-functions are identical. As integrals with equal p-functions contract differently than integrals with four different functions (see section 5.1.2), two different codes are needed, one to contract the DP_iSP_j batches, and

one for the DP_iSP_i ones, respectively. The actual number of different routes depends on the specific angular pattern, and the potential equalities or inequalities. For batches with only one angular momentum, i.e. $PPPP$, 15 different contraction routes have to be taken into account.

In the automatic code-generation, the full set of integrals is generated for each batch pattern, without respect to symmetry. Their contributions to the Fock matrix are then computed, and related to the actual integral sequence of the batch. The order of integrals is given by the implementation of the Obara-Saika scheme (see section 12.4.2). The obtained information is then used to generate the actual code. Exemplary, listing 13.2 shows the code for a $PS_iS_jS_i$-batch.

The extension `_0010` identifies shell equalities. As p- and s-shells are different per definition, no distinction has been made between the first and remaining shells. The equalities for the three s-shells are then given by `010`, indicating the equality of the first and third s-shell, and the difference of the second.

```
1  void psss_0010(
2      const double * in,
3      Matrix<double> & G,
4      const Matrix<double> & D,
5      const int * BI) {
6    double d[9];
7    double g[9];
8
9    d[0] = D(BI[1]+0,BI[2]+0);
10   d[1] = D(BI[1]+0,BI[1]+0);
11   d[2] = D(BI[0]+0,BI[2]+0);
12   d[3] = D(BI[0]+0,BI[1]+0);
13   d[4] = D(BI[0]+1,BI[2]+0);
14   d[5] = D(BI[0]+1,BI[1]+0);
15   d[6] = D(BI[0]+2,BI[2]+0);
16   d[7] = D(BI[0]+2,BI[1]+0);
17
18   g[0]  = d[0] * in[0] * -0.5;
19   g[1]  = d[1] * in[0] * 1;
20   g[2]  = d[2] * in[0] * 2;
21   g[3]  = d[3] * in[0] * -0.5;
22
23   g[4]  = d[0] * in[1] * -0.5;
24   g[5]  = d[1] * in[1] * 1;
25   g[2] += d[4] * in[1] * 2;
26   g[3] += d[5] * in[1] * -0.5;
27
28   g[6]  = d[0] * in[2] * -0.5;
29   g[7]  = d[1] * in[2] * 1;
30   g[2] += d[6] * in[2] * 2;
31   g[3] += d[7] * in[2] * -0.5;
32
33   G(BI[0]+0,BI[1]+0) += g[0];
34   G(BI[0]+0,BI[2]+0) += g[1];
35   G(BI[1]+0,BI[1]+0) += g[2];
```

```
36    G(BI[1]+0,BI[2]+0) += g[3];
37    G(BI[0]+1,BI[1]+0) += g[4];
38    G(BI[0]+1,BI[2]+0) += g[5];
39    G(BI[0]+2,BI[1]+0) += g[6];
40    G(BI[0]+2,BI[2]+0) += g[7];
41  }
```

Listing 13.2: Code-generated contraction of a $PS_iS_jS_i$ batch.

Every contraction is formulated in a three step ansatz. Prior to the contraction, the density elements needed for the particular batch are copied to a local array (`d`, lines nine to 16). In the second step, these local elements are then contracted with the integrals (lines 18 to 31), where the prefactors have been determined at the code-generation stage. The target of this contraction is again not the global **G**, but the local array `g`. This local array then gets written back to the actual **G** matrix in the final contraction step (lines 33 to 40).

This local formulation has been chosen to optimize memory accesses. Despite the overhead introduced by first reading from the density and writing back to **G**, the contraction can in this way be performed on compact memory structures. In almost all cases the matrix elements needed for the contraction are scattered over wide regions of memory, making their access inefficient. The actual indices to be used are generated by the code-generation engine, and can be optimized for access linearity.

Finally, the actual **D** and **G** matrix elements can be identified by the base-index `BI`. It points to a four dimensional array, holding the indices of the first functions of each shell of the actual batch. As the functions of a shell appear contiguous in the basis, the indices of subsequent functions in a shell can be generated implicitly from this index. For example the index of the third p-function of the second p-shell in a $PPSS$ batch would be `BI[1]+2`. Thus a general way to access the particular matrix elements needed in the contraction code is available.

Results

To get a reliable comparison of the efficiencies of the generic versus the plain code-generated approach, production runs on real life examples have been performed. With the exception of the contraction approach, the same codes for the rest of the calculation have been used. Any change of performance relevant quantities can thus be directly correlated with the corresponding contraction ansatz. In table 13.1, results obtained for the first SCF iteration of serotonine in a cc-pVTZ basis are shown. Prescreening has been turned on, using a threshold of 10^{-11} and the density used was the initial density obtained with TURBOMOLE.

	Generic	Code-generated	Factor
Time (seconds)	825.31	468.16	1.8
Cycles	2,082,878,883,368	1,180,716,426,905	1.8
Branches	170,724,018,453	41,112,915,867	4.2
Branch misses	0.8%	1.7%	-
L1 iCache loads	1,261,368,941,506	596,193,676,312	2.1
L1 iCache load-misses	3.7%	26.8%	-
L1 dCache loads	1,279,215,100,424	506,590,272,422	2.5
L1 dCache load-missses	1.0%	2.0%	-
L1 dCache stores	415,486,002,081	230,501,994,069	1.8
L1 dCache store-misses	1.0%	1.6%	-

Table 13.1.: Performance comparison of the generic versus the code-generated approach for serotonine in a cc-pVTZ basis.

Similar to changing the integration kernel from generic to code-generated, a significantly increased performance can be observed for the code-generated contraction. By changing the contraction kernel to the code-generated version, the overall wall clock time is reduced by almost a factor of two. Attributing to the branchless generated code, the number of branches to evaluate could be reduced by a factor of four. The various cache accesses have also been reduced by factors of about two.

However, and again comparable to the code-generated integration, the number of missed instruction cache loads is drastically increased to over 25 percent. This is again due to the plain approach chosen for the contraction, leading to huge code sizes, both in terms of lines of code and object file sizes. For each integral, between one ($\langle aa|aa\rangle$ integrals) and six ($\langle ab|cd\rangle$ integrals) lines of code are needed. For an $FFFF$-batch with four different f-shells, this amounts to 60000 lines of code. Additionally, the code for reading to the local copy of $\mathbf{D}$ and writing back to the global $\mathbf{G}$ has to be taken into account. For the aforementioned $FFFF$-batch, another 1200 lines of code are needed. Again similar to the integration codes, this ansatz is only practicable for basis sets which use up to f functions.

13.2.2. Rolling the Code

Using the ideas presented in section 5.2, the density contraction can be formulated in a rather compact form. As has been shown, each integral of a given, non-redundant batch

gives the following contribution to the Fock matrix.

$$\begin{aligned}
G_{pq} &+= f_s \cdot 2.0 \cdot D_{rs} \cdot \langle pr|qs\rangle \\
G_{rs} &+= f_s \cdot 2.0 \cdot D_{pq} \cdot \langle pr|qs\rangle \\
G_{ps} &-= f_s \cdot 0.5 \cdot D_{rq} \cdot \langle pr|qs\rangle \\
G_{pr} &-= f_s \cdot 0.5 \cdot D_{qs} \cdot \langle pr|qs\rangle \\
G_{rq} &-= f_s \cdot 0.5 \cdot D_{ps} \cdot \langle pr|qs\rangle \\
G_{qs} &-= f_s \cdot 0.5 \cdot D_{pr} \cdot \langle pr|qs\rangle
\end{aligned} \tag{13.1}$$

Here f_s is the symmetry correction of equation (5.10), depending on the actual batch.

Using these equations, a simple loop can be given to contract all integrals of some batch. This loop is the basis for the rolled contraction ansatz and shown in listing 13.3.

```
1  // batch: <b1 b2|k1 k2>
2  // density:   matrix D
3  // G:         matrix G
4  // integrals: array I
5  int c = 0;
6  for (int b1 = 0; b1 < b1_size; ++b1)
7   for (int b2 = 0; b2 < b2_size; ++b2)
8    for (int k1 = 0; k1 < k1_size; ++k1)
9     for (int k2 = 0; k2 < k2_size; ++k2, ++c) {
10      G(BI[0]+b1,BI[2]+k1) += 2.0 * f * D(BI[1]+b2,BI[3]+k2) * I[c];
11      G(BI[1]+b2,BI[3]+k2) += 2.0 * f * D(BI[0]+b1,BI[2]+k1) * I[c];
12      G(BI[0]+b1,BI[3]+k2) -= 0.5 * f * D(BI[1]+b2,BI[2]+k1) * I[c];
13      G(BI[0]+b1,BI[1]+b2) -= 0.5 * f * D(BI[2]+k1,BI[3]+k2) * I[c];
14      G(BI[2]+k1,BI[3]+k2) -= 0.5 * f * D(BI[0]+b1,BI[1]+b2) * I[c];
15      G(BI[1]+b2,BI[2]+k1) -= 0.5 * f * D(BI[0]+b1,BI[3]+k2) * I[c];
16    }
```

Listing 13.3: Generic rolled contraction loop, in terms of the global `G` and `D` matrices.

The integrals, stored in the array `I`, are accessed linearly (using the counter `c`). Each integral is then contracted with the density according to equation (13.1) (lines nine to 14). The particular **D** and **G** elements are accessed via the base-index `BI`, where the actual offset is calculated from the loop-counters of lines five to eight. These loops run over the sizes of the shells associated with the actual batch, reproducing the order in which the integrals are stored by the Obara-Saika implementation.

It shall be noted here that this ansatz works for integral batches in any basis, and any ordering of integrals. For a different integral ordering, the loops in lines five to eight have to be permuted to represent the given order (implying that there is some structure in the integral ordering). The basis of the integrals is addressed by the shell sizes in the four loops. For batches in the primitive basis, $n_l^p = \frac{(l+1)(l+2)}{2}$ has to be used, and $n_l^{sh} = 2l + 1$ for spherical-harmonic bases, respectively.

All six formulas of equation (13.1) can be formulated as matrix vector multiplications

by interpreting D and G as vectors, and the integrals as a matrix.

$$\begin{aligned}
\vec{G}_{(pr)} &= -f_s \cdot 0.5 \cdot & I_{pr,qs}\vec{D}_{(qs)} \\
\vec{G}_{(qs)} &= -f_s \cdot 0.5 \cdot & \vec{D}^T_{(pr)} \cdot I_{pr,qs} \\
\vec{G}_{(pq)} &= +f_s \cdot 2.0 \cdot & I_{pq,rs}\vec{D}_{(rs)} \\
\vec{G}_{(rs)} &= +f_s \cdot 2.0 \cdot & \vec{D}^T_{(pq)} \cdot I_{pq,rs} \\
\vec{G}_{(ps)} &= -f_s \cdot 0.5 \cdot & I_{ps,rq}\vec{D}_{(rq)} \\
\vec{G}_{(rq)} &= -f_s \cdot 0.5 \cdot & \vec{D}^T_{(ps)} \cdot I_{ps,rq}
\end{aligned} \tag{13.2}$$

Although this would allow for the use of effective LAPACK/BLAS routines, it has not been used in the implementation. To use this formulation, the integrals have to be resorted two times, to give the needed matrix structures. Furthermore, for each of the six equations, one LAPACK/BLAS call is needed, which effectively iterates over all integrals each time. In the formulation of listing 13.3 however, the integrals are iterated over just once. Finally, a huge amount of the **D** and **G** vectors are small in the batch formulation (i.e. three elements for a ps vector). In such cases the performance gain of the LAPACK/BLAS routines is negligible. Thus a further optimized version of listing 13.3 has been used in the final density contraction implementation.

Shell Pairs

To optimize memory usage, the same three-fold contraction ansatz as in the plain implementation has been chosen for the rolled ansatz. Copying from **D** and writing back to **G** can be greatly simplified by having a look at the particular indices involved in the contraction of equation (13.1). To contract one integral, six **D**-elements are needed, and written to six **G**-elements. The indices of these elements are just the six possible two-combinations obtainable from the four indices p, r, q and s, in both cases. The connection of the corresponding **D** and **G** elements then is mutually exclusive. Each **D**-element is contracted into the **G** element having the two indices not used in that **D** element. D_{pq} is contracted to G_{rs} and so on. Index combinations of the same (pq and rs) and mixed particles give the factors 2.0 and -0.5, respectively.

In the extension to batches, instead of six matrix elements, six sets of matrix elements are needed, respectively. These sets arise from the two combinations of the shells of the particular batch. For some batch $PRQS$, the shell pairs PR, PQ, PS, RQ, RS and QS arise. From each of these shell pairs, the associated set of elements can be obtained by the Cartesian product of the functions in the particular shells. Note that such sets of elements have already been introduced in the context of prescreening (section 11.1).

Additionally it shall be noted that the constants in (13.1), i.e. $f_s \cdot 2.0$ and $-f_s \cdot 0.5$, can be factored out of the actual contraction. This can be done in two different ways. On one hand the multiplication can be performed when reading from **D** to the local array. On the other hand it can be performed when writing back to **G**. In both ways the flop count is reduced, as in almost all cases the number of **D**, **G** elements is smaller than the number of integrals. In the $FFFF$-batch for example, 10000 integrals have to be

contracted. Each integral has to be multiplied by the two constants ($2f_s$ and $-0.5f_s$), giving 20000 prefactor multiplications. However, each FF-shell pair holds 100 elements. With six pairs, 600 elements have to be read from **D** and 600 written back to **G**. Doing the prefactor multiplication in one of these steps replaces the 20000 multiplications by 600.

The above can then be used to simplify the three parts copying, contracting and writing back. Instead of one local array for `d` and `g`, two times six individual ones are introduced, one for each shell pair of **D** and **G**, respectively. Listing 13.4 shows this for the copy of the density elements arising from the bra1 and ket1 shell pair.

```
1 double d_b1k1[b1_size * k1_size];
2 double f2 = f * 2.0;
3 for (int i = 0; i < b1_size; ++i)
4   for (int j = 0; j < k1_size; ++j)
5     d_b1k1[i * k1_size + j] = f2 * D(BI[0]+i, BI[2]+j);
```

Listing 13.4: Copying of the density elements associated with the bra1 ket1 shell pair to the corresponding local array.

Here the rectangular block of **D** is copied to the linear array `d_b1k1`, saving the block-matrix elements in row-major order. The prefactor `f2` is also already multiplied into the local representation. The remaining local `d` arrays are filled similarly, using the appropriate shell sizes, prefactors and base-indices. Writing the six local `g` arrays back to **G** is conceptually identical.

Using these local arrays, the contraction loop of listing 13.3 is replaced by the one shown in listing 13.5.

```
1  int c = 0;
2  for (int b1 = 0; b1 < b1_size; ++b1)
3   for (int b2 = 0; b2 < b2_size; ++b2)
4    for (int k1 = 0; k1 < k1_size; ++k1)
5     for (int k2 = 0; k2 < k2_size; ++k2, ++c) {
6       g_b1k1[b1 * k1_size + k1] += d_b2k2[b2 * k2_size + k2] * I[c];
7       g_b2k2[b2 * k2_size + k2] += d_b1k1[b1 * k1_size + k1] * I[c];
8       g_b1k2[b1 * k2_size + k2] += d_b2k1[b2 * k1_size + k1] * I[c];
9       g_b1b2[b1 * b2_size + b2] += d_k1k2[k1 * k2_size + k2] * I[c];
10      g_k1k2[k1 * k2_size + k2] += d_b1b2[b1 * b2_size + b2] * I[c];
11      g_b2k1[b2 * k1_size + k1] += d_b1k2[b1 * k2_size + k2] * I[c];
12    }
```

Listing 13.5: Generic rolled contraction loop, in terms of the local matrix representations, with factored out prefactors.

Clearly, the access to the local arrays is as linear as possible.

Block-Linearization

The **D** and **G** elements needed to contract a particular batch appear as individual blocks in the full matrix. Due to the way matrices are stored internally, accessing one of these blocks is far from memory efficient. Although being two-index quantities, matrices are stored linearly. In the case of C++, elements are stored in row major order. The

consequence is that two neighboring row elements are in adjacent memory locations. Two neighboring column elements however are separated in memory by the dimension of the matrix. In terms of reading blocks from a linear matrix representation, this results in non-contiguous memory accesses, which leads to performance loss. Pictorially, this is shown in figure 13.1, showing the matrix structure and shell pair block layout for a simple system with one s- and p-shell. The numbers in the matrix correspond to the linear position in memory.

	s	p_x	p_y	p_z
s	0	1	2	3
p_x	4	5	6	7
p_y	8	9	10	11
p_z	12	13	14	15

Figure 13.1.: Row major order matrix indices, and element correspondence to shell pair blocks.

Obviously, to read the ps- and pp blocks, non-contiguous memory accesses are needed. This problem has been addressed by block linearization of the corresponding matrices ($\mathbf{D}$ and $\mathbf{G}$). Effectively, the matrices are resorted in memory with respect to linearization of shell pair blocks. For the example shown in figure 13.1, this resorting gives the memory layout shown in figure 13.2.

	s	p_x	p_y	p_z
s	0	1	2	3
p_x	4	7	8	9
p_y	5	10	11	12
p_z	6	13	14	15

Figure 13.2.: Block linearized matrix storage of shell pair block matrices.

In the block-linearized form, each shell pair block appears contiguous in memory.

In each iteration, the current density is block linearized. This linearized representation is then contracted with the integrals to give the two-electron part $\mathbf{G}$ of the Fock matrix, also in block linearized form. After all integrals are contracted, the linearized $\mathbf{G}$ is then resorted to give the canonical matrix structure. The cost for this resorting step is negligible, compared to the generation of the Fock matrix. For serotonine in a cc-pVTZ basis with 815 primitive basis functions, resorting one 815 x 815 matrix takes about 0.007 seconds. To generate the Fock matrix for this system, 162 seconds are needed, in the first iteration. This number has been generated using the final, fastest, version of the SCF code. Accessing elements in these linearized matrices is however slightly more costly compared to accessing the standard matrices. As only whole shell pair blocks are needed in the implementation, individual element access has not been explicitly included. The access to the shell pair blocks has been implemented using the

`MapperWithMatrix` class presented in the prescreening implementation of section 11.1.1, on batch level. The underlying matrix of this class now holds the indices of the first elements of the shell pair blocks. Querying the mapper with arbitrary basis-function indices then gives the starting index of the according shell pair block in the linearized matrix. For each batch, six of these indices have to be queried. They can be used to identify both **D** and **G** elements.

Having the external matrices in linearized form, the contraction of listing 13.5 can be expressed in terms of those arrays, without the need to copy to the local representations. However this is not as efficient as doing the copy explicitly. The reason is, on one hand, that by copying the elements, they are already read to the cache, thus reducing costly cache misses at the later contraction stage. On the other hand, by copying, the shell pair blocks become contiguous in memory, which further optimizes memory usage.

Final Implementation

In principle, the ideas presented above can be used to implement a generic version of the rolled contraction ansatz. The code structure is the same for each batch, the only input are the memory addresses of the needed shell pair blocks and the shell sizes of the batch. Nevertheless an explicit implementation for each angular pattern has been chosen. In this way, the actual loop lengths are hardcoded, thus allowing the compiler to perform efficient optimizations. The actual code shall not be shown explicitly. To copy from **D** and write back to **G**, code similar to the one shown in listing 13.4 is used, with the modification of reading from the linearized version of **D**. For the contraction, the code of listing 13.5 is used.

Contrary to the plain implementation, only one function per angular pattern has to be implemented. In the rolled ansatz, the symmetry distinctions necessary are taken care of by the factor f_s.

Results

Reliable results for the rolled contraction ansatz have again been obtained by performing production runs, with changed underlying contraction kernels. For the rest of the calculation the same code has been used (see section 13.2.1). Table 13.2 repeats the results obtained for serotonine in the cc-pVTZ basis already shown in table 13.1, in addition to the results obtained using the rolled contraction ansatz.

	Generic	Code-generated	
		Plain	Rolled
Time (seconds)	825.31	468.16	319.26
Cycles	2,082,878,883,368	1,180,716,426,905	805,490,143,574
Branches	170,724,018,453	41,112,915,867	46,072,615,242
Branch misses	0.8%	1.7%	1.7%
L1 iCache loads	1,261,368,941,506	596,193,676,312	507,203,828,123
L1 iCache load-misses	3.7%	26.8%	9.5%
L1 dCache loads	1,279,215,100,424	506,590,272,422	468,880,988,703
L1 dCache load-missses	1.0%	2.0%	1.7%
L1 dCache stores	415,486,002,081	230,501,994,069	243,041,661,625
L1 dCache store-misses	1.0%	1.6%	1.2%

Table 13.2.: Performance comparison of the generic versus both code-generated approaches for serotonine in a cc-pVTZ basis.

A significant speedup of about 1.5, compared to the plain approach, has been achieved. One of the reasons is the optimized instruction cache handling. Less loads are needed and, more importantly, the iCache misses were reduced from about 25 to 10 percent.

It shall be noted that the above speedup of 1.5 corresponds to the full Fock matrix generation, of which the density contraction is just one step among others. In table 13.3, the attempt has been made to isolate the time needed to perform the sole contraction. This has been done by running two calculations for each contraction method, respectively. The first of these calculations is just the full Fock matrix generation. For the second calculation, the density contraction has been turned off. For serotonine in the cc-pVTZ basis, this contractionless calculation took about 221 seconds (the same for all contraction types, as these codes were kept identical). The difference of these two timings then gives a rough estimate for the time needed to perform the contraction. It has to be noted that the contraction timings obtained this way are not to be taken absolute. This is due to the fact that by omitting the contraction, less strains are put to the various system resources (especially the caches), and therefore the rest of the calculation (integration etc.) is executed in a different environment.

The results obtained are shown in table 13.3, giving the full timings, and the estimates for the contraction step. Speedups are reported with respect to the generic approach. For the contraction estimates, the percentage of the full calculation is reported.

Type	Full	Speedup	Contraction	Speedup
Generic	825.31	1	604.73(73.27%)	1
Code-generated, plain	468.16	1.8	247.58(52.88%)	2.4
Code-generated, rolled	319.26	2.6	98.68(30.91%)	6.1

Table 13.3.: Timings, in seconds, and speedups for the full Fock matrix generation and the estimated contraction steps.

Given the significant speedups of 1.8 and 2.6 of the code-generated contractions in the full calculations, the actual contraction has been speed up by a factor of 2.4 in the plain approach. The rolled approach executes about six times faster than the generic code, bringing the amount of time spent for the contraction down from two thirds of the whole Fock matrix generation to one third.

This is not an isolated effect, as is shown in table 13.4. Calculations as for serotonine in the cc-pVTZ basis have been performed for a variety of molecules and basis sets. Shown are the timings for all three contraction approaches, and the speedups of the code-generated with respect to the generic ansatz.

Basis	Molecule	Fock matrix generation (seconds)			Speedups to generic	
		Generic	Code-generated			
			Plain	Rolled	Plain	Rolled
Def2-SVP	Alanine	6.56	3.49	3.15	1.90	2.08
	Serotonine	60.75	32.55	29.39	1.87	2.07
cc-pVDZ	Alanine	8.02	4.30	3.91	1.87	2.05
	Serotonine	77.57	42.18	38.30	1.84	2.03
roos-aug-dz-ano	Alanine	332.90	152.27	124.11	2.19	2.68
	Serotonine	4085.08	1915.08	1522.55	2.13	2.68
Def2-TZVP	C_4H_{10}	30.45	16.57	13.08	1.84	2.33
	Alanine	64.77	35.04	25.63	1.85	2.53
	Serotonine	705.17	393.59	284.37	1.79	2.48
cc-pVTZ	C_4H_{10}	68.13	37.48	26.79	1.82	2.54
	Alanine	92.92	51.76	35.59	1.80	2.61
	Serotonine	825.31	468.16	319.26	1.76	2.59
roos-aug-tz-ano	C_4H_{10}	1583.10	857.37	545.39	1.85	2.90
	Alanine	2156.70	1201.91	743.54	1.79	2.90

Table 13.4.: Timings, in seconds, and speedups for the full Fock matrix generation for various molecules and basis sets.

For the plain approach, speedups of about 1.8 are observed, with the exception of the roos-aug-dz-basis. The speedups for the rolled ansatz increase with increasing basis set size, to almost a factor of three for the roos-aug-tz-ano basis.

14. Fock Matrix Generation

Fock matrix assembly is realized by putting the parts introduced above together. All results presented in the following are based on the same algorithmic structure. Listing 14.1 shows the key steps involved in the Fock matrix generation.

```
1  basisTransformation.radAng_2_prim(D_radAng, D_prim);
2  densityEstimate.update(D_prim);
3  blockLinearizer.linearize(D_prim, D_prim_lin);
4
5  for () {      // first iteration hierarchy
6    for () {    // second iteration hierarchy
7      for () { // third iteration hierarchy
8
9        // calculate integral batch
10       // contract integral batch with density
11
12     }
13   }
14 }
15
16 blockLinearizer.deLinearize(G_prim_lin, G_prim);
17 basisTransformation.prim_2_radAng(G_prim, G_radAng);
```

Listing 14.1: General Fock matrix assembly algorithm.

Explicit integral transformation is avoided in the ansatz presented here. This is realized by transforming density and Fock matrices from and to the respective bases (section 6.2.2 and section 6.2.3). These transformations are the introductory and closing steps of the Fock matrix generation. In line one the radial angular transformed density (`D_radAgn`) is transformed to the primitive basis (`D_prim`). In the next steps, the density estimate is updated (line two) and the density matrix is block linearized (line three). The actual integral calculation and density contraction steps are then performed (lines five to 14), in which the two-electron part **G** of the Fock matrix is obtained in the primitive basis, in block linearized form. In lines 16 and 17 this matrix is then delinearized and transformed back to the radial angular transformed basis.

The basis transformation, matrix (de)linearization and estimate update steps (lines one, two, three, 16 and 17) are not relevant to the performance. For the system with 17437 basis functions discussed in section 16, 22 seconds are needed for their execution.

For the actual Fock matrix assembly part of listing 14.1, lines five to 17, several strategies have been realized, and analyzed. Different iteration schemes (see section 10) have been combined with the available integration schemes (see section 12.2.2 and section 12.4). Not all results will be reported here, only the key steps leading to the finally used implementation.

14.1. Results

Performance analyses will be shown for the Fock matrix generation, lines five to 17, of listing 14.1. All results presented here have been obtained in an unified way. The input densities used are the initial densities obtained from TURBOMOLE using the extended Hückel guess, as described in section 15.2. Prescreening has been used, neglecting integral contributions smaller than 10^{-11}. In doing so, the presented results correspond to the first iteration of an actual SCF calculation. All calculations have been performed on one of the Xeon(R) E5520 hosts introduced in section 8.3.1. They have been run sequentially, assuring no interfering jobs. In all comparisons presented, only the codes for the part of interest were changed, using the same codebase for the remaining steps. In doing so, fair comparisons have been realized. For the density contraction step the approach presented in section 13.2.2 has been used for all calculations.

14.1.1. Comparison of the Integration Codes

In a first step, the quality of the developed integration codes shall be analyzed by comparing their performance to the code-generated ansatz which was already implemented in the QOL [18]. Table 14.1 compares the results obtained with the rolled, splitted ansatz of section 12.4 with the previous implementation of section 12.2.2. For the previous implementation the electron transfer E_1 is used, for the splitted, rolled ansatz the E_2 transfer as well. The results shown were obtained in the center-angular-zeta iteration framework of section 10.2.2. In table 14.1, the results obtained for alanine in correlation consistent basis sets of increasing quality are shown.

	Implementation		
	Previous	Splitted, rolled	
	E_1	E_1	E_2
cc-pVDZ	3.796	3.886	3.277
cc-pVTZ	36.919	35.503	32.493
cc-pVQZ	-	417.876	319.676

Table 14.1.: Performance comparisons for alanine in selected integration implementations, in seconds.

Almost identical results are obtained for the E_1 transfer, in the double- and triple-zeta basis sets. However, due to the huge code sizes of the previous implementation, results for quintuple-zeta quality could only be obtained with the splitted, rolled ansatz. The reason for the equal performances of the E_1 realizations is a performance shift with increasing batch size. For small batch sizes, the previous implementation is highly efficient, whereas the splitted ansatz gives poorer results due to function-call overhead. For large batch sizes the splitted, rolled ansatz is more efficient, lacking the huge code sizes present in the previous implementation. A combination of the two approaches, based on the batch sizes, leads indeed to improved performance.

The E_2 transfer, realized for the rolled, splitted ansatz shows increased performance for all basis set qualities. This is mainly due to the missing swap logic which is needed in the E_1 transfer to assure numerical stability (see section 12.3). Secondly the E_2 transfer is better suited for the splitted, rolled ansatz, as has been shown in section 12.4.4.

In the following, integration will be performed using the E_2 transfer in the rolled, splitted implementation.

14.1.2. Comparison of the Iteration Hierarchies

As discussed briefly in section 11.2.1 and section 11.3, the influence of the iteration structure shall be shown here explicitly. Included in the comparison are iteration schemes of the center-angular-zeta hierarchy, in the physicists' (see section 10.2.2) and chemists' (section 10.2.3) notations, respectively. Integration method was the rolled, splitted ansatz, using the E_2 transfer. The results for molecules of varying size in basis sets of differing quality are summarized in table 14.2.

		Notation		
		Physicists'	Chemists'	Speedup
Alanine	cc-pVQZ	329.92	322.45	1.02
Serotonine	cc-pVDZ	36.74	27.69	1.33
Serotonine	cc-pVTZ	267.53	239.37	1.12
Serotonine	cc-pVQZ	2726.44	2624.83	1.04
Serotonine	cc-pV5Z	27 393.41	26 939.78	1.02
Serotonine	roos-aug-dz-ano	1424.66	1253.70	1.14
Serotonine	roos-aug-tz-ano	6843.99	6588.91	1.04
$C_{36}H_{53}N_7O_9$	cc-pVDZ	1645.41	868.87	1.89
$C_{36}H_{53}N_7O_9$	Def2-SVP	1303.05	748.28	1.74

Table 14.2.: Performance comparison of the center-angular-zeta iteration hierarchy, in physicist' and chemists' notation (in seconds).

For all molecules and basis sets under investigation, the chemists' notation ansatz proved to be superior. Although only marginal for molecules of moderate size, speedups of almost 2 are observed for large molecules.

14.1.3. Streaming SIMD Extensions

All integration types (except the generic) and the rolled density contraction of section 13.2.2 have been adapted to explicitly use streaming SIMD extensions. Shortly put, these extensions allow the simultaneous multiplication of two double precision numbers with one instruction. These new instructions are partly included by the compiler itself, at least for simple constructs. For more complicated codes, manual modifications are necessary. In the QOL [18], this problem is addressed in a very elegant manner, by

providing a type for these extensions. Its usage is depicted in listing 14.2, in comparison to the use of the standard double precision type in listing 14.3.

```
1  int main() {
2    SIMD_Class<SIMD_Type<double, SIMD_128Bit> > a(0.1, 0.3);
3    SIMD_Class<SIMD_Type<double, SIMD_128Bit> > b(0.2, 0.4);
4
5    cout << a*b << endl;
6  }
7  // assembly
8  // ..
9  // mulpd   %xmm1, %xmm0
10 // ..
```

Listing 14.2: Multiplication using the `SIMD_class` type.

```
1  int main() {
2    double a = 0.1;
3    double b = 0.2;
4
5    cout << a*b << endl;
6  }
7  // assembly
8  // ..
9  // mulsd   -40(%rbp), %xmm0
10 // ..
```

Listing 14.3: Multiplication using a standard `double`.

In the assembly excerpts it can be seen that by using the `SIMD_Class` type indeed two multiplications are carried out with one instruction (`mulpd`: multiply 'packed double'). In the `double` implementation, the instruction `mulsd` (multiply 'scalar double') is executed.

Integration and density contraction codes have been modified for explicit use of streaming SIMD extensions using this type. The results obtained by using these extensions are summarized in table 14.3, for various molecules and basis sets of differing quality, compared to the `double` implementation. For the integration the rolled, splitted ansatz with the E_2 transfer is used, in chemists' notation center-angular-zeta iteration.

		double	SIMD_Class	Speedup
Alanine	cc-pVDZ	2.75	2.47	1.11
Alanine	cc-pVTZ	26.07	19.51	1.34
Alanine	cc-pVQZ	312.80	208.86	1.50
Serotonine	cc-pVTZ	217.95	161.52	1.35
Serotonine	cc-pVQZ	2521.12	1685.52	1.50
Serotonine	cc-pV5Z	26 939.78	17 459.71	1.54
Serotonine	roos-aug-dz-ano	1253.70	947.23	1.32
Serotonine	roos-aug-tz-ano	6588.91	4339.88	1.52
$C_{36}H_{53}N_7O_9$	Def2-SVP	748.28	679.57	1.10
$C_{36}H_{53}N_7O_9$	cc-pVDZ	868.87	787.04	1.10

Table 14.3.: Performances of the standard `double` implementation compared to the `SIMD_Class` realization, in seconds.

In all cases increased performance can be observed. Independent of molecular size, the speedups obtained depend on quality and type of the basis set. For Def2 and correlation consistent basis sets, speedups of about 1.1 are gained for basis sets supporting *d*-functions. For basis sets with up to *f*-functions, speedups of about 1.3 are obtained. Speedups of 1.5 and higher are observed for basis sets with quintuple-zeta and higher quality.

For the generally contracted roos-aug-*-ano basis sets, speedups of 1.3 are already obtained at double-zeta quality. In the triple-zeta case a speedup of 1.5 is observed.

Part III.

Results

15. Comparisons to the TURBOMOLE and MOLPRO Program Packages

The final version of the QOL SCF program developed in this work has been compared to the SCF versions of the TURBOMOLE and MOLPRO packages.

In a first step, the accuracy of the QOL has been compared to the commercially available codes. Obviously, the main reason for doing so is to assure the correctness of the developed codes. As will be shown, doing such comparisons based solely on the energies obtained during the SCF calculation is not sufficient to investigate the influence of various optimizations present in the commercial codes. To overcome this, the actual matrices involved have been obtained during the MOLPRO and TURBOMOLE calculations, to the extent possible. With this information available, the influence of various optimizations, such as convergence acceleration, have been studied at great detail. This information has then be used to fine-tune the commercial codes, to assure that identical optimizations were used, at least qualitatively. Such runs then have been the basis for the performance comparisons performed.

15.1. MOLPRO

In MOLPRO, to the authors knowledge, no option exists to print actual matrices (like the density or the Fock matrix) in each iteration. To nevertheless obtain the matrices of interest, in each iteration, a modified version of MOLPRO has been used.

The code for the direct SCF procedure is located in the MOLPRO source directory in the file `src/scf/rhfpro.F`. This file has been modified to write the matrices of interest to the output, in each iteration. This has been accomplished by the following output statement.

```
1  write(*,*) 'density',iter, (q(idapos+jheld), jheld=0,ntdg-1)
```

Here, the density is written, all elements in one line. The string `density` is user defined, and `iter` is the current SCF iteration. The first density element can be accessed by `q(idapos)`, and the remaining ones by incrementing the user defined counter `jheld`. In the source file, this statement has been put after the call to `update_densities` (line 497). The write statements for the other matrices ($\mathbf{F}$, $\mathbf{G}$, $\mathbf{S}$ and $\mathbf{H}^{\mathbf{core}}$) are similar, and not repeated here.

It shall be noted that the matrix elements obtained are given in full accuracy. Printed are 16 significant digits, and the according exponent.

15.1.1. Accuracy

Critical for in depth accuracy comparisons are the inputs used for the calculations. The identity of molecular geometries and basis sets has been assured as described in section 8.2.1. Of similar importance is the initial guess for the molecular orbitals. In MOLPRO, the default guess has been chosen, which uses atomic densities. To ensure identical start vectors in the QOL calculations, the density matrices obtained from the corresponding MOLPRO calculations have been used as initial guess.

Starting points have been default MOLPRO SCF calculations. The input for such a calculation, for C_4H_{10} in a cc-pVTZ basis is shown in listing 15.1.

```
1 direct;
2 nosym;
3 include,/basisSets/emsl/cc-pvtz/HCNOSCl.molpro;
4 bohr;
5 include,/geometries/c4h10.in;
6 {hf,accuracy=8,energy=1e-7,direct,thrdisk=1e999,thrint=1e-11}
```

Listing 15.1: Default MOLPRO input.

All calculations have been performed using the direct SCF ansatz, not exploiting symmetries (lines one and two). Basis sets and molecular geometries (in Bohr) are read from files (lines three to five). Finally, the SCF calculation is performed. For the energy, the convergence threshold has been set to 10^{-7}. Setting the option `accuracy` to eight corresponds to an effective threshold for the change of the norm of the density difference of $\sqrt{10^{-8}} = 10^{-4}$. In MOLPRO, the norm of a symmetric matrix $\mathbf{M}$ with dimension n is calculated as

$$norm(\mathbf{M}) = \frac{\sqrt{\sum_{i<=j}^{n}(M_{ij})^2}}{n(n+1)/2} \tag{15.1}$$

To check for convergence of the density, the density difference is used in this norm: $\mathbf{M} = \mathbf{D}^k - \mathbf{D}^{k-1}$, where k is the current iteration.

Additionally, no integrals are written to disk (`thrdisk`), and the prescreening threshold has been set to 10^{-11}. It shall be noted that the differential density scheme and prescreening on the density matrix are enabled by default.

The results of this calculation can now be compared to the SCF program developed in the course of this work. For the QOL calculation, the same values for energy-, density difference- and prescreening thresholds have been set. The one-electron energy convergence threshold needed for the TURBOMOLE comparisons of section 15.2.1 has been disabled. Also the differential density scheme has been used. As mentioned, the initial density has been taken from the MOLPRO calculation. Additionally, DIIS has been enabled in the QOL, using the four last densities in the interpolation. The DIIS error vector has been set to ($\mathbf{S}^{-1/2}\mathbf{FDS}^{1/2}$ - transpose).

Both calculations reached convergence after eight iterations. The results obtained by comparing both energies and Fock matrices are summarized in table 15.1, for each iteration, respectively. For the energies, absolute and relative errors are shown, in addition

to the norm of the difference of the Fock matrices.

Iteration	ΔE	rel(E)	norm(**ΔF**)
1	$-2.9 \cdot 10^{-10}$	$1.9 \cdot 10^{-12}$	$5.0 \cdot 10^{-10}$
2	$-1.9 \cdot 10^{-9}$	$1.2 \cdot 10^{-11}$	$4.5 \cdot 10^{-10}$
3	$-2.6 \cdot 10^{-9}$	$1.7 \cdot 10^{-11}$	$5.0 \cdot 10^{-10}$
4	$-1.2 \cdot 10^{-8}$	$7.7 \cdot 10^{-11}$	$6.7 \cdot 10^{-10}$
5	$-5.2 \cdot 10^{-9}$	$3.3 \cdot 10^{-11}$	$8.3 \cdot 10^{-10}$
6	$-1.5 \cdot 10^{-7}$	$9.3 \cdot 10^{-10}$	$3.2 \cdot 10^{-6}$
7	$-9.8 \cdot 10^{-9}$	$6.2 \cdot 10^{-11}$	$1.0 \cdot 10^{-6}$
8	$-8.6 \cdot 10^{-9}$	$5.5 \cdot 10^{-11}$	$6.8 \cdot 10^{-7}$

Table 15.1.: Comparison of the default MOLPRO SCF calculation with the QOL results, for C_4H_{10} in the cc-pVTZ basis. For each iteration, absolute and relative energy errors, and Fock matrix difference norms are shown, in Hartree.

With a slight deviation in iteration six, the energies calculated with MOLPRO are reproduced by the QOL SCF program with high accuracy. A different behaviour can be observed for the norm of the Fock matrix differences. It stays about constant for the first five iterations, with a magnitude of about 10^{-10}. After the fifth iteration, a drop of about three orders of magnitude can be observed.

It shall be noted that the Fock matrices calculated by MOLPRO and the QOL still are almost identical. Consider on the one hand that in the context of the performed calculations the density matrices are taken to be converged if the norm of their difference is smaller than 10^{-4}, which is two magnitudes larger than the largest Fock matrix difference norm. On the other hand, as mentioned, no observable change of the energy differences results from the increased difference norms. Nevertheless, the observed changes in magnitude indicate different internal pathways of calculation taken by MOLPRO and the QOL.

Obviously, the change observed in the difference norms is also present in the individual elements of the Fock matrices. For each matrix element, the absolute and relative errors have been calculated, for each iteration, respectively.

$$\Delta F_{ij}^{abs} = |F_{ij}^{MOLPRO} - F_{ij}^{QOL}| \tag{15.2}$$

$$\Delta F_{ij}^{rel} = \frac{\Delta F_{ij}^{abs}}{|F_{ij}^{MOLPRO}|} \tag{15.3}$$

In table 15.2, the maxima of these deviations

$$\epsilon^{abs} = max(\Delta \mathbf{F}^{abs}) \tag{15.4}$$

$$\epsilon^{rel} = max(\Delta \mathbf{F}^{rel}) \tag{15.5}$$

are shown, in addition to the corresponding actual matrix elements.

$$v = arg\ max(\mathbf{F}(\epsilon)) \tag{15.6}$$

Itrn	ϵ^{abs}	v^{MOLPRO}	v^{QOL}	ϵ^{rel}	v^{MOLPRO}	v^{QOL}
1	$3 \cdot 10^{-9}$	$1.755 \cdot 10^{-3}$	$1.755 \cdot 10^{-3}$	$1 \cdot 10^{-2}$	$1.793 \cdot 10^{-8}$	$1.771 \cdot 10^{-8}$
2	$3 \cdot 10^{-9}$	$-1.229 \cdot 10^{-3}$	$-1.229 \cdot 10^{-3}$	$4 \cdot 10^{-2}$	$6.167 \cdot 10^{-9}$	$5.920 \cdot 10^{-9}$
3	$4 \cdot 10^{-9}$	$-1.207 \cdot 10^{-3}$	$-1.207 \cdot 10^{-3}$	$8 \cdot 10^{-3}$	$-1.173 \cdot 10^{-7}$	$-1.163 \cdot 10^{-7}$
4	$5 \cdot 10^{-9}$	$-5.408 \cdot 10^{-4}$	$-5.408 \cdot 10^{-4}$	$4 \cdot 10^{-2}$	$6.323 \cdot 10^{-9}$	$6.094 \cdot 10^{-9}$
5	$6 \cdot 10^{-9}$	$-5.400 \cdot 10^{-4}$	$-5.400 \cdot 10^{-4}$	$7 \cdot 10^{-2}$	$-2.158 \cdot 10^{-8}$	$-2.002 \cdot 10^{-8}$
6	$1 \cdot 10^{-4}$	$-1.121 \cdot 10^{1}$	$-1.121 \cdot 10^{1}$	$1 \cdot 10^{1}$	$4.659 \cdot 10^{-9}$	$6.091 \cdot 10^{-8}$
7	$2 \cdot 10^{-5}$	$-1.122 \cdot 10^{1}$	$-1.122 \cdot 10^{1}$	$7 \cdot 10^{-1}$	$1.305 \cdot 10^{-8}$	$2.165 \cdot 10^{-8}$
8	$1 \cdot 10^{-5}$	$-1.122 \cdot 10^{1}$	$-1.122 \cdot 10^{1}$	$2 \cdot 10^{-1}$	$1.837 \cdot 10^{-8}$	$2.212 \cdot 10^{-8}$

Table 15.2.: Comparison of the default MOLPRO SCF calculation with the QOL results, for C_4H_{10} in the cc-pVTZ basis. For each iteration, the maximum absolute and relative errors of the individual Fock matrix element differences are shown, and the associated matrix elements (in Hartree).

In the first five iterations, as the maximum relative errors show, the individual matrix elements have been reproduced to at least one significant digit. The maximum absolute errors found in the first five iterations are small. Again, starting with the sixth iteration, both absolute and relative errors increase. The maximum relative errors however are connected to matrix elements of small magnitude.

The deviations found in the last three iterations are connected to convergence acceleration methods. MOLPROs default DIIS implementation, for example, is rather sophisticated, and has not been included in the QOL. However, a DIIS implementation using the same error vector as used in the QOL is available in MOLPRO with the keywords `iptyp='base',maxdis=5`. It shall be noted that in MOLPROs implementation, the `maxdis` value is used for the dimension of the DIIS matrix, resulting in an effective DIIS length of four (in accordance to the length set in the QOL calculation). Additionally, level-shifting, which is not available in the QOL, is enabled by default. It has been disabled by adding the option `shift,0.0,0.0` to the MOLPRO input. Finally, reorthonormalization after ten iterations (the default) has effectively been turned off with the option `orth,100`. Listing 15.2 shows the changed `hf` directive.

```
1 {hf,iptyp='base',maxdis=5,diis_start=0,accuracy=8,
2   energy=1e-7,direct,thrdisk=1e999,thrint=1e-11;
3 orth,100
4 shift,0.0,0.0}
```

Listing 15.2: Default MOLPRO input.

Using these options, the Fock matrices calculated by MOLPRO are reproduced by the QOL to great accuracy, in each iteration. This is summarized in table 15.3, again

showing absolute and relative errors of the energies and the norms of the Fock matrix difference.

itrn	ΔE	rel(E)	norm(Δ**F**)
1	$-2.9 \cdot 10^{-10}$	$1.9 \cdot 10^{-12}$	$5.0 \cdot 10^{-10}$
2	$-1.9 \cdot 10^{-9}$	$1.2 \cdot 10^{-11}$	$4.5 \cdot 10^{-10}$
3	$-2.6 \cdot 10^{-9}$	$1.7 \cdot 10^{-11}$	$5.0 \cdot 10^{-10}$
4	$-1.2 \cdot 10^{-8}$	$7.7 \cdot 10^{-11}$	$6.7 \cdot 10^{-10}$
5	$-5.2 \cdot 10^{-9}$	$3.3 \cdot 10^{-11}$	$8.3 \cdot 10^{-10}$
6	$-6.9 \cdot 10^{-9}$	$4.4 \cdot 10^{-11}$	$9.3 \cdot 10^{-10}$
7	$-9.8 \cdot 10^{-9}$	$6.2 \cdot 10^{-11}$	$1.0 \cdot 10^{-9}$
8	$-1.3 \cdot 10^{-8}$	$8.0 \cdot 10^{-11}$	$9.4 \cdot 10^{-10}$

Table 15.3.: Comparison of the adjusted MOLPRO SCF calculation with the QOL results, for C_4H_{10} in the cc-pVTZ basis. For each iteration, absolute and relative energy errors, and Fock matrix difference norms are shown, in Hartree.

Again, the energies are reproduced with high accuracy. Additionally, the increase in the Fock matrix difference norms observed in the previous example vanish.

Similar results are found for the individual matrix elements, as shown in table 15.4. Again, maxima were calculated over the absolute and relative individual matrix element deviations (see equation (15.2) to equation (15.5)). Additionally, the associated matrix elements are shown (equation (15.6)), for each iteration, respectively.

Itrn	ϵ^{abs}	v^{MOLPRO}	v^{QOL}	ϵ^{rel}	v^{MOLPRO}	v^{QOL}
1	$3 \cdot 10^{-9}$	$1.755 \cdot 10^{-3}$	$1.755 \cdot 10^{-3}$	$1 \cdot 10^{-2}$	$1.793 \cdot 10^{-8}$	$1.771 \cdot 10^{-8}$
2	$3 \cdot 10^{-9}$	$-1.229 \cdot 10^{-3}$	$-1.229 \cdot 10^{-3}$	$4 \cdot 10^{-2}$	$6.167 \cdot 10^{-9}$	$5.920 \cdot 10^{-9}$
3	$4 \cdot 10^{-9}$	$-1.207 \cdot 10^{-3}$	$-1.207 \cdot 10^{-3}$	$8 \cdot 10^{-3}$	$-1.173 \cdot 10^{-7}$	$-1.163 \cdot 10^{-7}$
4	$5 \cdot 10^{-9}$	$-5.408 \cdot 10^{-4}$	$-5.408 \cdot 10^{-4}$	$4 \cdot 10^{-2}$	$6.323 \cdot 10^{-9}$	$6.094 \cdot 10^{-9}$
5	$6 \cdot 10^{-9}$	$-5.400 \cdot 10^{-4}$	$-5.400 \cdot 10^{-4}$	$7 \cdot 10^{-2}$	$-2.158 \cdot 10^{-8}$	$-2.002 \cdot 10^{-8}$
6	$6 \cdot 10^{-9}$	$-3.556 \cdot 10^{-3}$	$-3.556 \cdot 10^{-3}$	$3 \cdot 10^{-2}$	$5.927 \cdot 10^{-8}$	$6.091 \cdot 10^{-8}$
7	$7 \cdot 10^{-9}$	$-3.556 \cdot 10^{-3}$	$-3.556 \cdot 10^{-3}$	$9 \cdot 10^{-2}$	$1.981 \cdot 10^{-8}$	$2.165 \cdot 10^{-8}$
8	$6 \cdot 10^{-9}$	$3.196 \cdot 10^{-3}$	$3.196 \cdot 10^{-3}$	$1 \cdot 10^{-1}$	$2.016 \cdot 10^{-8}$	$2.212 \cdot 10^{-8}$

Table 15.4.: Comparison of the adjusted MOLPRO SCF calculation with the QOL results, for C_4H_{10} in the cc-pVTZ basis. For each iteration, the maximum absolute and relative errors of the individual Fock matrix element differences are shown, and the associated matrix elements (in Hartree).

With maximum absolute deviations of about 10^{-9} in each iteration, at least one significant digit has been reproduced, for all matrix elements. Again, the largest relative errors correspond to Fock matrix elements of small magnitude.

It shall be noted here that the deviations observed in small matrix elements are numerical noise. Each matrix element is generated from a huge sum of numbers of varying

magnitudes. As has been explained in section 9.1.2, such sums are easily affected by numerical errors, especially if the final result is small. In such cases, just two almost canceling numbers of higher magnitude will taint the final result.

Another detail worth mentioning is the number of iterations in MOLPRO calculations. As it turns out, one additional iteration is performed by MOLPRO after convergence is reached. In this final iteration, some orthonormalization is performed, not changing the final SCF energy. This has been confirmed by inspecting the source code of MOLPRO.

Concluding, the energies and Fock matrices calculated by MOLPRO have been reproduced almost to the number by the QOL calculation. This indicates strongly that, at least qualitatively, the same internal pathways for the calculations have been performed. Different internal routes, although not observable in the energy differences, do show in the corresponding Fock matrices, as has been shown.

This analysis has been performed for all systems shown in the following performance comparison.

15.1.2. Performance

The performance comparisons presented here are based on the insights gained in the accuracy analysis. Having established qualitatively identical internal calculation pathways, a fair performance analysis can be executed.

The QOL SCF program has been run with the options described in section 15.1.1. The input densities used have been obtained using the modified MOLPRO version. The results obtained from these calculations can then be compared to the corresponding MOLPRO runs. For these performance runs the original, unmodified, serial version of MOLPRO has been used, which was built from the sources as described in section 8.1. For all calculations the input options of listing 15.2 have been used.

Each of the calculations has been run on one of the Intel(R) Xeon(R) E5520 hosts of section 8.3.1, assuring the same physical layout (CPU type and cache sizes) has been used. Additionally, great care has been taken to ensure the absence of interfering processes. With the exception of operating system relevant processes, no other jobs than the SCF calculation were allowed to be executed on the particularly systems.

The performance of the calculations performed has been determined in terms of wall clock time only. For both programs, several runs on the same system have been performed to check for timing deviations. The `time -p` command has been used to compare internal with external timings. In MOLPRO calculations, internal timings can be taken from the output file. In the QOL, timings are gathered with `time.h`, and for wall clock timings the `clockid_t CLOCK_REALTIME` has been used. The results for five separate SCF runs on C_4H_{10} in a cc-pVTZ basis are shown in table 15.5.

	MOLPRO		QOL	
	Internal	`time -p`	Internal	`time -p`
	264.09	264.15	96.46	96.77
	263.58	263.64	96.39	96.70
	262.81	262.88	96.15	96.46
	262.57	262.63	96.92	97.26
	262.70	262.75	96.64	96.94
Average	263.15 ± 0.66	263.21 ± 0.25	96.51 ± 0.29	96.83 ± 0.30

Table 15.5.: Timings obtained for the SCF calculation of C_4H_{10}/cc-pVTZ in five separate runs, for MOLPRO (adjusted input) and the QOL, in seconds.

For both programs, the internally obtained wall clock timings correspond to the externally measured time. Additionally, no severe deviations of the timings are observed in separate runs, respectively. The timings reported in the following are therefore based on single runs.

To gather more performance relevant data, all calculations have been run with the `perf` utility. By using this utility, no overhead is introduced. Running the C_4H_{10}/cc-pVTZ MOLPRO calculation with `perf`, internally 264.27 seconds are reported, the `perf` utility giving 264.34 seconds. For the corresponding QOL calculation, 96.60 seconds are measured internally, and 96.92 seconds by `perf`. Both results are in accordance with the timings measured without `perf`.

As mentioned previously, in MOLPRO calculations one additional orthonormalization iteration is performed after convergence. This has been accounted for in the speedups reported in table 15.6. In the C_4H_{10}/cc-pVTZ case, the MOLPRO output gives the time for this iteration as 21.3 seconds. The MOLPRO calculation without this iteration then takes 242.97 seconds. Recalling the accuracy analysis of section 15.1.1, the same information is obtained by the QOL, in 96.60 seconds. This corresponds to a speedup of 2.5 of the QOL SCF program versus MOLPROs SCF version.

Using an identical approach, timings have been generated for a set of molecules, in various basis sets. The speedups obtained are summarized in table 15.6.

	C_4H_{10}	Alanine	Serotonine	$C_{36}H_{53}N_7O_9$
Def2-SVP	1.7	1.7	1.7	1.8
Def2-TZVP	1.7	1.9	2.1	2.2
Def2-QZVP	2.5	2.5	2.6	-
cc-pVDZ-ogc	1.7	1.7	1.8	1.8
cc-pVTZ-ogc	2.4	2.4	2.4	2.5
cc-pVQZ-ogc	2.6	2.6	2.6	-
cc-pV5Z-ogc	2.8	2.8	-	-
cc-pV6Z-ogc	3.1	3.3	-	-
cc-pVDZ	1.7	1.7	1.8	1.9
cc-pVTZ	2.5	2.5	2.6	2.8
cc-pVQZ	2.7	2.7	2.8	-
cc-pV5Z	2.9	2.8	2.9	-
cc-pV6Z	3.0	3.1	3.4	-
roos-aug-dz-ano	1.2	1.3	1.3	-
roos-aug-tz-ano	1.9	2.0	2.2	-
ano-rcc	4.0	4.1	4.9	-

Table 15.6.: Speedups of SCF runs performed with the QOL, versus MOLPRO.

In all cases, the QOL SCF program is faster than MOLPROs version. For the roos-aug-dz-ano bases, almost equal timings were obtained. In the case of the ano-rcc basis sets, speedups of about four to five can be observed. For all molecules and basis sets, the speedups increase with increasing molecule- or basis size.

15.2. TURBOMOLE

No source code has been available for the TURBOMOLE program package. Nevertheless, the matrices of interest could be obtained, on a per iteration basis. This has been accomplished by setting the `debug` option of the `scfdiis` keyword to 2. In doing so density and Fock matrices are written in each iteration, except the last. It has to be noted that a fixed format is used for this output. One digit before, and eight digits after the decimal point are given, regardless of the actual magnitude of the number. The number 10^{-8} is printed as 0.00000001. Full accuracy is not available.

15.2.1. Accuracy

Unfortunately, the TURBOMOLE and MOLPRO programs can not be adjusted to give identical results. The main reason is that TURBOMOLE uses the change of the one-electron energy as a convergence criterion, which is not available in MOLPRO. The threshold for this criterion can not be set, and therefore not be disabled (at least to the authors knowledge). Among the other reasons are mutually exclusive initial

guesses, i.e. the extended Hückel guess of TURBOMOLE is not present in MOLPRO, and MOLPROs atomic density guess not in TURBOMOLE. Using the null guess was also not possible, as MOLPRO then starts the DIIS expansion at iteration two, which could not be overwritten by the `diis_start` option. This behaviour is hard coded in the MOLPRO sources and has not been changed. Concluding, the QOL SCF calculations of the comparison to MOLPRO could not be used in the TURBOMOLE comparison, they had to be repeated with a different set of input options.

Similar to the steps explained in section 15.1.1, advanced convergence acceleration techniques available in TURBOMOLE have been turned of. In detail, damping has been turned off by setting the `start`, `step` and `min` options of the `scfdamp` keyword to zero. Orbital shifting has been turned off by setting `scforbitalshift` to `noautomatic`. Damping in DIIS has been disabled by setting the `qscal` option of the `scfdiis` keyword to one. The following other options have been set: The default DIIS error vector $\mathbf{FDS} - \mathbf{SDF}$ has been used, with an expansion length of five. To prevent integrals to be written to disc, `thize` and `thime` have been set to the largest possible values of 10^{99} and 9999. The prescreening threshold `scftol` was set to 10^{-11}. The differential density scheme has been used, restricting TURBOMOLE to use only the last differential density by setting `scfdenapproxl` to one. The energy convergence criterion `scfconv` has been set to seven, corresponding to an convergence threshold of 10^{-7}. This results in a one-electron energy threshold of 10^{-4}. For all calculations, geometries and basis sets have been used as described in section 8.2.1. Geometries have been transformed internally to generalized internal coordinates (`ired` in `define`). The external basis sets have been made available to TURBOMOLE via the `~/.definerc` file and used with the `lib` sub-menu of `define`. MO start vectors have been generated with `define`, using the extended Hückel guess (`eht`).

The QOL calculations then have been performed emulating the TURBOMOLE inputs. Convergence thresholds for energy and one-electron energy were set to 10^{-7} and 10^{-4}, respectively. Density difference convergence has been turned off. DIIS convergence acceleration has been turned on, using the error vector $\mathbf{FDS} - \mathbf{SDF}$, for the last five iterations. The differential density scheme has been used, and the prescreening threshold was set to 10^{-11}. Initial densities were obtained from the densities printed by TURBOMOLE.

By performing the calculations using these options, the TURBOMOLE calculations could be reproduced by the QOL SCF program to great accuracy. This is shown in table 15.7, for C_4H_{10} in a cc-pVTZ basis. Both calculations reached convergence after eight iterations. Summarized are absolute and relative errors of the energy, and the norms of the Fock matrix differences (for the last iteration, TURBOMOLEs Fock matrix was not available).

itrn	ΔE	rel(E)	norm(Δ**F**)
1	$-4.4 \cdot 10^{-7}$	$2.8 \cdot 10^{-9}$	$5.5 \cdot 10^{-9}$
2	$-2.0 \cdot 10^{-8}$	$1.3 \cdot 10^{-10}$	$3.5 \cdot 10^{-9}$
3	$-5.8 \cdot 10^{-9}$	$3.7 \cdot 10^{-11}$	$2.9 \cdot 10^{-9}$
4	$-7.5 \cdot 10^{-9}$	$4.7 \cdot 10^{-11}$	$2.9 \cdot 10^{-9}$
5	$-1.1 \cdot 10^{-8}$	$6.8 \cdot 10^{-11}$	$2.9 \cdot 10^{-9}$
6	$-1.1 \cdot 10^{-8}$	$6.7 \cdot 10^{-11}$	$2.9 \cdot 10^{-9}$
7	$-1.1 \cdot 10^{-8}$	$7.1 \cdot 10^{-11}$	$2.9 \cdot 10^{-9}$
8	$-1.3 \cdot 10^{-8}$	$8.5 \cdot 10^{-11}$	-na-

Table 15.7.: Comparison of the adjusted TURBOMOLE SCF calculation with the QOL results, for C_4H_{10} in the cc-pVTZ basis. For each iteration, absolute and relative energy errors, and Fock matrix difference norms are shown, in Hartree.

In all iterations, both energies and Fock matrices are reproduced almost identically. The somewhat diminished accuracy of the energy in the first iteration is connected to the density guess. In the QOL calculation, the density printed by TURBOMOLE was used. As mentioned, this density is not available in full accuracy, but only to the eighth digit after the decimal point. This leads to the slight discrepancy of the energy observed in the first iteration. The (QOL) Fock matrix built from this density however is then almost identical to the one obtained with TURBOMOLE. This observation again shows how sensitive the calculations respond to slight changes of the internal quantities.

Again, also individual matrix elements of the Fock matrix were investigated. The maximum absolute and relative errors (equation (15.2) to equation (15.5)), and the associated matrix elements (equation (15.6)) are shown in table 15.8, for each iteration, respectively.

Itrn	ϵ^{abs}	$v^{TURBOMOLE}$	v^{QOL}	ϵ^{rel}	$v^{TURBOMOLE}$	v^{QOL}
1	$7 \cdot 10^{-8}$	$-1.151 \cdot 10^{1}$	$-1.151 \cdot 10^{1}$	$2 \cdot 10^{-1}$	$-2.000 \cdot 10^{-8}$	$-2.423 \cdot 10^{-8}$
2	$5 \cdot 10^{-8}$	$-1.108 \cdot 10^{1}$	$-1.108 \cdot 10^{1}$	$2 \cdot 10^{-1}$	$-1.000 \cdot 10^{-8}$	$-1.175 \cdot 10^{-8}$
3	$3 \cdot 10^{-8}$	$-4.983 \cdot 10^{-5}$	$-4.986 \cdot 10^{-5}$	$5 \cdot 10^{-2}$	$1.000 \cdot 10^{-7}$	$9.544 \cdot 10^{-8}$
4	$2 \cdot 10^{-8}$	$4.568 \cdot 10^{-5}$	$4.570 \cdot 10^{-5}$	$5 \cdot 10^{-2}$	$6.000 \cdot 10^{-8}$	$5.719 \cdot 10^{-8}$
5	$3 \cdot 10^{-8}$	$-4.536 \cdot 10^{-5}$	$-4.539 \cdot 10^{-5}$	$7 \cdot 10^{-2}$	$5.000 \cdot 10^{-8}$	$5.372 \cdot 10^{-8}$
6	$3 \cdot 10^{-8}$	$4.509 \cdot 10^{-5}$	$4.512 \cdot 10^{-5}$	$1 \cdot 10^{-1}$	$2.000 \cdot 10^{-8}$	$1.715 \cdot 10^{-8}$
7	$2 \cdot 10^{-8}$	$4.514 \cdot 10^{-5}$	$4.516 \cdot 10^{-5}$	$3 \cdot 10^{-1}$	$1.000 \cdot 10^{-8}$	$1.319 \cdot 10^{-8}$

Table 15.8.: Comparison of the adjusted TURBOMOLE SCF calculation with the QOL results, for C_4H_{10} in the cc-pVTZ basis. For each iteration, the maximum absolute and relative errors of the individual Fock matrix element differences are shown, and the associated matrix elements (in Hartree).

In all iterations, the Fock matrices were reproduced within the accuracy available from TURBOMOLE. For the matrix elements connected to the maximum relative errors, for the TURBOMOLE case, the shown non-zero digits were the only numerical information

present. Again, it shall be noted that the diminished accuracy present in the QOL initial density does not show in the calculated Fock matrices.

Concluding, it could be shown that, by setting the appropriate options, the results obtained with TURBOMOLE were reproduced by the QOL with high accuracy. Given the sensitivity of the resulting quantities, qualitatively identical internal calculation pathways were assured.

This analysis has been performed for all systems shown in the following performance comparison.

15.2.2. Performance

Having established qualitatively identical calculation pathways for TURBOMOLE and the QOL, performance analyses were conducted. Separate calculations have been performed, using the options established in the accuracy comparison, with the exception of printing the matrices. This has been done to not taint the results. As for the MOLPRO calculations of section 15.1.2, all calculations were executed on the Intel(R) Xeon(R) E5530 hosts of the compute cluster of the Institute for Theoretical Chemistry (see section 8.3.1). Only serial runs were performed, again assuring the absence of interfering processes.

Again the performance analysis has been based on the wall time needed for the calculations. In a first step, the internally reported time of TURBOMOLE has been compared to the externally measured one (`time -p`). Additionally it has been assured that no significant deviations in the timings appear in several runs on the same system. The results obtained with TURBOMOLE for C_4H_{10} in the cc-pVTZ basis are shown in table 15.9. Although already shown in table 15.5, the QOL calculations are repeated here, using the settings of the TURBOMOLE context.

	TURBOMOLE		QOL	
	Internal	`time -p`	Internal	`time -p`
	326	326.31	115.44	116.10
	327	327.51	115.59	116.89
	326	326.49	115.56	116.70
	327	326.70	115.36	116.64
	327	326.86	115.22	115.58
Average	326.6 ± 0.55	326.77 ± 0.46	115.43 ± 0.15	116.38 ± 0.55

Table 15.9.: Timings obtained for the SCF calculation of C_4H_{10}/cc-pVTZ in five separate runs with TURBOMOLE, using the adjusted input, in seconds.

Again, the internally reported timings correspond to the externally measured ones, and no severe deviations in the timings are observed. The speedups reported in the following are therefore based on singular runs.

It shall be noted here that the significant differences for the time required to calculate C_4H_{10} in cc-pVTZ basis with the QOL are caused by the different convergence criteria

applied. Besides the total energy criterion, TURBOMOLE uses the change in the one-electron energy. MOLPRO however uses the norm of the density difference and the total energy. The corresponding changes for the C_4H_{10}/cc-pVTZ (QOL) calculation are summarized in table 15.10.

Itrn	ΔE	ΔE_1	norm($\Delta\mathbf{D}$)
1	$-2.87 \cdot 10^{2}$	$-4.62 \cdot 10^{2}$	$3.28 \cdot 10^{-2}$
2	$-5.15 \cdot 10^{-1}$	-8.38	$1.74 \cdot 10^{-2}$
3	$-4.11 \cdot 10^{-2}$	3.08	$4.42 \cdot 10^{-3}$
4	$-5.40 \cdot 10^{-3}$	$-8.12 \cdot 10^{-1}$	$1.04 \cdot 10^{-3}$
5	$-1.71 \cdot 10^{-4}$	$-1.50 \cdot 10^{-2}$	$2.12 \cdot 10^{-4}$
6	$-1.40 \cdot 10^{-5}$	$3.91 \cdot 10^{-3}$	$6.66 \cdot 10^{-5}$
7	$-8.62 \cdot 10^{-7}$	$3.68 \cdot 10^{-3}$	$1.98 \cdot 10^{-5}$
8	$-2.14 \cdot 10^{-8}$	$-3.07 \cdot 10^{-4}$	$3.92 \cdot 10^{-6}$
9	$-3.86 \cdot 10^{-10}$	$1.04 \cdot 10^{-4}$	$9.90 \cdot 10^{-7}$
10	$-2.06 \cdot 10^{-9}$	$-1.87 \cdot 10^{-5}$	$1.46 \cdot 10^{-7}$

Table 15.10.: Changes of total energy, one-electron energy and density difference norm during the calculation of C_4H_{10} in cc-pVTZ basis. Obtained with the QOL SCF program, in Hartree.

For this calculation, the norm of the density difference satisfies the convergence criterion of 10^{-4} in the sixth iteration. The total energy criterion of 10^{-7} is satisfied in iteration eight. The one-electron energy criterion of 10^{-4} at iteration 10. With the MOLPRO criteria, the calculation is considered to have converged at iteration eight. TURBOMOLE however considers the calculation converged at iteration ten.

Contrary to MOLPRO, all TURBOMOLE and QOL calculations converged after the same number of iterations, respectively. No adjustments were applied. The results obtained for the timing comparison are summarized in table 15.11. Shown are the speedups of the QOL SCF code versus TURBOMOLE dscf program.

	C_4H_{10}	Alanine	Serotonine	$C_{36}H_{53}N_7O_9$
Def2-SVP	0.7	0.8	0.8	0.7
Def2-TZVP	1.0	1.1	1.1	0.9
Def2-QZVP	1.4	1.4	1.3	-
cc-pVDZ-ogc	1.2	1.4	1.3	1.0
cc-pVTZ-ogc	1.5	1.6	1.5	1.3
cc-pVQZ-ogc	1.6	1.6	1.5	-
cc-pV5Z-ogc	1.6	1.6	1.5	-
cc-pV6Z-ogc	1.6	1.6	1.5	-
cc-pVDZ	2.4	3.0	3.2	2.9
cc-pVTZ	2.8	3.2	3.2	3.1
cc-pVQZ	2.6	2.8	2.7	-
cc-pV5Z	2.2	2.3	2.3	-
cc-pV6Z	2.0	2.1	-	-
roos-aug-dz-ano	27.6	34.1	34.5	-

Table 15.11.: Speedups of SCF runs performed with the QOL, versus TURBOMOLE.

TURBOMOLE is specially optimized for effective treatment of segmented basis sets. For the Def2-SVP basis set, the TURBOMOLE calculations were in all cases executed in less time than needed with the QOL code. However, with increasing angular momenta of the basis, the QOL SCF program started to need less time than TURBOMOLE, although only slightly. With the Def2-TZVP bases, about similar results were obtained, the QOL then being faster for Def2-QZVP bases.

Being optimized for segmented basis sets, generally contracted bases are handled poorly by TURBOMOLE, explaining the huge speedups observed for the ANO basis.

The correlation consistent basis sets are somewhat in the middle of those two extremes. In the QOL case, based on the internal structure (see section 14), the same amount of time is needed for a particular quality, regardless of the use of optimized general contractions.

With TURBOMOLEs internal cc-pV*Z-ogc block-segmented basis sets (figure 8.1d), speedups of about 1.5 were observed, in most cases. Again the speedups increase with the basis size. A decrease is visible for large molecules.

A somewhat different trend is present in the correlation consistent basis sets without general contractions. Not optimized for such basis sets, TURBOMOLE takes longer to finish these calculations, up to a factor of three for $C_{36}H_{53}N_7O_9$. With increasing basis set size, the speedups first increase, peaking at the triple-zeta quality basis sets. For basis sets of higher angular momenta the speedups then decrease successively.

A similar behaviour can be observed for the dependence on molecular size, for constant basis sets. The speedups first increase, having their maxima by alanine or serotonine. The largest investigated molecule $C_{36}H_{53}N_7O_9$ then shows decreased speedups, sometimes lower than observed for butane.

16. Applications

In the SCF program developed in this work no hard coded restrictions exist with respect to the size of the system under investigation. This shall be demonstrated with the results obtained for the 1021 atom cluster $C_{315}Cl_1H_{511}N_{82}O_{107}S_5$ [64] shown in figure 16.1.

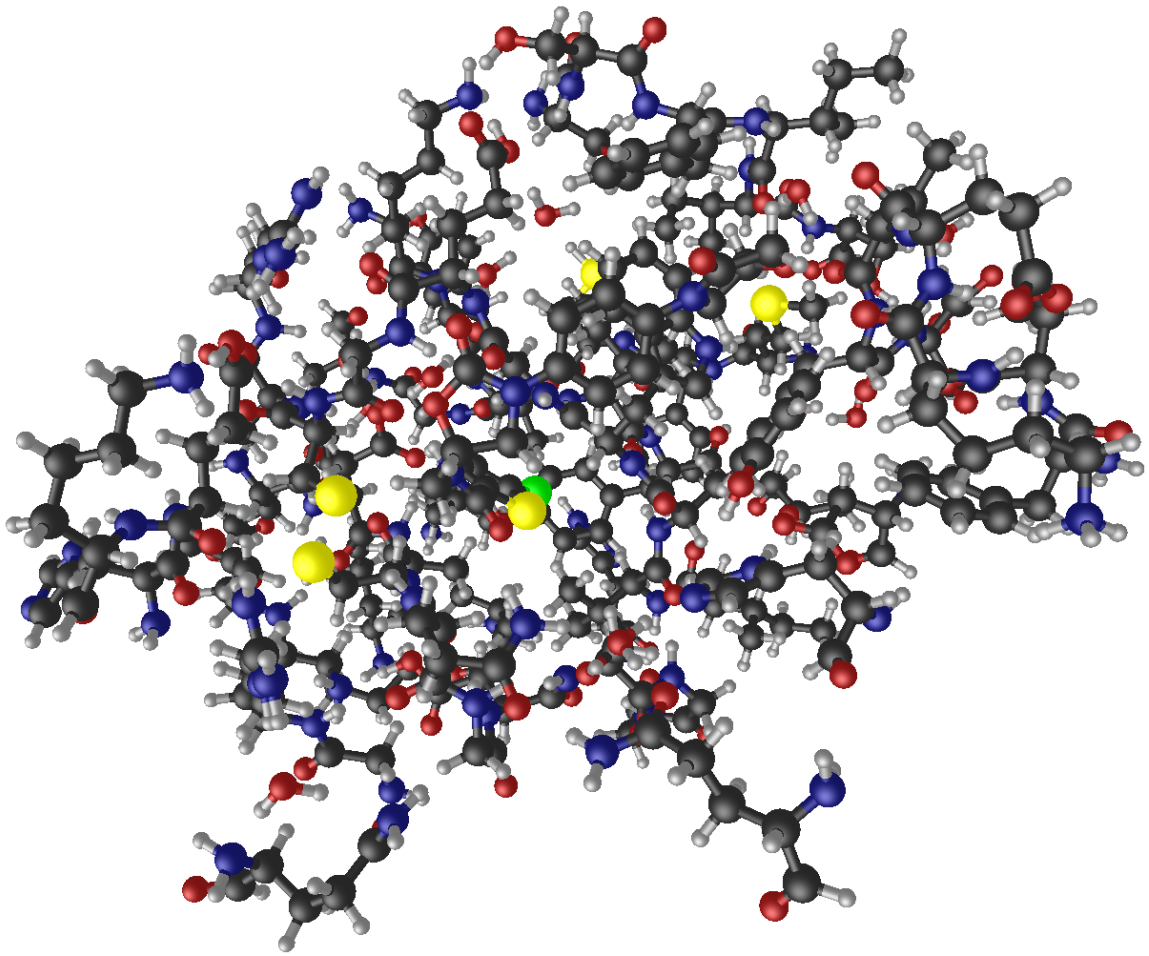


Figure 16.1.: Structure of the 1021 atom cluster $C_{315}Cl_1H_{511}N_{82}O_{107}S_5$.

The calculation presented here has been carried out in a genuine cc-pVDZ basis (without using optimized general contractions, see section 8.2.1). Using the cc-pVDZ basis for this cluster results in 17437 basis functions in the primitive, and 9719 in the radial

contracted, angular transformed basis. The calculation has been performed on a multi-core host at the facilities of the Institute of Theoretical Chemistry, University of Cologne. On the used host 32 Intel(R) Xeon(R) E5-4620 CPU's with a clock rate of 2.2 GHz are available. The OMP parallelized version of the SCF program was used, utilizing all 32 cores. With an atomic densities guess and convergence thresholds of 10^{-6} and 10^{-4} for energy and density difference norm, the calculation converged after 13 iterations to an energy of -27151.235243 Hartree. DIIS convergence acceleration has been used with the $\mathbf{FDS} - \mathbf{SDF}$ error vector and an expansion length of five. The differential density scheme was used, and integral contributions smaller 10^{-11} were neglected.

Finishing the calculation took 35 days, 9 hours and 36 minutes of CPU time. With the use of OMP parallelization, 28 hours and 38 minutes of real time were needed, corresponding to a factor of about 30.

Two aspects shall be pointed out. On one hand, the pre SCF initializations are negligible compared to the actual SCF calculation. In the initialization step, one-electron and Schwarz integrals are calculated, besides the initialization of various other quantities. The 3 hours 13 minutes CPU time needed correspond to about 13 minutes realtime achieved by OMP parallelization.

On the other hand, the iteration steps are dominated by diagonalization of the Fock matrix and the calculation and contraction of the two-electron integrals. Table 16.1 shows the observed CPU timings, for all iterations. In the first iteration the Fock matrix is not diagonalized, the density is obtained from the guess. Additionally the percentages of the diagonalization and integration/contraction steps with respect to the time needed for the whole iteration are shown.

	Fock matrix diagonalization		Integration/density contraction	
Itrn	Time	Percentage of whole iteration	Time	Percentage of whole iteration
1	0	0	32415	98
2	5560	2	344748	98
3	53396	15	310589	85
4	54798	16	279259	83
5	54541	18	254848	82
6	54397	20	214340	80
7	55036	22	189077	77
8	54675	24	169474	75
9	54605	25	164294	75
10	55074	28	141933	72
11	54239	31	120229	69
12	54754	32	115184	67
13	54455	36	96012	64

Table 16.1.: CPU times for Fock matrix diagonalization and integration/contraction steps, per iteration, in seconds. Percentages with respect to the whole iteration.

The other steps to be performed in each iteration, such as the basis transformation of the density and Fock matrices, as well as their block linearization/delinearization and the update of the density thresholds are negligible compared to the diagonalization and integration/contraction steps. For these steps about 22 seconds (CPU time) are needed in the calculation shown. The DIIS step takes about 600 seconds of CPU time, which is still negligible.

With exception of iteration two, almost constant time is needed to diagonalize the Fock matrix. Due to the use of the differential density scheme, more and more integral contributions can be neglected with increasing convergence of the density. The resulting decrease of time needed for the integration/contraction step shifts the relative share needed compared to the Fock matrix diagonalization. Being almost negligible in the first few iterations, the percentage of time needed for the diagonalization step increases to almost 40 percent in the last iteration.

It shall be noted that this calculation could not be carried out using the TURBOMOLE [11] and MOLPRO [10] versions available. In both programs restrictions exist for the number of atoms. In TURBOMOLE, a maximum of 700 atoms is supported. In MOLPRO, a hardcoded limit of 1000 atoms exists.

Part IV.

Summary and Outlook

In this work, a highly efficient restricted closed-shell Hartree-Fock SCF program has been developed. In technical realizations of the Hartree-Fock equations, the main bottleneck is the evaluation and processing of the two-electron integrals. These integrals scale with the fourth power of the system size. The difficulties arising in their handling have been addressed at several levels.

Connected to the two-electron integrals are different bases appearing in Hartree-Fock implementations. For ease of integration, integrals are evaluated over primitive Cartesian Gaussians. The Hartree-Fock equations however are solved in a basis of radial contracted spherical harmonics. In this context, the integrals are transformed to the radial angular transformed basis after evaluation. This transformation is avoided in the ansatz presented in this work, by transforming the density and Fock matrices instead. In particular, the radial angular transformed density is transformed to the primitive basis prior to integral evaluation. During the integration step, this primitive density is contracted with the integrals to give the Fock matrix. The final Fock matrix, in primitive basis, is then transformed back to the radial angular transformed basis, in which it is diagonalized to give the updated density matrix.

The evaluation of the integrals is based on the recursion relations of Obara and Saika. The numerical instabilities inherent in the common four step Obara-Saika scheme have been analyzed and successfully removed. For the implementation a code-generating ansatz has been chosen. Instead of evaluating the complicated recursion relations at runtime, the iterative structure is processed at compile time. A black box code-generator has been realized, by which the recursion relations are analyzed, and optimized code is generated in an automated way, for batches of all angular momenta. These codes are then called at runtime.

The contraction of the integrals with the density has also been addressed using the code-generating ansatz. Optimized code is generated at compile time in a black box fashion, for batches of arbitrary angular momenta. Contraction codes for both primitive and spherical harmonic bases have been realized.

The implementations of both integration and density contraction have been extended to the explicit use of streaming SIMD extensions. Overall speedups of the calculations of up to a factor of 1.5 were observed.

To accelerate individual iterations, prescreening of integrals has been included. Based on the Schwarz inequality, the magnitude of integrals is estimated prior to the actual evaluation. For integrals which give negligible contributions, their calculation can be omitted, which drastically reduces the amount of integrals to be calculated. This is even more amplified by incremental Fock matrix generation schemes, such as the differential density scheme implemented. With increasing convergence of the density, more and more integrals can be neglected, leading to decreasing iteration times throughout a calculation. For the technical realization a hierarchical ansatz has been used, capable of neglecting whole groups of integrals, at different levels of the Fock matrix generation. This turned out to be quite efficient, especially for large molecules.

Convergence acceleration has been realized with the DIIS scheme, in which the current density matrix is approximated by linear combinations of a given number of previous densities.

The developed SCF program is competitive to commercially available solutions, both in terms of capability and performance. It therefore constitutes no proof of concept study. In the comparisons to MOLPRO, speedups of 1.2 to 4.9 have been observed, depending on molecule- and basis size. For common correlation consistent basis sets speedups of 1.7 to 3.4 are reported. The best results of speedups of 4.0 to 4.9 were obtained for the ano-rcc basis sets with generalized contractions.

In comparisons to TURBOMOLE, the observed speedups are not as pronounced, since an optimized treatment of segmented basis sets is used in this program. It shall however be noted that TURBOMOLE is not well suited for generalized contractions.

Although comparable in most aspects of capability, the program developed lacks functionalities present in commercially available programs. First of all, gradients are not available. Furthermore, advanced convergence acceleration and stabilization methods are missing. The DIIS approach implemented could be improved, and level-shifting and damping included.

Conceptually, the work presented here for restricted closed-shell systems should easily be extended to the unrestricted open-shell case.

In terms of performance, several extensions seem promising. On one hand, prescreening might be further optimized by including estimates beyond the Schwarz inequality. On the other hand, reintroducing integral transformations might prove favourable in some cases, especially in the context of segmented basis sets. Finally there are hints that the Obara-Saika integration might be improved by drastically reducing the number of intermediates needed to calculate certain batches.

Bibliography

[1] E. Schrödinger, Phys. Rev. **28**, 1049 (1926).

[2] M. Born and J. R. Oppenheimer, Ann. Rev. Lett. **389**, 457 (1927).

[3] D. R. Hartree, Proc. Cambr. Phil. Soc. **24**, 89 (1928).

[4] D. R. Hartree, Proc. Cambr. Phil. Soc. **24**, 111 (1928).

[5] D. R. Hartree, Proc. Cambr. Phil. Soc. **24**, 426 (1928).

[6] V. Fock, Z. Phy. **61**, 126 (1930).

[7] J. C. Slater, Phys. Rev. **35**, 210 (1930).

[8] C. C. J. Roothaan, Rev. Mod. Phys. **23**, 69 (1951).

[9] G. G. Hall, Proc. Roy. Soc. London Ser. A **205**, 541 (1951).

[10] H.-J. Werner et al., Molpro, version 2012.1, a package of ab initio programs, 2012, see http://www.molpro.net.

[11] TURBOMOLE V6.6 2014, a development of University of Karlsruhe and Forschungszentrum Karlsruhe GmbH, 1989-2007, TURBOMOLE GmbH, since 2007; available from
`http://www.turbomole.com`.

[12] Dalton, a molecular electronic structure program, release dalton2016.0 (2015), see http://daltonprogram.org.

[13] J. Almlöf, K. Faegri, and K. Korsell, J. Comput. Chem. **3**, 385 (1982).

[14] M. Häser and R. Ahlrichs, J. Comput. Chem. **10**, 104 (1989).

[15] D. S. Lambrecht and C. Ochsenfeld, J. Chem. Phys. **123**, 184101 (2005).

[16] D. Cremer and J. Gauss, J. Comput. Chem. **7**, 274 (1986).

[17] S. F. Boys, Proc. Roy. Soc. **A200**, 542 (1950).

[18] M. Hanrath, Quantum objects library (*QOL*), 2006-2012.

[19] P. Pulay, Chemical Physics Letters **73**, 393 (1980).

[20] S. Obara and A. Saika, J. Chem. Phys. **84**, 3963 (1986).

[21] S. Obara and A. Saika, J. Chem. Phys. **89**, 1540 (1988).

[22] A. Szabo and N. S. Ostlund, *Modern Quantum Chemistry*, Dover Publications, Mineola, NY, 1996.

[23] I. N. Levine, *Quantum Chemistry*, Pearson Prentice Hall, Upper Saddle River, NJ, 2009, 6. Edition.

[24] T. Helgaker, P. Jørgensen, and J. Olsen, *Molecular Electronic-Structure Theory*, John Wiley & Sons Ltd., Chichester, 2000.

[25] J. C. Slater, Phys. Rev. **34**, 1293 (1929).

[26] H. Eves, *Elementary Matrix Theory*, Dover Publications, New York, 1966.

[27] W. Pauli, Z. Physik **31**, 765 (1925).

[28] J. C. Slater, Phys. Rev. **34**, 1293 (1929).

[29] E. U. Condon, Phys. Rev. **36**, 1121 (1930).

[30] M. Dupuis, J. Rys, and H. F. King, J. Chem. Phys. **65**, 111 (1976).

[31] H. F. King and M. Dupuis, J. Comput. Phys. **21**, 141 (1976).

[32] L. E. McMurchie and E. R. Davidson, J. Comput. Phys. **26**, 218 (1978).

[33] P. M. W. Gill, Adv. Quantum Chem. **25**, 141 (1994).

[34] J. A. Pople and W. J. Hehre, J. Comput. Phys. **27**, 161 (1978).

[35] M. Head-Gordon and J. Pople, J. Chem. Phys. **89**, 5777 (1988).

[36] T. P. Hamilton and H. F. Schaefer, Chem. Phys. **150**, 163 (1991).

[37] R. Lindh, U. Ryu, and B. Liu, J. Chem. Phys. **95**, 5889 (1991).

[38] D. Strout and G. Scuseria, J. Chem. Phys. **102**, 8448 (1995).

[39] Intel® *Math Kernel Library* (11.0), 2012, https://software.intel.com/en-us/articles/intel-mkl-11dot0.

[40] M. Schütz, R. Lindh, and H.-J. Werner, Mol. Phys. **96**, 719 (1999).

[41] *BOOST C++ Libraries* (1.59.00), 2015, http://www.boost.org.

[42] D. Feller, J. Comput. Chem. **17**, 1571 (1996).

[43] K. L. Schuchardt et al., J. Chem. Inf. Model. **47**, 1045 (2007).

[44] F. Weigend and R. Ahlrichs, Phys. Chem. Chem. Phys. **7**, 3297 (2005).

[45] A. Schäfer, H. Horn, and R. Ahlrichs, J. Chem. Phys. **97**, 2571 (1992).

[46] A. Schäfer, C. Huber, and R. Ahlrichs, J. Chem. Phys. **100**, 5829 (1994).

[47] F. Weigend, F. Furche, and R. Ahlrichs, J. Chem. Phys. **119**, 12753 (2003).

[48] P.-O. Widmark, P.-Å. Malmqvist, and B. O. Roos, Theor. Chim. Acta. **77**, 291 (1990).

[49] P.-O. Widmark, B. J. Persson, and B. O. Roos, Theor. Chim. Acta. **79**, 419 (1991).

[50] B. O. Roos, V. Veryazov, and P.-O. Widmark, Theor. Chem. Acc. **111**, 345 (2004).

[51] T. Dunning Jr., J. Chem. Phys. **90**, 1007 (1989).

[52] A. Wilson, T. van Mourik, and T. Dunning Jr., J. Mol. Struct.: THEOCHEM **388**, 339 (1996).

[53] D. Woon and T. Dunning Jr., J. Chem. Phys. **98**, 1358 (1993).

[54] T. van Mourik and T. Dunning Jr., Int. J. Quantum Chem. **76**, 205 (2000).

[55] F. Weigend, A. Köhn, and C. Hättig, J. Chem. Phys. **116**, 3175 (2002).

[56] C. Hättig, Phys. Chem. Chem. Phys. **7**, 59 (2005).

[57] S. H. Vosko, L. Wilk, and M. Nusiar, Can. J. Phys. **58**, 1200 (1980).

[58] J. C. Slater, Phys. Rev. **81**, 385 (1951).

[59] A. D. Becke, Phys. Rev. A **38**, 3098 (1988).

[60] J. P. Perdew, Phys. Rev. B **33**, 8822 (1986).

[61] P. A. M. Dirac, Proc. Roy. Soc. **A123**, 717 (1929).

[62] K. Eichkorn, O. Treutler, H. Öhm, M. Häser, and R. Ahlrichs, Chem. Phys. Lett. **240**, 283 (1995).

[63] Ieee 754-2008: Standard for floating-point arithmetic, ieee standards association, 2008.

[64] S. Grimme, private communications, 2016.

Abbreviations and Acronyms

AO	Atomic Orbital
API	Application Programming Interface
BLAS	Basic Linear Algebra Subprograms
CPU	Central Processing Unit
DFT	Density Functional Theory
DIIS	Direct Inversion of the Iterative Subspace
EMSL	Environmental Molecular Science Laboratory
IEEE	Institue of Electrical and Electronics Engineers
LAPACK	Linear Algebra Package
MO	Molecular Orbital
OMP	Open Multi-Processing
OS	Obara-Saika
PCG	Primitive Cartesian Gaussian
QOL	Quantum Objects Library
RAM	Random-Access Memory
SCF	Self-Consistent Field
SIMD	Single Instruction Multiple Data
SSE	Streaming SIMD Extensions
XML	Extensible Markup Language

Danksagung

- Prof. Dr. Michael Dolg danke ich für die Möglichkeit diese Arbeit anfertigen zu können. Ihm gilt mein aufrichtiger Dank für das hervorragende Arbeitsumfeld und die gewährte Unterstützung, auch und im besonderen in schweren Zeiten.
- PD. Dr. Michael Hanrath danke ich für die Betreuung dieser Arbeit. Mit großer Hilfsbereitschaft und Geduld hatte er stets ein offenes Ohr für alle Fragen theoretischer und technischer Natur. In vielen Gesprächen diskutierte er mit mir die Feinheiten und Fallstricke der technischen Umsetzung wissenschaftlicher Probleme.
- Meinen Bürokollegen, jetzigen wie früheren, Oliver Mooßen, Nils Herrmann, Dr. Tim Hangele und Dr. Jan Ciupka danke ich für die angenehme Atmosphäre.
- Norah Heinz, Oliver Mooßen und Nils Herrmann danke ich für das sorgfältige Lesen dieser Arbeit und die konstruktiven Korrekturvorschläge.
- Allen Mitarbeitern und ehemaligen Mitarbeitern, M. Böhler, B. Börsch-Pulm, Dr. X. Cao-Dolg, S. Segieth, Dr. A. Weigand, Dr. D. Weißmann, Dr. K. Walczak, Dr. M. Hülsen, Dr. J. Wiebke, Dr. A. Engels-Putzka und Dr. J. Friedrich danke ich für die die gute Zeit und die vielen Weihnachtsfeiern, Grillabende und Ausflüge.
- Dr. Simon Renner, Ruth Bisterfeld, Andreas Mohr, Dr. Sebastian Peuker, Dr. Georgios Liaptsis und Daniel Röttger danke ich für Ihre langjährige Freundschaft.
- Mein besonderer Dank gilt meinen Eltern. Ohne Ihr Vertrauen wäre diese Arbeit nicht möglich gewesen.

Erklärung

Ich versichere, dass ich die von mir vorgelegte Dissertation selbständig angefertigt, die benutzten Quellen und Hilfsmittel vollständig angegeben und die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken im Wortlaut oder dem Sinn nach entnommen sind, in jedem Einzelfall als Entlehnung kenntlich gemacht habe; dass diese Dissertation noch keiner anderen Fakultät oder Universität zur Prüfung vorgelegen hat; dass sie – abgesehen von unten angegebenen Teilpublikationen – noch nicht veröffentlicht worden ist, sowie, dass ich eine solche Veröffentlichung vor Abschluss des Promotionsverfahrens nicht vornehmen werde. Die Bestimmungen der Promotionsordnung sind mir bekannt. Die von mir vorgelegte Dissertation ist von Prof. Dr. Michael Dolg betreut worden.

Ich versichere, dass ich alle Angaben wahrheitsgemäß nach bestem Wissen und Gewissen gemacht habe und ich verpflichte mich, jedmögliche, die obigen Angaben betreffende, Veränderung dem Dekanat unverzüglich mitzuteilen.

_____ (Datum)

_____ (Unterschrift)

Curriculum Vitae

Persönliche Daten

Name:	Joseph Held
Geboren am:	09.10.1981
Geburtsort:	München
	ledig

Schulbildung

1988–1992	Grundschule Geiselhöring
1992–2001	Johannes-Turmair-Gymnasium Straubing, Abschluss: Abitur

Studium

10/2001–02/2012	Diplom Chemie, Universität zu Köln
2/2012	Abschluss: Diplom Chemiker
8/2012–	Promotion am Institut für Theoretische Chemie der Universität zu Köln

Berufliche Erfahrungen

07/2004–07/2008	Studentische Hilfskraft am Zentrum für angewandte Informatik Köln - Rechenzentrum
09/2007–06/2010	Studentische Hilfskraft am Institut für Theoretische Chemie der Universität zu Köln