Advancing Linear-Scaling Techniques in Computation of Exchange Matrices in Mean-Field and Cluster-in-Molecule Calculations

by

Nan Song

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Science
in
Chemistry

Waterloo, Ontario, Canada, 2024

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Quantum chemistry faces ongoing challenges in developing methods that combine efficiency with accuracy, especially for large molecular systems. The Cluster-in-Molecule (CiM) technique, integrated with Coupled Cluster theory (CC) methods, offers a promising solution by accurately computing correlated ground state energies through division into computations of small subsystems. These systems utilize a subset of localized natural orbitals (LNO) defined by localized orbital domains [1, 2, 3, 4, 5, 6]. The advantage of CiM-CC approach is that all subsystem calculations can trivially be computed in parallel with a relatively straightforward algorithm. The main challenges are in defining small orbital domains in accurate and efficient ways, and the required integral transformation from the global atomic orbitals (AO) basis to the subset of LNO.

In this work, we enhance the efficiency of calculating two electron repulsion integral (ERI) through advanced computational techniques that incorporate the Resolution of Identity (RI) metric matrix and a three-index short-range Coulomb potential with Gaussian-Type Geminal (GTG) correction. This aspect of the research, inspired by the thesis work of Dr.Michael J. Lecours in the Nooijen group, focuses on improving the efficiency of calculating the exchange matrices $\mathbf{K}$ while maintaining acceptable error margins [7, 8]. Our newly developed algorithms in the Python module for quantum chemistry platform (PySCF) program, especially for calculating Coulomb $\mathbf{J}$ and exchange $\mathbf{K}$ matrices through the JK-Engine, are shown to achieve a linear correlation between performance and the size of molecular systems. These improvements are not only vital for CiM but also for Hartree-Fock (HF) and (hybrid) Density Functional Theory (DFT) mean-field calculations, with accuracy controlled by a single parameter defining the short-range Coulomb potential's range.

Utilizing the exchange matrix, we present an efficient orbital domain construction scheme for occupied localized molecular orbitals (LMO) based on the pivoted Cholesky decomposition of the exchange matrix. This method improves the efficiency of the partitioning into LMO subspaces, crucial for CiM calculations.

In summary, our advancements in linear-scaling exchange matrix calculations and orbital domain construction mark significant progress toward more efficient and accurate electronic structure calculations for mean-field and CiM approaches, promising enhanced computational performance for large molecular systems.

Acknowledgements

I would like to extend my deepest gratitude to my supervisor, Dr.Marcel Nooijen. We had many bursts of ideas in our academic discussions every week, where we delved into deriving equations and addressing complex problems. His expertise and guidance have been instrumental in my development and success during my undergrad and master's journey. His encouragement and confidence in my ability have not only fostered my development in theoretical chemistry but also opened doors for me to engage in significant events like the Chemical Physics Symposium and the Canadian Quantum Cup.

Additionally, I appreciate the efforts and comments by committee Dr. Pierre-Nicholas Roy, and Dr. Scott Hopkins. At the same time, I am grateful to my early-stage master's group members, Dr. Ondrej Demel, Dr. Michael James Lecours and Haobo Liu. Also, thanks for all my theoretical chemistry group members and friends, Songhao Bao, Benny Chen, Azharuddin Alfaz Sarfraz Mohammed, and Wenxue Zhang.

My deepest appreciation also goes to my parents, whose unwavering support and love have been my constant source of strength.

Lastly, I express gratitude towards OpenAI's ChatGPT for its assistance in my writing process, particularly for debugging code and checking grammar [9].

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AO atomic orbitals iii, xx, 6, 9, 10, 13, 14, 19, 35–37, 43, 44, 46, 47, 78, 80, 81, 97, 98, 101, 102, 105, 106

API Application Programming Interface 21

CC Coupled Cluster theory iii, 1, 2, 4–6

CCSD Coupled Cluster Single Doubles 2, 4

CCSD(T) Coupled Cluster Single Double and Perturbative Triple method 2, 4

CI Full Configuration Interaction 1

CiM Cluster-in-Molecule iii, 2–6, 12, 33, 35, 77, 78, 85, 86

COD Construction of Orbital Domains xv, 77–82, 85

COO Coordinate List format ix, 36, 39–42, 61, 96, 97

COSX chain-of-spheres exchange 7

CPU Central Processing Unit 5, 46, 50, 55, 56, 58, 61, 63, 85

CSC Compressed Sparse Column format x, 39–41

CSR Compressed Sparse Row format ix, 39, 40

DC divide-and-conquer 2, 4

DF Density Fitting v, 6–8, 15–17, 19, 21–23, 27, 34, 53–56, 63, 66

DFT Density Functional Theory iii, 1, 6, 15, 86

DM density matrix 46, 47

ERI electron repulsion integral iii, 2, 6–8, 12–16, 21, 28, 77

FT Fourier transformation 10

GTG Gaussian-Type Geminal iii, 16, 27, 28, 32

GTO Gaussian-Type Orbitals 13, 19, 21

HDD Hard Disk Drive 42, 46, 47, 86

HF Hartree-Fock iii, 1, 4, 6, 7, 11, 12, 14, 15, 21, 34, 42, 43, 47, 50, 61, 77, 86

LMO occupied localized molecular orbitals iii, x–xiii, 3, 5, 6, 12, 35, 43, 44, 46, 51, 53, 57–62, 65, 68–70, 73, 75, 77–79, 81, 101, 102, 105

LNO localized natural orbitals iii, 3–5

MBPT Many-Body Perturbation Theory 1

MO molecular orbitals 14

MP2 Second-order Møller-Plesset perturbation theory 2, 5, 6, 9, 11

NNZ the number of non-zero elements xii, xiii, 23, 61, 69, 71–76

OOM Out-of-Memory 35, 69

OOP Object-Oriented Programming 33

OSV orbital specific virtual orbital 2, 4

PAO projected atomic orbital 2, 4

PNO pair natural orbital 2, 4

PS pseudospectral 7

PySCF Python module for quantum chemistry platform iii, 12, 21–23, 50, 54–56, 63, 76, 85

RAM random-access memory 5, 37, 46, 69, 75, 85

RI Resolution of Identity iii, 6–8, 11, 15, 17–19, 22, 28, 30, 31, 34, 35, 42, 43, 47, 53, 61, 85, 86

SCF Self-Consistent Field 34, 50

THC tensor hypercontraction 7, 8

List of Symbols

J Coulomb Integrals iii, 104

K Exchange Integrals iii, 35, 102

$\mathbf{n_{ao}}$ the number of AO 6, 13, 14, 19, 36, 97

$\mathbf{n_{naux}}$ the number of auxiliary orbitals 36, 97

$\mathbf{n_{gatm}}$ the number of grouped atoms 21, 43, 71

$\mathbf{n_{heavy}}$ the number of heavy atoms in a segment ix, 37, 43

$\mathbf{n_{ip}}$ the number of interpolation vectors 8

$\mathbf{n_{slice}}$ the number of total slices 38

Chapter 1

Introduction

1.1 Motivations

Nowadays, one of the critical concerns in quantum chemistry is to solve electronic problems, such as electronic energy and molecular properties. With the rapid development of computers, several approaches, including HF Theory, DFT, and Coupled-Cluster (CC) Theory, have been implemented to solve electronic systems. To start with, HF Theory is the most straightforward approach, which only considers a single Slater determinant and neglects the interactions of correlated motion of electrons, such as Coulomb correlation and near-degeneracy correlation [24]. Although the HF theory reaches perhaps 99% of exact energy, the remaining 1% error from approximation is crucial for applications in quantum chemistry [24], which is primarily concerned with energy differences. Hence, the post-HF methods are mainly based on HF theory and made some improvements to solve these restrictions.

When performing different wave-function based post-HF approaches, the value of correlation energy is treated as a measurement of accuracy. The correlation energy (E_{corr}) considers the difference between the exact non-relativistic energy (ξ_0) and HF energy (E_{HF}) with limitation[24, 25]:

$$E_{corr} = \xi_0 - E_{HF} \tag{1.1}$$

The value of correlation energy is always negative, as E_{HF} from variational optimization is an upper limit of exact energy. There are mainly three types of wave-function based post-HF methods considering electronic correlation: Full Configuration Interaction (CI), Many-Body Perturbation Theory (MBPT), and CC Theory [24, 25, 26, 27].

Compared to other methods, the CC theory not only considers the effects of electron correlation in many-electron systems to provide precise solutions of the Schrödinger equation but also calculates cluster wavefunctions and electronic energies efficiently by applying exponential operators [27]. Although the CC theory makes the description of electronic structure for small- and medium-sized systems feasible and controllable by employing computer techniques, the analysis of electronic correlation in large molecule systems regarding

high accuracy is still a big challenge nowadays [26]. The most important reason is the high dependence between computational cost and system size.

The remainder of this chapter consists of four sections that cover the theoretical framework of the local correlation method, a review of previous work on the cluster-in-molecule (CiM) approach, an overview of computation methods for two-electron repulsion integrals (ERIs), and a discussion on representation of two-electron integrals developed by the Nooijen group.

1.2 Overview of Local Correlation Methods

In order to make wave function-based *ab initio* calculations feasible for large molecules, a variety of local electronic correlation methods have been introduced. Currently, local correlation methods are available for both ground and also excited states. In applications to ground state energies, the focus is on the local nature of the correlation energy. A key feature of local correlation approaches is to approximate the correlation energy by eliminating distant interactions between localized orbitals [28, 29, 30, 1, 2, 3, 4, 14, 19, 18, 15, 16, 20, 5].

Generally, according to previous studies, the local correlation methods within CC may be classified into two main groups. On the one hand, the class of "direct" local approaches allows the application of local approximation to the canonical equations. Typically, this group incorporates the projected atomic orbital (PAO) method and the pair natural orbital (PNO) method [28, 29, 30]. On the other hand, the crux of the class of "fragment-based" local approaches is to partition the overall calculation of a large system into multiple calculations on smaller subsystems. In these approaches, the overall correlation energy is the summation of each energy contribution of a fragment. The difference between fragment-based approaches is the criteria for splitting molecules into subsystems, like atoms, bonding orbitals, electrons, and electron pairs. This category mainly consists of the divide-and-conquer (DC) method introduced and the CiM approach [2, 3, 4, 5, 1, 14, 15, 16, 18, 19, 20, 6]. **Table 1.1** summarizes the critical studies of local electron correlations.

Neese and co-workers developed a PNO approach based on Coupled Cluster Single Doubles (CCSD) and Coupled Cluster Single Double and Perturbative Triple method (CCSD(T)) to local correlation [29, 30]. In the PNO approach each pair of localized occupied orbitals is related to its own small set of virtual orbitals, the so-called PNOs, and this greatly reduces the number of excitations. However, the final calculations involve all excitations in a global calculation on the full system, and the association of different virtual orbitals with each pair of occupied orbitals greatly complicates the implementation. In 2011, Yang and colleagues have introduced a method based on orbital specific virtual orbital (OSV) for local Second-order Møller-Plesset perturbation theory (MP2) and CCSD calculations [21, 22]. Furthermore, in these local correlation methods, the virtual space can be described using PNOs or OSVs, as alternatives to PAOs. It has been observed that although PNOs domain sizes are not smaller than those of PAOs or OSVs by design, significantly smaller number of PNOs can achieve comparable accuracy [29, 30, 21, 22].

Since 2002, when Li's group initially introduced the CiM approach to tackling the computational complexity of large systems using a basis of LMOs, there have been significant advancements in the method [31]. The CiM method has been further refined and enhanced by Piecuch's group [2, 3]. Later in Kállay's group, a highly efficient implementation uses LNO based on the CiM approach in which extensive molecular calculation allows partition into a subset of small calculations of individual electrons, not electron pairs [4, 5]. It is difficult and relatively expensive to select occupied orbital domains or LMO in the CiM approach. The next step is to construct a restricted complementary virtual space, strongly interacting with selected localized orbital domains. Then, the calculation of all localized occupied orbital domains would be carried in independent calculations. Thus, the computational cost for the complete system will be simply the sum of the cost of the more minor calculations. The bottlenecks in the CiM calculations lie in obtaining the subspaces of orbitals for each smaller calculation, and the transformation of AO integrals to the smaller basis of selected molecular orbitals. This cost does not scale linearly with the size of the complete system in large systems. In general, the CiM approach provides efficient computational reduction by orders of magnitude to analyze large molecular systems, but the bottlenecks in the calculation can be improved still. These bottlenecks are the starting point for this investigation.

Authors	Methods	Key Contributions	Limitations
Pulay and Saebo	PAO method	Construct virtual space by PAO onto complementary occupied space. Apply to coupled electron pair approximation.	The computational ability for molecules did not allow implementing during that time.
Werner, Schutz, and co-workers	Extend in CCSD and CCSD(T)	Computational cost is linear scaling to the system size	Approximation of domain of orbital pairs limits the accuracy.
Neese, and co-workers	PNOs	Interacting domains are defined for electron pairs. It can connect with large and flexible basis sets. "Black-box" format	Complicated implementation. Most recent DPLNO uses PAO in efficient implementation.
Yang, and co-workers	OSV	Apply tensor factorization. Save computational time and have controllable errors by adding a threshold	Inefficient with small thresholds due to large number of virtual orbitals.
Flocke and Bartlett	Natural linear scaling CC approach	Localized natural bond orbitals are applied, other than the localized occupied HF orbitals. Treat extended nonperiodic systems of infinite basis set size.	The choice of each subsystem is mainly based on chemical intuition, not mathematics.
Li and Li	The DC CC	Separate molecules into subspaces by their local environments.	Construct subsystem manually. The accuracy is based on choices for separating.
Li and Piecuch.	CiM approach	Interacting spaces are constructed from each electron contribution with a parallelism of calculation and linear scaling.	Application and calculations in a large molecular system are still challengeable.
Kállay	LNO CiM approach	Apply local natural orbitals in the CiM calculations	Calculations in a large molecular system are still expensive.

Table 1.1: Summary of critical studies to local correlation methods [1, 3, 2, 4, 14, 15, 16, 17, 18, 19, 20, 5, 6, 21, 22, 23]

1.3 The CiM Approach in Nooijen Group

Although CC provides the accuracy of correlation interactions for calculation in many-body systems, it is only suitable for small and medium-sized molecules and will reach its limits in extensive molecular orbital systems. Several local correlation methods have been developed based on the CiM approach proposed by Li's group to tackle these defects. The basic idea underlying the CiM approach is to break up extensive molecular calculations into many small pieces of each LMO calculations and constructs virtual space under LMO domains. The energy can be derived as the sum of the contributions of individual LMO [1, 2, 3]. By contrast, Kállay and co-workers utilized LNOs as a means to split the molecular system in the CiM framework, then obtained occupied and virtual LMO subspaces from the density matrix obtained from an MP2 calculation [4, 5, 6]. **Figure 1.1** illustrates two various orbital domains P and Q in a molecule in CiM approach [10]. The black orbital in P (or Q) represents the central LMOs, and surrounding blue orbitals represent interacting LMOs domains that need to be calculated in a fragment. Also, the gray part in the molecule has weak interactions that can be ignored in the CC calculation. Some of the remaining correlation can be treated using second order perturbation theory MP2

The advantage of the CiM approach reveals that each subsystem can be evaluated by regular CC calculations. Moreover, the CiM calculations manifest the parallel calculations of each orbital domain to reduce the computational resources, including Central Processing Unit (CPU) time and random-access memory (RAM).

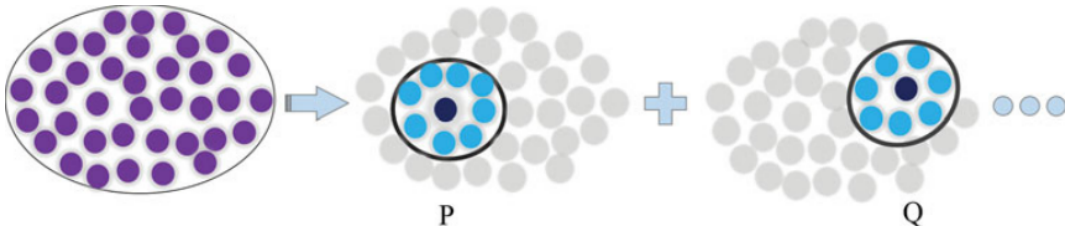


Figure 1.1: Illustration of Cluster-in-Molecule Approach [10]

The electronic correlation energy for closed shell molecules based on CiM approach: [23]

$$\begin{aligned} E_{corr} &= \sum_{i,j,a,b} (ia|jb)\tau_{ab}^{ij} \\ &= \sum_{I} \sum_{i\in I} \sum_{j,a,b} (ia|jb)\tau_{ab}^{ij} \equiv \sum_{I} \sum_{i\in I} E_i \end{aligned} \tag{1.2}$$

where the notations of i, j represent localized occupied orbitals, and the notations of a, b represent localized virtual orbitals. The quantity $(ia|jb)$ is antisymmetrized two-electron exchange integral, and only considers spatial coordinates. The sum over i is partitioned in a number of subsystems center I. Besides, the excitation amplitudes τ_{ab}^{ij} are different in

MP2 and CC calculations:

$$\tau(CC)_{ab}^{ij} = t_{ij}^{ab} + t_a^i t_b^j - t_a^j t_b^i \tag{1.3}$$

$$\tau(MP2)_{ab}^{ij} = t_{ij}^{ab} \tag{1.4}$$

There are two challenging difficulties for performing electronic energy of CiM. The first difficulty of the CiM approach is efficiently partitioning systems into LMO subspaces. The division criteria will impact the computational timing and accuracy of the exact energy. Also, another significant issue is to transform from the AO based basis to the orbital domain.

In the work of Kállay the LMOs are defined based on the MP2 one-particle density matrix. This approach does not scale well with the size of the system. In preliminary work in the Nooijen group the orbital selection is based on the calculation of the exchange matrix, using various selected orbitals to define this quantity for different orbital subspaces (to be discussed in the final chapter of this thesis). While this is a fundamental improvement in scaling compared to MP2 one-particle density matrix, the calculation of exchanges matrices is expensive even today in Hartree-Fock and Hybrid DFT calculations. For example, in solid state calculations hybrid functionals are rarely used, because of their great expense. The exchange matrix plays a central role in this research and in the next section, an overview is presented of work done in the literature. In the following section the work done in the Nooijen group is discussed and this provides a point of departure for the work in this thesis. This introductory chapter will be completed with an outline of the thesis.

1.4 Overview of Computational Methods for Two-Electron Repulsion Integrals

In electronic structure calculations, the calculation of two-electron repulsion integrals (ERIs), especially during the computation of exchange (K) and Coulomb (J) matrices, is a fundamental yet computationally demanding task. The accurate two-electron integrals can be expressed by:

$$(\alpha\beta|\gamma\delta) = \int d^3r_1 d^3r_2 \xi_\alpha(\mathbf{r}_1)\xi_\beta(\mathbf{r}_1)\frac{1}{r_{12}}\xi_\gamma(\mathbf{r}_2)\xi_\delta(\mathbf{r}_2), \tag{1.5}$$

where $(\alpha\beta|\gamma\delta)$ denotes the integral of the product of four atomic orbital functions ξ. The implementation of these integrals is expensive, scaling with $O(N^4)$, where N represents the number of atomic orbital $\mathbf{n_{ao}}$. This leads to the computational burden, especially in HF, DFT and post-HF.

To mitigate this, several approaches have been introduced. Among them, the the density fitting (DF) technique, also called resolution of identity (RI) is prevalent for its efficiency in approximating electron density via an auxiliary basis set, thereby streamlining the manipulation of the ERIs tensor [18, 32, 33]. The approximation is formulated as

follows:

$$\xi_\alpha(\mathbf{r})\xi_\beta(\mathbf{r}) \approx \sum_x C_x^{\alpha\beta}\phi_x(\mathbf{r}) \tag{1.6}$$

$$(\alpha\beta|\gamma\delta) \approx \sum_{xy}(\alpha\beta|x)(x|y)^{-1}(y|\gamma\delta) \tag{1.7}$$

where $C_\mu^{\alpha\beta}$ are the expansion coefficients for the auxiliary basis functions ϕ_x that approximate the product of the atomic orbitals α and β. Consequently, four-center ERIs are approximated by three-center integrals and the two-center Coulomb integral, reducing computational complexity from $O(n^4)$ to $O(n^3)$ for molecules. Similarly, the Cholesky Decomposition of two-electron integrals, proposed by Beebe and Linderberg, offers a comparable level of efficiency and compactness to RI factorization [34]. The construction of exchange based on RI:

$$K_{\alpha\beta}^{RI} = -\frac{1}{2}\sum_{ixy}(\alpha i|x)(x|y)^{-1}(y|i\beta) \tag{1.8}$$

where the RI-K algorithm grows in quartic proportion $n_{ao}^2 n_{occ} n_{aux}$ to the size of the system.

Adopting an alternative approach, Friesner utilized a pseudospectral (PS) method to approximate ERIs tensor [35, 36, 37]. The PS approach involves constructing a secondary basis set defined by points in three-dimensional space,

$$(\alpha\beta|\gamma\delta) \approx \sum_g \gamma(r_g)\omega_\alpha(r_g)\omega_\beta(r_g)V_{\gamma\delta}(r_g) \tag{1.9}$$

$$V_{\gamma\delta}(r_g) = \int dr \frac{\omega_\gamma(r)\omega_\delta(r)}{|r - r_g|} \tag{1.10}$$

where $\{r_g\}$ is a set of grid points in three-dimensional space and $\gamma(r_g)$ is associated weight of grids. The advantage of PS is its circumvention of the Coulomb singularity, ensuring numerical stability in the algorithm. This straightforward PS strategy is effectively implemented in HF theory, resulting in a cubic scaling algorithm. This represents a lower scaling compared to the traditional quartic scaling of RI-based HF theory. However, a significant limitation of PS methods is their limited applicability. To solve this limitation, Neese and co-workers combined semi-numerical integration techniques and DF techniques for HF exchange part, including Friesner's PS method and linear-scaling chain-of-spheres exchange (COSX), both of which demonstrate excellent scalability with the basis set's highest angular momentum [38].

Another innovative approach is tensor hypercontraction (THC), which decomposes the four-index tensor into a series of matrix multiplications, enhancing flexibility and lowering storage demands from the cubic level in conventional RI method to quadratic level [39, 40, 41, 42].

$$(\alpha\beta|\gamma\delta) \approx \sum_{KL}\omega_\alpha(r_K)\omega_\beta(r_K)M_{KL}\omega_\gamma(r_L)\omega_\delta(r_L) \tag{1.11}$$

where $\{r_K\}$ denotes a chosen set of grid points. Head-Gordon's group combined an ad-

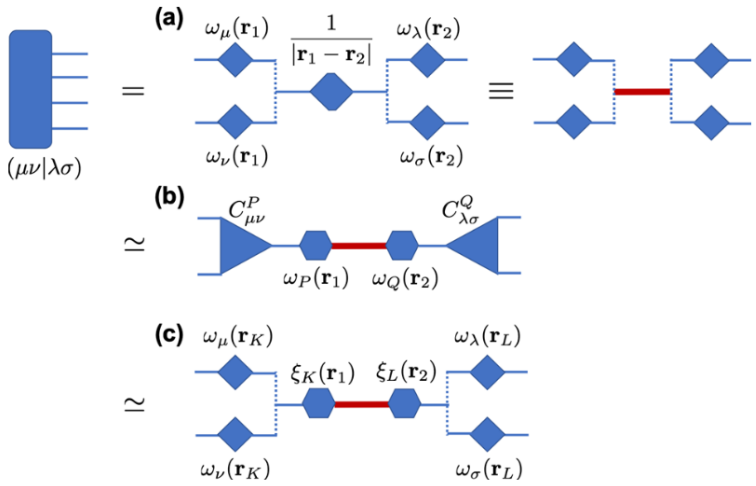


Figure 1.2: Visual Representation of the Density Tensor: (a) its exact representation, (b) its representation via RI, and (c) its representation through THC. Solid blue lines denote contractions over common indices between two tensors, whereas dotted blue lines signify an element-wise multiplication across linked indices. The bold red bar illustrates a contraction with the Coulomb operator $\frac{1}{r_2-r_1}$. [11]

vanced tensor hypercontraction (THC) factorization approach with interpolative separable DF for selecting grid points [11, 41]. Here, $\{r_K\}$ are used to approximate the function via an interpolation vector, $\xi_K(r)$ [43]. The number of interpolation vectors($\mathbf{n_{ip}}$) influences the accuracy and efficiency — large $\mathbf{n_{ip}}$ enhances accuracy, while small $\mathbf{n_{ip}}$ improves efficiency. Furthermore, **Figure 1.2** visualizes the representation of exact ERIs, RI ERIs, and THC-RI ERIs respectively.

The exchange matrix based on THC-RI method can be given as:

$$K^{THC}_{\alpha\beta} = -\frac{1}{2}\sum_{iKL}\omega^K_\alpha \phi^K_i M_{KL}\phi^L_i \omega^L_\beta \tag{1.12}$$

where M_{KL} is Coulomb integral involving interpolation vectors. The THC-RI-K algorithm grows in cubic proportion $n_{ao}n^2_{ip}$ to the size of the system.

Inspired by previous work on exchange matrix, our group has employed range-separated Coulomb potential to represent DF two-electron integrals. Next, we will review previous accurate two-electron integral representations by our group.

1.5 Overview of Accurate Representations of Two-Electron Integrals Developed Previously in the Nooijen Group

The calculation of the so-called two-electron integrals

$$(\alpha\beta|\gamma\delta) = \int d^3r_1 d^3r_2 \xi_\alpha(\mathbf{r}_1)\xi_\beta(\mathbf{r}_1)\frac{1}{r_{12}}\xi_\gamma(\mathbf{r}_2)\xi_\delta(\mathbf{r}_2), \tag{1.13}$$

in which ξ_α represent atomic orbitals AO's is a crucial ingredient of electronic structure calculations. The key to efficient computations in quantum chemistry is to avoid calculating the two-electron integrals directly, but find other representations of this interaction. Dr.Michael J. Lecours wrote a PhD thesis on this subject while Dr.Ondrej Demel in collaboration with Dr.Lecours developed a Laplace MP2 approach based on this representation of integrals [8, 7]. We will first represent this representation of integrals to put the present work in perspective.

The Coulomb interaction is partitioned into short- and long-range parts based on a modified Ewald partitioning. The two-body Coulomb operator $\frac{1}{r_{12}}$, is written as

$$\frac{1}{r_{12}} = V_{sr}(r_{12}) + V_{lr}(r_{12}). \tag{1.14}$$

Here, the short-range part V_{sr} itself consists of two components: the complementary error function $erfc(r_{12})$, which is commonly used for this purpose, and a Gaussian-like term

$$V_{sr}(r_{12}) = \frac{\mathrm{erfc}\,(\alpha r_{12})}{r_{12}} + \frac{2\alpha}{\sqrt{\pi}}e^{-\frac{\alpha^2}{3}r_{12}^2}. \tag{1.15}$$

The long-range part is then

$$V_{lr}(r_{12}) = \frac{1}{r_{12}} - V_{sr}(r_{12}). \tag{1.16}$$

The exponent and prefactor in the Gaussian term in Equation 1.15 is chosen in such a way that $V_{sr}(r)$ and its second derivative have a zero limit for $r \to 0$. We note that $V_{sr}(r)$ is fully parameterized by the choice of α.

Figure 1.3 illustrates the Coulomb integral is divided into two components, short-range and long-range. In detail, the short-range component decays rapidly with distance and is close to the full Coulomb potential at short range, whereas the long-range component retains the $1/r_{12}$ decay at large r_{12}.

The short-range part of the Hamiltonian is treated using density fitting techniques in standard ways. The respective components of 4-index integrals can be evaluated as

$$(\alpha\beta|\gamma\delta)_{sr} = \sum_x (\alpha\beta|x)_{sr}(\gamma\delta|x)_{sr} \equiv \sum_x (\alpha\beta|x)_{sr}(x|\gamma\delta)_{sr}. \tag{1.17}$$

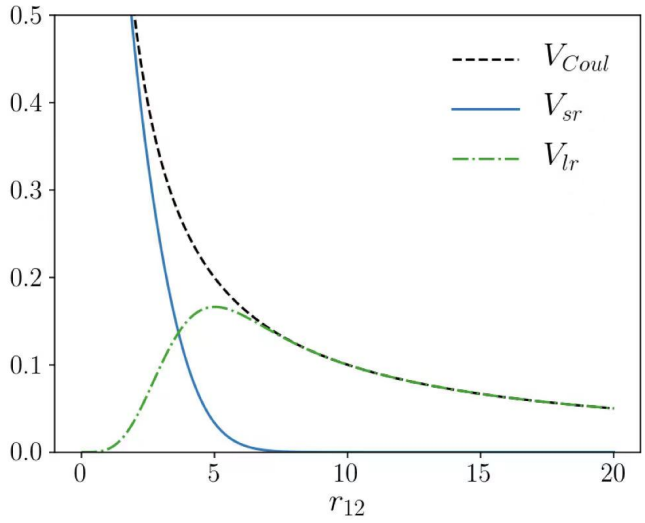


Figure 1.3: The Range-Partitioned Coulomb Potential [7]

Here we have introduced an (orthonormalized) auxiliary basis set x. The key feature is that the number of such short-range 3-center two-electron integrals grows linearly with the size of the system.

The long-range contribution is evaluated numerically using a Fourier transformation (FT) in k-space

$$V_{lr}(r_{12}) = \int d^3g \; V_{lr}(g) \; exp(i\mathbf{g} \cdot (\mathbf{r}_2 - \mathbf{r}_1)). \tag{1.18}$$

This FT is known analytically. The corresponding long-range component to the 4-center integrals is then evaluated as

$$(\alpha\beta|\gamma\delta)_{LR} = \sum_g (\alpha\beta|g)\sigma_g(g|\gamma\delta), \tag{1.19}$$

where g represents a plane wave index. The σ_g parameter is a sign factor corresponding to the sign of the $V_{lr}(g)$ contribution. In this formulation the 3-index quantities $(\alpha\beta|g)$ are calculated efficiently using analytical formulas at a set of grid points g, which are defined using a numerical integration in spherical coordinates. This Jacobian of spherical coordinates cancels the g^{-2} singularity, and accurate results are obtained for small Fourier grids of about 3000 points. It is surprising that the spherical integration grid has not been used before in the literature. It is (mostly) accurate and can be used even for compact core orbitals, which are treated differently in the usual plane wave codes [8].

Unfortunately, the resulting long-range 4-center integrals lose accuracy when the AO pairs $\alpha\beta$ and $\gamma\delta$ are separated by a long distance (e.g. 15 a.u.). Individual contributions are small but they are responsible for long-range forces. Even more, it is deemed important to correct for the error, as the FT approach may not yield small values due to rapid oscillations of the integrant.

For this reason, a third layer of approximation was introduced, which involves a multipole expansion in Cartesian coordinates. Let V be a general radial potential. The position vector $\mathbf{r}$ is expressed as $\mathbf{r} = \mathbf{A} + \mathbf{R}$. The Taylor expansion of $V(\mathbf{r_2} - \mathbf{r_1})$ around the point $\mathbf{A} = \mathbf{B} = 0$ is made using the substitution

$$\mathbf{r_1} = \mathbf{A} + \mathbf{R_1} \tag{1.20}$$
$$\mathbf{r_2} = \mathbf{B} + \mathbf{R_2} \tag{1.21}$$
$$\mathbf{R} = \mathbf{R_2} - \mathbf{R_1}, \tag{1.22}$$

and takes the general form

$$V(\mathbf{B} - \mathbf{A} + \mathbf{R}) = \sum_{K,L} m_B^K f^{K,L}(\mathbf{R}) m_A^L, \tag{1.23}$$

where m_B^K and m_A^L are polynomials in components of vectors $\mathbf{A}$ and $\mathbf{B}$, $f^{K,L}(\mathbf{R})$ consists of (analytically known) derivatives of potential and components of $\mathbf{R}$ vector, and indices K, L correspond to multipole expansion. For example, the Taylor expansion up to third order following

$K = 0$ corresponds zeroth-order term of the expansion

$K = 1, 2, 3$ corresponds to first-order term

$K = 4, \ldots, 12$ correspond to second order term (nine combinations)

$K = 13, \ldots, 39$ correspond to third order term (twenty seven combinations)

In both HF and Laplace MP2 calculations all three contributions to the Coulomb potential are judiciously implemented in various steps of the calculation. In previous publications, it was shown that this overall scheme is indeed accurate [8, 7, 44]. There are very significant drawbacks, however. First of all a careful selection has to be made when to switch between Fourier and multipole expansions. More importantly, implementations become very complicated, even when they exhibit linear scaling as a result. Finally, due to the complexity, it is hard to write fully optimized code, and the current versions are not very efficient compared to other approaches in the literature.

In this thesis, we take another look at the representation of integrals using RI. We will use the same short-range component (using even shorter-range integrals) as in our previous work, but we introduce a different metric matrix that will allow us to represent the full two-electron integrals (i.e. not just the short-range component), using a metric matrix as

$$(\alpha\beta|\gamma\delta)_{fr} = \sum_{x,y} (\alpha\beta|x)_{sr} M_{xy} (\gamma\delta|y)_{sr} \equiv \sum_{x,y} (\alpha\beta|x)_{sr} M_{xy} (y|\gamma\delta)_{sr}. \tag{1.24}$$

.

At this point, we are ready to discuss the final section in this introductory chapter.

1.6 Outlook of the Thesis

This thesis provides a detailed investigation into the efficiency and scalability of the foundation layer of the CiM approach, with a particular focus on the implementation of Coulomb and exchange matrices. Here is an outlook summarizing the key contributions of each chapter:

The following Chapter 2 introduces novel computational strategies for the efficient computation of two-electron repulsion integrals (ERIs), as implemented in the Int-Class package. By combining density fitting with approximations for short-range potentials, this new representation of ERIs provides a solid groundwork for JK-Engine, achieving a reduction in computational time without compromising accuracy. Additionally, it contains adjustments of several parameters to preserve the accuracy of HF calculations.

Chapter 3 delves into the core development of the algorithm (JK-Engine) for calculating the Coulomb and exchange matrices, a crucial input component in the CiM approach. Utilizing the sparsity of short-range integrals, this algorithm shows linear scaling efficiency. Building on this, Chapter 4 demonstrates a comprehensive analysis of the linear-scaling calculation for the exchange matrix within both timing and memory consumption, compared with PySCF. It underlines the applicability and scalability of our JK-Engine in large systems.

Chapter 5 addresses the challenge of partitioning molecular systems into localized molecular orbital (LMO) subspaces. An algorithm based on pivoted Cholesky decomposition of exchange matrices is introduced, enabling the efficient and accurate construction of orbital domains. This development is crucial for the practical implementation of the CiM approach in extensive simulations.

Chapter 2

Efficient Three-index Two-electron Integrals

The starting point for electronic structure calculations is the specification of an initial molecular structure and the definition of a basis set. In this work we use real atomic orbitals (AO) centered on the nuclei (or other meaningful centers) that consist of Gaussian type orbitals Gaussian-Type Orbitals (GTO). The AO basis functions will always be labeled by Greek indices $\alpha, \beta, \gamma, \delta$. Users may specify a particular AO basis set upon input, such as 6-31G(d,p) or cc-PVTZ, which are commonly used basis sets.

The fundamental building blocks that define the Hamiltonian operator consist of one- and two-electron integrals over AOs , utilizing atomic orbitals, $\xi_\alpha(\mathbf{r})$. These basis functions are non-orthogonal and this is reflected in the overlap integrals

$$S_{\alpha\beta} = \int d\mathbf{r}\ \xi_\alpha(\mathbf{r})\xi_\beta(\mathbf{r}) \tag{2.1}$$

The one-electron Hamiltonian integrals in the AO basis are defined as:

$$h_{\alpha\beta} = \langle\alpha|\hat{h}|\beta\rangle = \int d\mathbf{r}\ \xi_\alpha(\mathbf{r})\hat{h}(\mathbf{r})\xi_\beta(\mathbf{r}) \tag{2.2}$$

$$\hat{h} = -\frac{1}{2}\nabla^2 - \sum_A \frac{Z_A}{|\mathbf{r} - \mathbf{R}_A|} \tag{2.3}$$

where Z_A and $\mathbf{R}_A$ label the charges and positions of the nuclei, respectively. We will use atomic units throughout, as is common in quantum chemistry. The one-electron integral includes the sum of the kinetic energy operator for electrons and the nuclear-electron attraction potential. These integrals are computationally inexpensive, with $O(n^2)$ for a size $\mathbf{n_{ao}}$ basis set. For large molecules there is additional sparsity and the integrals decay in a linear fashion in the limit, as does the overlap matrix.

The most cumbersome building block in electronic structure calculations are the two-electron repulsion integrals ERIs over spatial atomic orbitals, which in so-called chemist's

notation, are defined as,

$$(\alpha\beta|\gamma\delta) = \int d^3r_1 d^3r_2 \xi_\alpha(\mathbf{r}_1)\xi_\beta(\mathbf{r}_1)\frac{1}{r_{12}}\xi_\gamma(\mathbf{r}_2)\xi_\delta(\mathbf{r}_2) \tag{2.4}$$

For small molecules, there are $O(n^4)$ of these ERIs where n=$\mathbf{n_{ao}}$. For large molecules the centers of AOs (α, β) have to be relatively close, for the integral to be non-zero, as do the centers of AOs γ, δ. The number of non-zero ERIs then scales as $O(n^2)$, with a large prefactor.

In Hartree-Fock (HF) theory, the wave function is approximated as a single determinant of molecular orbitals (MO)s that are linear combinations of AOs:

$$\psi_i(\mathbf{r}) = \sum_\alpha \xi_\alpha(\mathbf{r})C_{\alpha,i}, \tag{2.5}$$

The MO coefficients $C_{\alpha,i}$ are to be determined such that the energy of the single determinant is minimized. The MO coefficients define the one-particle reduced density matrix in the AO basis

$$D_{\alpha\beta} = \sum_{i\in occ} C_{\alpha,i}C_{\beta,i}, \tag{2.6}$$

which characterizes the single determinant HF state completely. In the self-consistent field procedure (SCF) to solve for the optimal MOs, the Fock matrix $\mathbf{F}$ is constructed that includes the effective potential felt by an electron due to both nuclei and the other electrons' average repulsion. For a closed shell system the elements of the Fock matrix in the AO basis are given by:

$$F_{\alpha\beta} = h_{\alpha\beta} + \sum_{\gamma\delta} D_{\gamma\delta}\left((\alpha\beta|\delta\gamma) - \frac{1}{2}(\alpha\gamma|\beta\delta)\right), \tag{2.7}$$

The contributions to the Fock matrix present different levels of difficulty in actual efficient HF calculations. We distinguish the (direct) Coulomb matrix

$$J_{\alpha\beta} = \sum_{\gamma\delta}(\alpha\beta|\delta\gamma)D_{\gamma\delta} \tag{2.8}$$

and the so-called exchange matrix

$$K_{\alpha\beta} = \sum_{\gamma\delta}(\alpha\gamma|\beta\delta)D_{\gamma\delta}, \tag{2.9}$$

and we obtain

$$\mathbf{F} = \mathbf{h} + \mathbf{J} - \frac{1}{2}\mathbf{K} \tag{2.10}$$

The Coulomb matrix quantifies the average repulsion between electrons, representing the electrostatic interaction among all pairs. In contrast, the exchange matrix is a purely quantum concept that follows from the antisymmetry requirement of the wave function. A primary focus of this thesis is the efficient calculation of the Coulomb and in particular exchange matrices.

To mitigate the computational burden of calculating two-electron repulsion integrals (ERIs), especially in methods like HF, DFT and post-Hartree-Fock, the DF technique, also called RI, is introduced as a strategic solution. The core conceptual idea behind DF is to approximate the product of two orbitals $\xi_\alpha(\mathbf{r})\xi_\beta(\mathbf{r})$ using an auxiliary basis set $x_\mu(\mathbf{r})$ [45]. This approximation significantly reduces the computational cost associated with ERIs without substantially sacrificing accuracy. The approximation can be written as:

$$\xi_\alpha(\mathbf{r})\xi_\beta(\mathbf{r}) \approx \sum_x C_x^{\alpha\beta}\phi_x(\mathbf{r}) \tag{2.11}$$

where $C_\mu^{\alpha\beta}$ are the expansion coefficients for the auxiliary basis functions ϕ_x that approximate the product of the atomic orbitals α and β. By applying DF, the four-center ERIs can be approximated as:

$$(\alpha\beta|\gamma\delta) \approx \sum_{xy}(\alpha\beta|x)(x|y)^{-1}(y|\gamma\delta) \tag{2.12}$$

where $(\alpha\beta|x)$ and $(y|\gamma\delta)$ are three-center integrals,

$$(\alpha\beta|x) \equiv \int d^3r_1 d^3r_2 \xi_\alpha(\mathbf{r}_1)\xi_\beta(\mathbf{r}_1)\frac{1}{r_{12}}\phi_x(\mathbf{r}_2) \tag{2.13}$$

and $(x|y)$ is the two-center Coulomb integral between auxiliary basis functions.

$$(x|y) \equiv \int d^3r_1 d^3r_2 \phi_x(\mathbf{r}_1)\frac{1}{r_{12}}\phi_y(\mathbf{r}_2) \tag{2.14}$$

The inverse of $(x|y)$ is referred to as the Coulomb metric matrix. The DF strategy reduces the complexity of the ERIs calculations from $O(n^4)$ to $O(n^3)$ for small molecules. For larger systems the number of 3-center integrals still scales as $O(n^2)$, although with a much smaller pre-factor as the four-center ERI. With DF, larger molecular systems can be studied with a modest loss of accuracy. The key to using the RI approximation is to never construct the 4-index quantities themselves, but rather optimize the computational steps when calculating quantities of interest. We will see many examples of this strategy in the remainder of this thesis.

This thesis will explore variations of the use of DF with the purpose to reduce the cost of the calculation of the Coulomb and exchange matrices at the HF stage of calculations. Similar efficiencies can be exploited at the post HF level. The key ingredient is the use of short-range three-index two-electron integrals in HF computations, which will be discussed next.

2.1 Short-range Two-electron Integrals

As previously discussed in **Section 1.5**, the Coulomb potential can be segmented into short-range and long-range components. In our project, we utilize the Ewald partitioning approach to address this [46, 47]. The focus here is on the short-range three-index integrals,

derived by approximating the short-range component of the Coulomb potential. These short-range integrals can be optimized due to the sparsity that arises from the property of exponential decay, resulting in linear scaling with respect to molecular system size. The short-range potential $V_{sr}(r_{12})$ can be represented as:

$$V_{sr}(r_{12}) = \frac{\text{erfc}(\alpha|r_{12}|)}{|r_{12}|} \tag{2.15}$$

$$V_{lr}(r_{12}) = \frac{1}{r_{12}} - V_{sr}(r_{12}) \tag{2.16}$$

where $\text{erfc}(\alpha|r_{12}|)$ denotes the complementary error function of the product of a decay constant, α, and the inter-electronic distance, $|r_{12}|$, providing an effective means to separate the potential into short-range component and long-range component for computational purposes. Additionally, the tuning parameter α plays an important role in balancing the precision and efficiency of the algorithm. Selecting an appropriate value for α, is critical. A lower α value enhances precision, ensuring the short-range potential near to full-range potential, an accurate representation of interactions. Conversely, a higher α value reduces significant interactions, potentially diminishing computational load. The most optimal value is to be determined.

Next, by computing integrals of atomic basis functions over the range-separated Coulomb potential, we can also divide two-electron integrals into segments on short-range and long-range interactions.

$$(\alpha\beta|\gamma\delta) = (\alpha\beta|\gamma\delta)_{sr} + (\alpha\beta|\gamma\delta)_{lr} \tag{2.17}$$

The three-center two-electron integral with resolution of identity (RI) can likewise be expressed by:

$$(\alpha\beta|x) = (\alpha\beta|x)_{sr} + (\alpha\beta|x)_{lr} \tag{2.18}$$

In our previous work [44] we used DF to represent short-range integrals, while a Fourier transform and/or multipole expansion was used to represent the long-range integrals.

Our goal here is to obtain the full Coulomb ERIs, but using a representation in terms of three-index integrals that have a short-range nature. In the usual Ewald partitioning the long-range potential at short-range approaches a constant, but is not zero. The value of the 3-center short-range integrals therefore deviates appreciably from the full Coulomb integrals, even when all orbitals are close together. To accurately simulate the full short-range electron-electron repulsion integrals, we introduce the GTG correction for three-centered integrals. The GTG correction is added to the usual short-range integrals to

correct for their deficiency at short-range term[8].

$$\begin{aligned} V_{sr\_gtg}(r_{12}) &= V_{sr}(r_{12}) + V_{gtg}(r_{12}) \\ &= \frac{\text{erfc}(\alpha|r_{12}|)}{|r_{12}|} + X_0 e^{-\gamma|r_{12}|^2} \end{aligned} \tag{2.19}$$

$$\begin{aligned} V_{lr\_gtg}(r_{12}) &= \frac{1}{r_{12}} - V_{sr}(r_{12}) - V_{gtg}(r_{12}) \\ &= \frac{1}{r_{12}} - V_{sr}(r_{12}) - X_0 e^{-\gamma|r_{12}|^2} \end{aligned} \tag{2.20}$$

The parameter X_0 and γ are chosen such that the value of $V_{lr}(0)$ is zero, as well as the derivative $\frac{dV_{lr}}{dr}|_{r=0}$. Through these condition, the α parameter in range-separated Coulomb determines the value of X_0 and γ:

$$\begin{aligned} X_0 &= \lim_{r\to 0} \frac{\text{erfc}(\alpha|r_{12}|)}{|r_{12}|} \\ &= \frac{2\alpha}{\sqrt{\pi}} \end{aligned} \tag{2.21}$$

$$\frac{d^2}{dr^2} V_{lr}(r=0) = 2X_0(-\frac{\alpha^2}{3} + \gamma)$$

$$\gamma_{opt} = \frac{\alpha^2}{3} \tag{2.22}$$

Hence, the potential is controlled by a single parameter, α, with the characteristic that a smaller α value extends the short-range region's reach. In the next section we discuss the construction of variations of DF, in particular using short-range 3-center integrals.

2.2 Construction of DF approximations using short-range 3-centered integrals.

Despite the widespread and accurate application of the standard RI approximation previously discussed, it should be noted that the implementation of three-index integrals $(\alpha\beta|x)$ scales quadratically (proportional to $n_{aux} \cdot n_{ao} \cdot L$, where L is a constant factor) with the size of the molecular systems [32, 48, 49, 50]. This scaling property still leads to escalated computational demands as system size increases, resulting in the standard RI being resource-intensive. In this section we will first discuss a derivation of the usual RI approximation, and then generalize the derivation to use short-range 3-center integrals.

The auxiliary basis set is non-orthogonal. The overlap matrix is defined as

$$S_{xy} = \int d^3r \phi_x(\mathbf{r}) \phi_y(\mathbf{r}) \tag{2.23}$$

The resolution of the identity takes the form $I = \sum_{x,y} |x)(S^{-1})_{xy}(y|$. Let us first derive the usual formula for the RI approximation. We will write the real space multiplicative

potential explicitly in a short-hand notation as $\mathcal{V}$. The integrals can then be represented as

$$\begin{aligned}(\alpha\beta|\gamma\delta) &= (\alpha\beta|\mathcal{V}|\gamma\delta) \\ &= (\alpha\beta|\mathcal{V}\mathcal{V}^{-1}\mathcal{V}|\gamma\delta) \approx (\alpha\beta|\mathcal{V}I\mathcal{V}^{-1}I\mathcal{V}|\gamma\delta) \\ &= \sum_{x,y}(\alpha\beta|\mathcal{V}|x)\Big[\sum_{t,u} S^{-1}_{xt}(t|\mathcal{V}^{-1}|u)S^{-1}_{uy}\Big](y|\mathcal{V}|\gamma\delta) \end{aligned} \tag{2.24}$$

We can then show that the quantity between square brackets is the inverse of the matrix $\mathbf{V}$ with matrix elements $(x|\mathcal{V}|y)$:

$$\sum_{t,u} S^{-1}_{xt}(t|\mathcal{V}^{-1}|u)S^{-1}_{uy} = (x|\mathcal{V}|y)^{-1} \equiv (\mathbf{V}^{-1})_{xy} \tag{2.25}$$

The proof essentially follows from a matrix multiply by $(w|\mathcal{V}|x)$, and recognizing the resolution of the identity

$$\begin{aligned}\sum_{x,t,u}(w|\mathcal{V}|x)\Big[S^{-1}_{xt}(t|\mathcal{V}^{-1}|u)S^{-1}_{uy}\Big] &= \sum_{u}(w|\mathcal{V}I\mathcal{V}^{-1}|u)S^{-1}_{uy} \\ &\approx \sum_{u}(w|1|u)S^{-1}_{uy} = \delta_{wy}\end{aligned} \tag{2.26}$$

The representation of two-electron integrals using an RI expansion using *short-range* Coulomb potentials is obtained from

$$\begin{aligned}(\alpha\beta|\mathcal{V}|\gamma\delta) &= (\alpha\beta|(\mathcal{V}_{sr}\mathcal{V}^{-1}_{sr}\mathcal{V}\mathcal{V}^{-1}_{sr}\mathcal{V}_{sr})|\gamma\delta) \\ &\approx (\alpha\beta|(\mathcal{V}_{sr}I\mathcal{V}^{-1}_{sr}I\mathcal{V}I\mathcal{V}^{-1}_{sr}I\mathcal{V}_{sr})|\gamma\delta)\end{aligned} \tag{2.27}$$

Note that $\mathcal{V}_{sr}\mathcal{V}^{-1}_{sr}\mathcal{V}\mathcal{V}^{-1}_{sr}\mathcal{V}_{sr} = \mathcal{V}$ in **Equation 2.27** and we proceed by including additional resolutions of the identity between each operator. If we combine the inverse overlap matrices inside the I operators with the inverses of the potential and simplify, we find

$$(\alpha\beta|\mathcal{V}|\gamma\delta) \approx \sum_{x,t,u,y}(\alpha\beta|\mathcal{V}_{sr}|x)(x|\mathcal{V}_{sr}|t)^{-1}(t|\mathcal{V}|u)(u|\mathcal{V}_{sr}|y)^{-1}(y|\mathcal{V}_{sr}|\gamma\delta) \tag{2.28}$$

where the inverse of an operator can be expressed as $(\mathcal{V}_{sr})^{-1} \rightarrow (x|\mathcal{V}_{sr}|t)^{-1} \equiv (\mathcal{V}^{-1}_{sr})_{xy}$ in matrix representations.

Using a more compact notation we write

$$\begin{aligned}(\alpha\beta|\gamma\delta) &\approx \sum_{x,t,u,y}(\alpha\beta|x)_{sr}(x|t)^{-1}_{sr}(t|u)_{fr}(u|y)^{-1}_{sr}(y|\gamma\delta)_{sr} \\ &\equiv \sum_{x,y}(\alpha\beta|x)_{sr}M_{xy}(y|\gamma\delta)_{sr},\end{aligned} \tag{2.29}$$

where

$$M_{xy} = \sum_{t,u}(x|t)^{-1}_{sr}(t|u)_{fr}(u|y)^{-1}_{sr} = (\mathcal{V}^{-1}_{sr}\mathcal{V}_{fr}\mathcal{V}^{-1}_{sr})_{xy} \tag{2.30}$$

Some observations are in order. In principal the operator $\mathcal{V}_{sr}$ can be arbitrary. In practice we use a local multiplicative operator, for which the inverse exists. It is important that intgrals over GTO's can be obtained efficiently. The calculation of the metric matrix over the auxiliary basis and the required inverses are not considered to be a bottleneck in calculations of our current interest, as they are 2-index quantities. To obtain the 2-index quantities, and the metric matrix M_{xy} the indices run over the full size of the molecule, and these steps scale as $O(n^2)$ and the matrices are dense. The prefactor is small because they are 2-index quantities.

The key simplification arises for the 3-center integrals. By using a short-range potential the 3-center integrals decay, and the integrals are non-zero only if all three Gaussians $\xi_\alpha, \xi_\beta, \phi_x$ are relatively close together. In detail, this scalability arises because the orbital β needs to be in proximity to α, showing the decay characteristics of the AO overlap matrix. Additionally, the auxiliary index x is required to be spatially close to the (α, β) pair due to the inherently short-range nature of these integrals. Hence, the complexity for evaluating the short-range two-electron three-centered integral will be reduced to $O(n)$, where n represents the $\mathbf{n_{ao}}$. The prefactor of the scaling depends on the range parameter α that is used. The accuracy of the approximation needs to be tested for actual calculations.

We have a number of considerations to be tested. The value of α is of interest, as well as the decision to include the GTO correction to the short-range potential. The results of the tests may be different for the Coulomb and exchange contributions to the Fock matrix, and the total energy.

2.2.1 Delta Correction in Metric Matrix

It should be noted that our approach introduces our RI error four times, as detailed in **Equation 2.27**. As a result, the metric matrix $\mathbf{M}$ may have lost some accuracy. To deal with this, we suggest here to incorporate a Δ-correction within our metric matrix M_{xy}. This correction is predicated on revisiting our approximation strategy. We would like our new approximation to be as accurate as the original DF approximation. We can adjust the definition of the 2-center metric matrix as follows.

$$(\alpha\beta|x)_{fr}(x|y)^{-1}_{fr}(\gamma\delta|y)_{fr} \approx (\alpha\beta|x)_{sr}\Big(M_{xy} + \Delta_{xy}\Big)(\gamma\delta|x)_{sr} \tag{2.31}$$

If we multiply with the transpose of the short-range three-index integrals on both sides:

$$(\alpha\beta|x)^T_{sr}(\alpha\beta|x)_{fr}(x|y)^{-1}_{fr}(\gamma\delta|y)_{fr}(\gamma\delta|x)^T_{sr} \approx (\alpha\beta|x)^T_{sr}(\alpha\beta|x)_{sr}\Big(M_{xy} + \Delta_{xy}\Big)(\gamma\delta|x)_{sr}(\gamma\delta|x)^T_{sr} \tag{2.32}$$

$$O^T_{xy}(x|y)^{-1}_{fr}O_{xy} \approx P^T_{xy}\Big(M_{xy} + \Delta_{xy}\Big)P_{xy} \tag{2.33}$$

To enhance the efficiency and conciseness of integral calculations, least square fitting has been introduced to the metric matrix to find the best-fitting curve to the two-electron integrals [41, 11]. These improvements involve the definition of two new matrices, P_{xy} and

O_{xy}, as follows:

$$P_{xy} = \sum_{\alpha\beta} (\alpha\beta|x)_{sr} (\alpha\beta|y)_{sr} \tag{2.34}$$

$$O_{xy} = \sum_{\alpha\beta} (\alpha\beta|x)_{fr} (\alpha\beta|y)_{sr} \tag{2.35}$$

The matrix $\mathbf{P}$ is positive definite and symmetric, but in practice it can have very small eigenvalues. The equation for Δ can be expressed in terms of a residual matrix

$$R_1 = O_{xy}^T (x|y)_{fr}^{-1} O_{xy} - P_{xy}^T M_{xy} P_{xy} \tag{2.36}$$

We can then solve for the correction Δ

$$\Delta = (P_{xy}^{-1}) R_1 P_{xy}^{-1} \tag{2.37}$$

Because of the presence of small eigenvalues in the P_{xy} matrix, this **Algorithm A.1.8** employs the pseudo-inverse of the P_{xy} matrix. When the eigenvalues (λ) are smaller than the $thresholdP$ of 10^{-6}, these eigenvalues are adjusted to zero, and the inverse of P_{xy} is computed as $P^{-1} = \sum_{\lambda \in nonzero} v_\lambda \lambda^{-1} v_\lambda^T$. The corrected metric matrix (Q_{xy}) should be implemented with correction term, Δ, which is define by:

$$Q_{xy} = M_{xy} + \Delta_{xy} \tag{2.38}$$

$$R_2 = O_{xy}^T (x|y)_{fr}^{-1} O_{xy} - P_{xy}^T Q_{xy} P_{xy} \tag{2.39}$$

This represents the new residual R_2 after the correction has been applied. Ideally, R_2 should be smaller than R_1, showing that the correction term Δ has effectively reduced the inaccuracies in the metric matrix, thereby enhancing the overall precision of the two-electron integral calculations.

This refined approach, which integrates the matrices P_{xy}, O_{xy}, and the corrected metric matrix Q_{xy}, should improve the precision of calculated two-electron integrals. Through careful correction of inaccuracies with the term Δ and the utilization of sparse matrix storage, this method strikes an optimal balance between computational efficiency and the accuracy essential for quantum chemistry simulation.

A disadvantage of inclusion of the correction is that it requires the calculation of the full-range 3-center integrals. The key is that this has to be only once, and the integrals can easily be processed in small batches (as discussed in a next chapter). The fact that one should use a threshold because the matrix $\mathbf{P}$ is ill-conditioned is another drawback. In the later result section, we will explore the effect of including this correction, to see if the advantages are larger than the drawbacks.

2.3 Implementation in PySCF

This section introduces the Int-Class, a key component of our algorithm, serving as a bridge to both the PySCF package and the LibCint integral library, where it interfaces with C and Fortran Application Programming Interface (API)s for computing one-electron and two-electron integrals over Cartesian, real-spherical, and spinor Gaussian-type functions [51, 52]. Especially, a lower-level function called 'getints' which accepts 'intor_name' to specify the type of integrals (e.g., 'int2e2c', 'int2e3c' and 'int2e4c'), is identified as particularly time-intensive in calculations.

Moreover, this section delves into benchmarking Hartree-Fock calculations against the PySCF DF object. Specifically, we examine variations in α values and the incorporation of GTO corrections to improve two-electron integral approximations. Furthermore, it explores the definition of metric matrices, labeled as M_{xy} and Q_{xy}, aimed at refining the accuracy of electron repulsion integral computations. The strategic use of these adjustable parameters, metric matrices, and thresholds in harmony with the DF approach significantly lowers the computational demand and enhances the accuracy of ERIs estimations for extensive molecular systems. The most important insight is there are no universally optimal values for these parameters, hence the choices made during implementation are tailored to the specific requirements of the computational task at hand. With this understanding, let us now explore the impact of varying these parameters:

2.3.1 Efficiency

To optimize the computational efficiency, these matrices are computed by block-sparse format as outlined in **Algorithm A.1.7**, and finally will be stored in hard disk drives. The approach to the molecular division will be further discussed in chapter 3. In this scenario, our goal is to partition three-index integrals across both the α and β layers, based on the number of atoms in each group, denoted by $\mathbf{n_{gatm}}$.

$$P_{xy} = \sum_{s=1}^{n_{\text{slice}}} \sum_{k=1}^{n_{\text{gatm}}} \sum_{j=1}^{n_{\text{gatm}}} (A_j B_k|x)_{sr} (A_j B_k|y)_{sr} \tag{2.40}$$

$$O_{xy} = \sum_{s=1}^{n_{\text{slice}}} \sum_{k=1}^{n_{\text{gatm}}} \sum_{j=1}^{n_{\text{gatm}}} (A_j B_k|x)_{fr} (A_j B_k|y)_{sr} \tag{2.41}$$

The storage method for matrices, P_{xy} and O_{xy}, on hard drives utilizes a sparse matrix format. This method specifically manipulates in the npz file format, which is supported by the SciPy package, to maximize storage efficiency and optimize space utilization.

2.3.2 Accuracy

To analyze the accuracy under various parameters, we evaluate Exchange and Coulomb integrals using our JK-Engine for Hartree-Fock (HF) calculations, comparing these results

with the accuracy achieved using the PySCF density fitting (DF) method. More detailed procedures of JK-Engine will be illustrated in the next chapter, with a current focus on accuracy across different settings. Below are the configurations set up for evaluation:

Basis set: cc-pVTZ or cc-pVDZ, with a particular focus on the cc-pVTZ basis set for its importance in accurately modeling these molecular systems.

Auxiliary basis set: cc-pvtz-jkfit

Initial tests on small systems for: Our initial tests are carried out for small systems, notably, H_2O, C_2H_4, C_2H_6. These systems are large enough to test some critical aspects. In addition, selected simulation systems are included (a non-interacting water dimer and a long alkane chain with 20 carbon atoms) that are large enough that some of the short-range integrals of $(\alpha\beta|x)_{sr}$ with the long distance between (α, β) pairs approach zero. Using these initial test systems we investigate the following aspects:

Important Aspects for Benchmarking:

- Calculation of the K-matrix (Exchange matrix, $K_{\alpha\beta}$).
- Calculation of the J-matrix (Coulomb matrix, $J_{\alpha\beta}$).
- Determination of the total electronic energy E and exchange energy E_{ex} through the following equations:

$$E = \frac{1}{2}\mathrm{Tr}(\hat{h}_{\alpha\beta} + F_{\alpha\beta}) \cdot D_{\alpha\beta} \tag{2.42}$$

$$E_{ex} = -\frac{1}{2}\mathrm{Tr}(K_{\alpha\beta} \cdot D_{\alpha\beta}) \tag{2.43}$$

  where the term Tr stands for the trace of a matrix, which is the sum of the elements on the matrix's main diagonal.

- Reference benchmarking is conducted using Pyscf with the density fitting (DF) approach, a method chosen for its effective error cancellation capabilities inherent in the resolution of identity (RI) approach. This involves the instantiation of a density fitting object ($'dfobj = df.DF(Mol).build()'$).

We explore several variations in our JK-Engine approach, which include:

a) The choice of a decay constant (α) in the complementary error function for range-separated Coulomb potential: $0.1, 0.2, 0.3, \cdots 2.0, 5.0, 10.0, 15.0, 20.0, 100$.

b) The choice of inclusion of Gaussian correction term ($V_{gtg}(r_{12})$).

c) The choice of the threshold for pseudo inversion ($ThreshP$).

d) The choice to include a least squares fitting correction term (Δ-correction term), which involves selecting either the metric matrix M_{xy} or Q_{xy}.

f) Validation on additional molecules , including A. Glycine($C_2H_5NO_2$), B. Toluene($C_6H_5CH_3$), C. (3Z,5Z,7Z,9Z,11Z)-12-aminododeca-3,5,7,9,11-pentaen-2-one($C_{12}H_{15}NO$), and D. Benzophenone($(C_6H_5)_2C{=}O$)).

Optimal Settings: We find that lower values of α present more challenges in computational cost but lead to greater accuracy. It is recommended to evaluate options b), c), and d) to determine the optimal value of α for specific tasks. These different values of α should then be used in subsequent calculations for improved accuracy and performance.

a) Variations of the Parameter α

Figure 2.1 through Figure 2.8 present the logarithm of differences in calculated values for exchange integrals ($K_{\mathrm{M}} - K_{\mathrm{pyscf}}$), Coulomb integrals ($J_{\mathrm{M}} - J_{\mathrm{pyscf}}$), exchange energies ($E_{\mathrm{ex,\ M}} - E_{\mathrm{ex,\ pyscf}}$), and electronic energies ($E_{\mathrm{elec,\ M}} - E_{\mathrm{elec,\ pyscf}}$), between our JK-Engine with metric matrix M_{xy} and JK from PySCF DF method, utilizing different α parameters with the range from 0.1 to 100. Furthermore, Appendix B, containing **Tables B.1, B.2, B.3, B.4**, presents specific values of these discrepancies. The maximum absolute discrepancies between the JK-Engine and the DF approach from PySCF are computed to lie within the 10^{-5} to 10^{-2} range for exchange integrals, denoted as $K_{\alpha\beta}$. These discrepancies were observed across several α values, from 0.1 to 20, for small molecular systems including the water monomer, water dimer, ethane, and ethylene. Additionally, the dark blue curve in **Figure 2.9** illustrates the decrease in the number of non-zero (NNZ) entries of short-range integral slice $(\alpha B|x)_{sr}$ as the value of α parameter increases in a relatively large molecular system $C_{20}H_{42}$. Consequently, this reduction in minor entries may lead to a decrease in the precision of our algorithm, shown in **Table 2.1**.

It is noted that a lower value of α leads to increased accuracy and computational cost for both exchange and Coulomb integrals. Conversely, larger α values significantly reduce the effective range of the short-range potential, thus increasing the sparsity of the short-range integrals $(\alpha\beta|x)_{sr}$ and offering computational efficiency advantages.

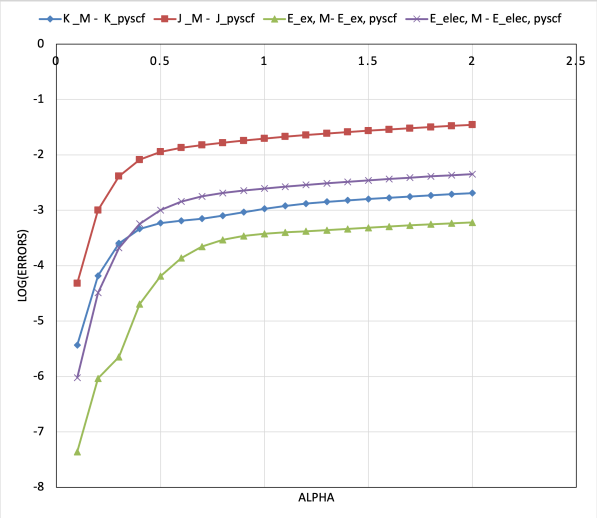


Figure 2.1: Comparative Analysis of Error Metrics Between JK-Engine Algorithm with the metric matrix M_{xy} and Density Fitting Object from PySCF for Water Monomer H_2O under various α values in cc-pVTZ basis set and cc-pvtz-jkfit auxiliary basis set.

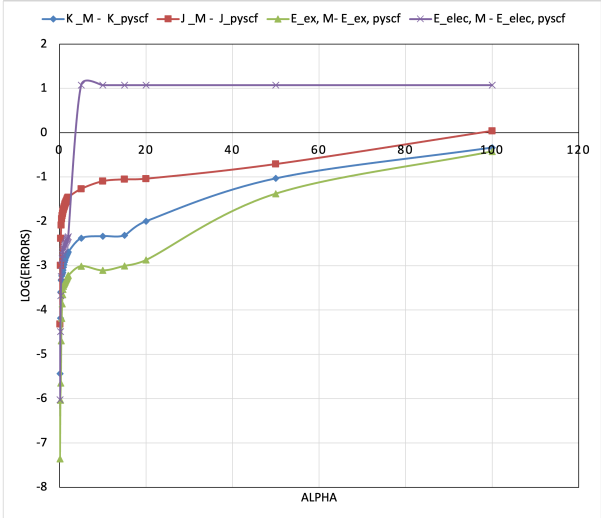


Figure 2.2: Comparative Analysis of Error Metrics Between JK-Engine Algorithm with the metric matrix M_{xy} and Density Fitting Object from PySCF for Water Monomer H_2O under various α values in cc-pVTZ basis set and cc-pvtz-jkfit auxiliary basis set

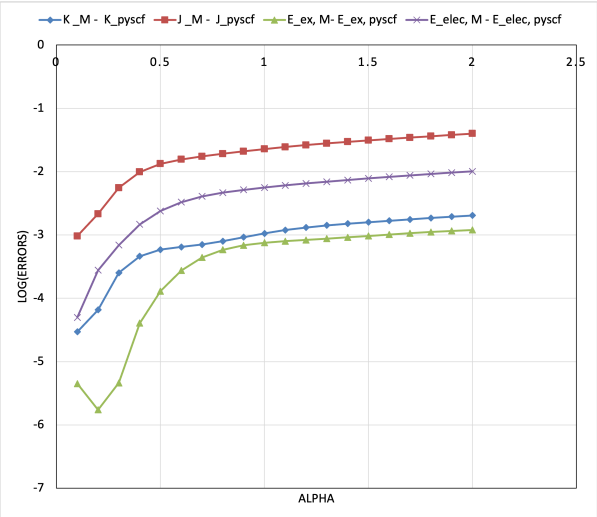


Figure 2.3: Comparative Analysis of Error Metrics Between JK-Engine Algorithm with the metric matrix M_{xy} and Density Fitting Object from PySCF for Water Dimer $(H_2O)_2$ under various α values in cc-pVTZ basis set and cc-pvtz-jkfit auxiliary basis set.

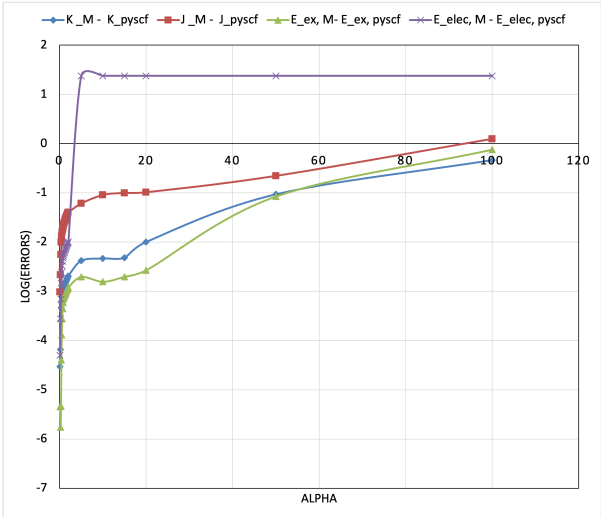


Figure 2.4: Comparative Analysis of Error Metrics Between JK-Engine Algorithm with the metric matrix M_{xy} and Density Fitting Object from PySCF for Water Dimer $(H_2O)_2$ under various α values in cc-pVTZ basis set and cc-pvtz-jkfit auxiliary basis set

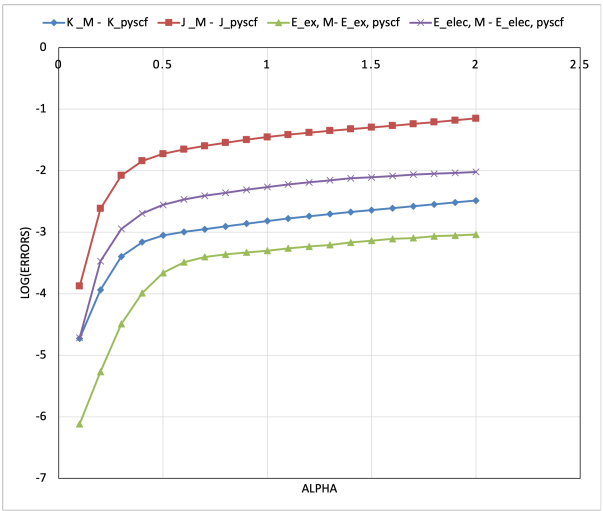


Figure 2.5: Comparative Analysis of Error Metrics Between JK-Engine Algorithm with the metric matrix M_{xy} and Density Fitting Object from PySCF for C_2H_4 under various α values in cc-pVTZ basis set and cc-pvtz-jkfit auxiliary basis set.

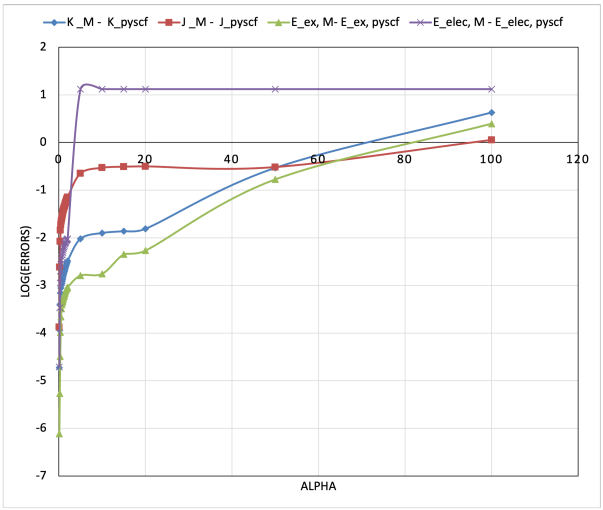


Figure 2.6: Comparative Analysis of Error Metrics Between JK-Engine Algorithm with the metric matrix M_{xy} and Density Fitting Object from PySCF for C_2H_4 under various α values in cc-pVTZ basis set and cc-pvtz-jkfit auxiliary basis set

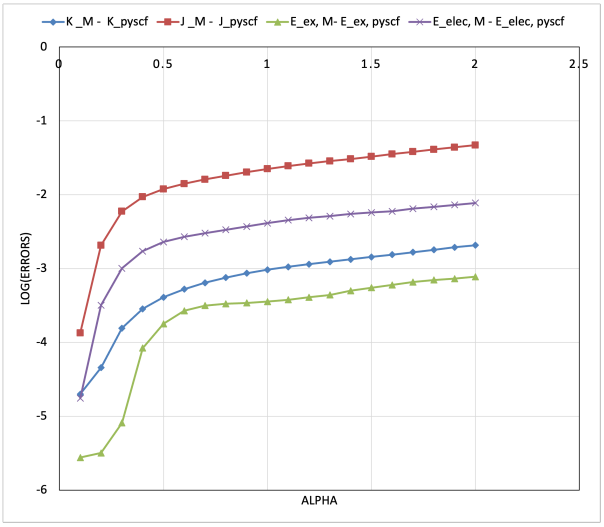


Figure 2.7: Comparative Analysis of Error Metrics Between JK-Engine Algorithm with the metric matrix M_{xy} and Density Fitting Object from PySCF for C_2H_6 under various α values in cc-pVTZ basis set and cc-pvtz-jkfit auxiliary basis set.

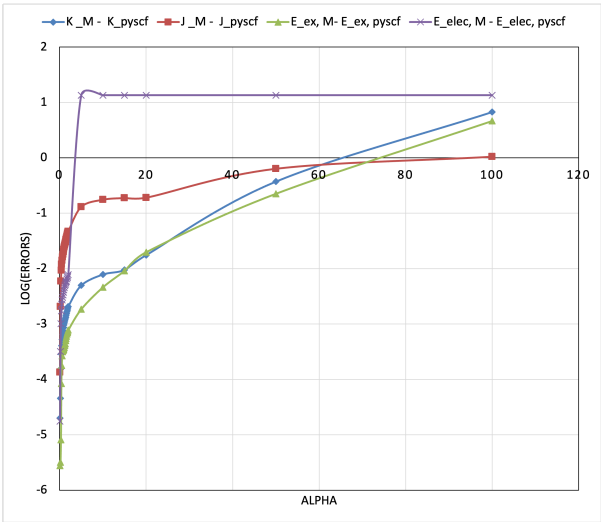


Figure 2.8: Comparative Analysis of Error Metrics Between JK-Engine Algorithm with the metric matrix M_{xy} and Density Fitting Object from PySCF for C_2H_6 under various α values in cc-pVTZ basis set and cc-pvtz-jkfit auxiliary basis set

α	$K_{\mathrm{M}} - K_{\mathrm{pyscf}}$	$K_{\mathrm{Q}} - K_{\mathrm{pyscf}}$	$J_{\mathrm{M}} - J_{\mathrm{pyscf}}$	$J_{\mathrm{Q}} - J_{\mathrm{pyscf}}$
0.6	1.08×10^{-4}	1.08×10^{-4}	3.52×10^{-2}	9.31×10^{-3}
1	8.95×10^{-4}	8.79×10^{-4}	5.96×10^{-2}	1.36×10^{-2}
5	4.82×10^{-3}	4.60×10^{-3}	2.20×10^{-1}	9.04×10^{-2}
10	2.78×10^{-2}	2.74×10^{-2}	2.73×10^{-1}	1.45×10^{-1}
15	9.94×10^{-2}	9.93×10^{-2}	3.83×10^{-1}	3.26×10^{-1}
20	2.68×10^{-1}	2.67×10^{-1}	5.84×10^{-1}	5.83×10^{-1}
50	5.93	5.93	3.23	3.23
100	5.03×10	5.03×10	8.30×10	8.30×10

Table 2.1: Discrepancies between JK-Engine and PySCF DF method for $C_{20}H_{42}$ under various α values in cc-pVDZ basis set and cc-pvtz-jkfit auxiliary basis set.

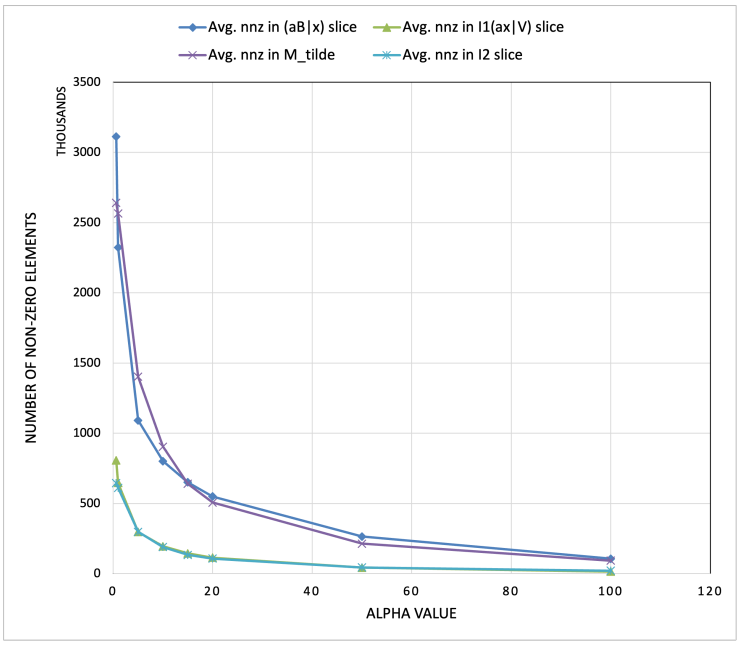


Figure 2.9: Comparative Analysis of Non-Zero Entries in JK-Engine Algorithm with the metric matrix $M_{\alpha\beta}$ for $C_{20}H_{42}$ under various α values in cc-pVDZ basis set and cc-pvtz-jkfit auxiliary basis set.

b) Variations in Gaussian Corrections $V_{\mathbf{gtg}}(r_{12})$

Molecules	$K_{\mathrm{M}} - K_{\mathrm{pyscf}}$	$J_{\mathrm{M}} - J_{\mathrm{pyscf}}$	$E_{\mathrm{ex,\ M}} - E_{\mathrm{ex,\ pyscf}}$	$E_{\mathrm{elec,\ M}} - E_{\mathrm{elec,\ pyscf}}$
$(H_2O)_1$	2.04×10^{-3}	3.55×10^{-2}	5.51×10^{-4}	4.05×10^{-3}
$(H_2O)_2$	2.04×10^{-3}	4.05×10^{-2}	1.10×10^{-3}	9.14×10^{-3}
C_2H_4	3.29×10^{-3}	7.36×10^{-2}	8.18×10^{-4}	8.95×10^{-3}
C_2H_6	1.92×10^{-3}	4.48×10^{-2}	7.13×10^{-4}	8.19×10^{-3}

Table 2.2: Discrepancy Evaluation Between JK-Engine without Gaussian Germinal Corrections ($V_{gtg}(r)$) and Density Fitting in PySCF for Various Molecules: This table shows the differences in computed values for exchange and Coulomb integrals, exchange energies, and electronic energies between the JK-Engine algorithm, employing the metric matrix $M_{\alpha\beta}$, and the Density Fitting approach in PySCF. These calculations were performed under the setting of $\alpha = 0.6$ with the cc-pVTZ basis set and the cc-pvtz-jkfit auxiliary basis set for molecules including water clusters, ethane, and ethylene.

Molecules	$K_{\mathrm{M,gtg}} - K_{\mathrm{pyscf}}$	$J_{\mathrm{M,gtg}} - J_{\mathrm{pyscf}}$	$E_{\mathrm{ex,\ gtg}} - E_{\mathrm{ex,\ pyscf}}$	$E_{\mathrm{elec,\ gtg}} - E_{\mathrm{elec,\ pyscf}}$
$(H_2O)_1$	6.47×10^{-4}	1.34×10^{-2}	1.37×10^{-4}	1.43×10^{-3}
$(H_2O)_2$	6.47×10^{-4}	1.56×10^{-2}	2.75×10^{-4}	3.30×10^{-3}
C_2H_4	1.01×10^{-3}	2.21×10^{-2}	3.25×10^{-4}	3.40×10^{-3}
C_2H_6	5.27×10^{-4}	1.41×10^{-2}	2.69×10^{-4}	2.70×10^{-3}

Table 2.3: Discrepancy Evaluation Between JK-Engine with Gaussian Germinal Corrections ($V_{gtg}(r)$) and Density Fitting in PySCF for Various Molecules: This table shows the differences in computed values for exchange and Coulomb integrals, exchange energies, and electronic energies between the JK-Engine algorithm, employing the metric matrix $M_{\alpha\beta}$, and the Density Fitting approach in PySCF. These calculations were performed under the setting of $\alpha = 0.6$ with the cc-pVTZ basis set and the cc-pvtz-jkfit auxiliary basis set for molecules including water clusters, ethane, and ethylene.

In the context of evaluating the impact of Gaussian-Type Germinal (GTG) corrections on computational accuracy, the JK-Engine algorithm includes a Boolean parameter, $JKEngine.gtg$, which can be set to True or False. This parameter dictates whether the short-range three-centered integrals are adjusted for GTG corrections. **Table 2.2** illustrates the discrepancy analysis in the absence of GTG corrections, contrasting the JK-Engine algorithm calculated values with those from PySCF DF method. On the other hand, **Table 2.3** represents the results where GTG corrections are applied to the short-range DF integrals. The discrepancies in calculated values for exchange and Coulomb integrals, exchange energies, and electronic energies between the two methods are documented under the same computational conditions, specifically an α value of 0.6, and using both the cc-pVTZ basis set and the cc-pvtz-jkfit auxiliary basis set for a set of molecules.

$$\begin{aligned} V_{sr\_gtg}(r_{12}) &= V_{sr}(r_{12}) + V_{gtg}(r_{12}) \\ &= \frac{\mathrm{erfc}(\alpha|r_{12}|)}{|r_{12}|} + X_0 e^{-\gamma|r_{12}|^2} \end{aligned} \tag{2.44}$$

When comparing the two tables, the calculations with GTG corrections (**Table 2.3**) consistently yield more accurate results for exchange and Coulomb integrals, exchange energies, and electronic energies. To improve the accuracy of future computational analyses, the parameter *JKEngine.gtg* will be set to True as default.

c) Variations in the Value of ThreshP

As detailed in the previous discussion, a novel approach for representing effective range integrals (ERIs) through the use of short-range three-centered integrals and the metric matrix M_{xy} addresses the resolution of identity (RI) errors, denoted as R_1:

$$O_{xy}^{T}(x|y)fr^{-1}Oxy \approx P_{xy}^{T}(M_{xy}+\Delta)P_{xy} \tag{2.45}$$

$$R_1 = O_{xy}^{T}(x|y)_{fr}^{-1}Oxy - P_{xy}^{T}M_{xy}P_{xy} \tag{2.46}$$

To mitigate RI errors, we introduce a Δ-correction, aiming to enhance the alignment with full-range two-electron integrals. The calculation of Δ-corrections involves the pseudo-inverse of the P_{xy} matrix. Significantly, the selection threshold for non-zero eigenvalues within the pseudo-inverse algorithm plays an important role in determining the accuracy of exchange and Coulomb integrals.

$$\Delta = (P_{xy}^{-1})^{T}R_1P_{xy}^{-1} \tag{2.47}$$

$$R_2 = O_{xy}^{T}(x|y)_{fr}^{-1}O_{xy} - P_{xy}^{T}(M_{xy}+\Delta)P_{xy} \tag{2.48}$$

Figures 2.10, 2.11, 2.12, 2.13 depict the logarithm of differences in computed values for exchange ($K_{\mathrm{Q}} - K_{\mathrm{pyscf}}$) and Coulomb integrals ($J_{\mathrm{Q}} - J_{\mathrm{pyscf}}$), exchange energies ($E_{\mathrm{ex,\,Q}} - E_{\mathrm{ex}}$), and electronic energies ($E_{\mathrm{elec,\,Q}} - E_{\mathrm{elec}}$) between the JK-Engine algorithm and the Density Fitting approach in PySCF, across different pseudo-inverse thresholds denoted as *threshP*. These computations were conducted using the settings $\alpha = 0.6$, the cc-pVTZ basis set, and the cc-pvtz-jkfit auxiliary basis set for various molecular systems including water clusters, H_2O, $(H_2O)_2$, ethane C_2H_4, and ethylene C_2H_6. Specifically, **Tables B.5, B.6, B.7, B.8** in Appendix B illustrate the impact of varying the pseudo-inverse threshold (*threshP*) on the RI errors, particularly the least squares fitting error R_2, within water clusters, ethane, and ethylene molecular systems. In principle, the optimal choice of *threshP* minimizes R_2, thereby enhancing the accuracy of the JK-Engine using metric matrix Q_{xy} in simulating electronic interactions within these molecules. Although some of *threshP* values significantly reduce the least squares fitting error R_2, they do not significantly alter the overall accuracy of the calculations for exchange and Coulomb integrals. Ultimately, the optimal *threshP* values for minimizing RI errors, which are depicted as the minimum points in **Figures 2.10 to 2.13**, fall within the range of 10^{-4} to 10^{-6}. The metric matrix Q_{xy} employs a default value of 10^{-4} for *ThreshP*.

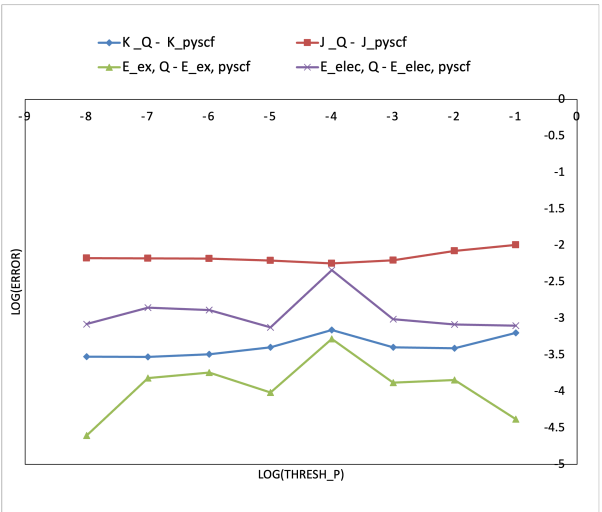


Figure 2.10: Error Analysis Between JK-Engine with the metric matrix Q_{xy} and Density Fitting Object in PySCF for Various Threshold in the inversion of P_{xy} Matrix, $threshP$ for Water Monomer (H_2O).

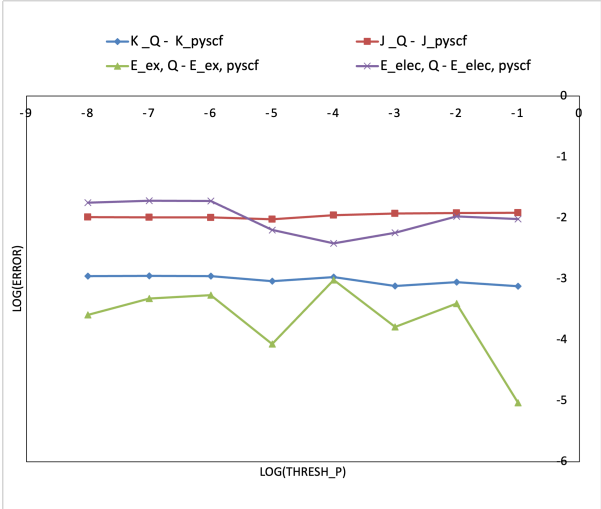


Figure 2.11: Error Analysis Between JK-Engine with the metric matrix Q_{xy} and Density Fitting Object in PySCF for Various Threshold in the inversion of P_{xy} Matrix $threshP$ for Water Dimer $(H_2O)_2$.

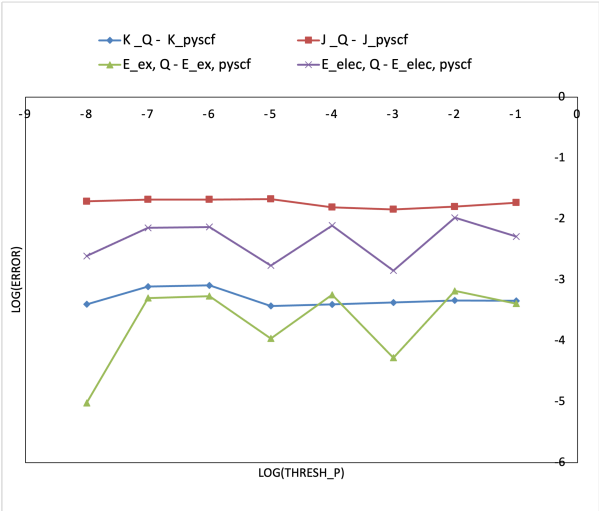


Figure 2.12: Error Analysis Between JK-Engine with the metric matrix Q_{xy} and Density Fitting Object in PySCF for Various Threshold in the inversion of P_{xy} Matrix $threshP$ for Ethylene (C_2H_4).

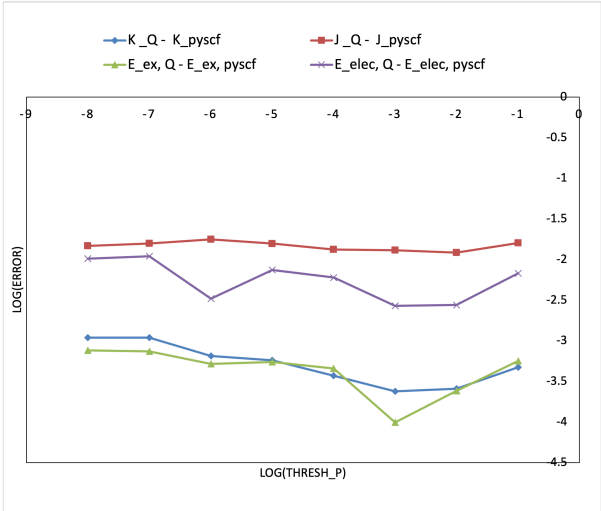


Figure 2.13: Error Analysis Between JK-Engine with the metric matrix Q_{xy} and Density Fitting Object in PySCF for Various Threshold in the inversion of P_{xy} Matrix $threshP$ for Ethane (C_2H_6).

d) Variations in the Metric Matrices M_{xy} and Q_{xy}

Molecules	$K_{\mathrm{M}} - K_{\mathrm{pyscf}}$	$J_{\mathrm{M}} - J_{\mathrm{pyscf}}$	$E_{\mathrm{ex,\ M}} - E_{\mathrm{ex,\ pyscf}}$	$E_{\mathrm{elec,\ M}} - E_{\mathrm{elec,\ pyscf}}$
$(H_2O)_1$	6.47×10^{-4}	1.34×10^{-2}	1.37×10^{-4}	1.43×10^{-3}
$(H_2O)_2$	6.47×10^{-4}	1.56×10^{-2}	2.75×10^{-4}	3.30×10^{-3}
C_2H_4	1.01×10^{-3}	2.21×10^{-2}	3.25×10^{-4}	3.40×10^{-3}
C_2H_6	5.27×10^{-4}	1.41×10^{-2}	2.69×10^{-4}	2.70×10^{-3}
Mols	$K_{\mathrm{Q}} - K_{\mathrm{pyscf}}$	$J_{\mathrm{Q}} - J_{\mathrm{pyscf}}$	$E_{\mathrm{ex,\ Q}} - E_{\mathrm{ex}}$	$E_{\mathrm{elec,\ Q}} - E_{\mathrm{elec}}$
$(H_2O)_1$	3.22×10^{-4}	6.57×10^{-3}	1.82×10^{-4}	1.30×10^{-3}
$(H_2O)_2$	1.11×10^{-3}	1.02×10^{-2}	5.41×10^{-4}	1.90×10^{-2}
C_2H_4	8.13×10^{-4}	2.08×10^{-2}	5.42×10^{-4}	7.37×10^{-3}
C_2H_6	6.50×10^{-4}	1.77×10^{-2}	5.21×10^{-4}	3.30×10^{-3}

Table 2.4: Error Analysis Between JK-Engine with the metric matrix M_{xy} and Density Fitting Object in PySCF for Various Molecules: This table shows the differences in computed values for exchange and Coulomb integrals, exchange energies, and electronic energies between the JK-Engine algorithm, and the Density Fitting approach in PySCF. These calculations were performed under the setting of $\alpha = 0.6$ with the cc-pVTZ basis set and the cc-pvtz-jkfit auxiliary basis set for molecules including water clusters, ethane, and ethylene. The default $threshP = 10^{-4}$ is used.

Molecuels	$(R_1)_{max}$	$(R_2)_{max}$
$(H_2O)_1$	43.8	0.012
$(H_2O)_2$	43.8	3.067
C_2H_4	65.4	0.412
C_2H_6	147.3	0.291

Table 2.5: Error Analysis Between JK-Engine with the metric matrix Q_{xy} and Density Fitting Object in PySCF for Various Molecules: This table indicates the differences in computed values for exchange and Coulomb integrals, exchange energies, and electronic energies between the JK-Engine algorithm, and the Density Fitting approach in PySCF. These calculations were performed under the setting of $\alpha = 0.6$ with the cc-pVTZ basis set and the cc-pvtz-jkfit auxiliary basis set for molecules including water clusters, ethane, and ethylene. The default $threshP = 10^{-4}$ is used

Table 2.4 provides comparative error analysis for different molecules using metric matrix Q_{xy} and M_{xy} respectively, in contrast to the standard PySCF density fitting object. The observed differences are due to the cumulative approximations within the RI errors, particularly regarding the transformed two-electron two-centered integrals. The differences between these transformed integrals, denoted by $O_{xy}^T(x|y)_{fr}^{-1}O_{xy}$, and the transformed metric matrices $P_{xy}^T M_{xy} P_{xy}$ and $P_{xy}^T Q_{xy} P_{xy}$, are represented as residuals R_1 and R_2. These

residuals are defined as:

$$R_1 = O_{xy}^T (x|y)_{fr}^{-1} O_{xy} - P_{xy}^T M_{xy} P_{xy} \tag{2.49}$$

$$R_2 = O_{xy}^T (x|y)_{fr}^{-1} O_{xy} - P_{xy}^T Q_{xy} P_{xy} \tag{2.50}$$

As shown in **Table 2.5**, applying the Δ-correction substantially reduces the maximum residuals from R_1 and R_2 in the fitting process. However, the Hartree-Fock analysis demonstrates only slight adjustments, containing exchange integrals, Coulomb integrals, exchange energy, and electronic energy. This is a surprising result. It indicates that certain two-electron integrals can be improved, but apparently these integrals are not very important in actual calculations. The necessary use of $threshP$ is inconvenient and somewhat suspect, while additionally, the application of Δ corrections introduces extra computational burdens. In particular the calculation of full-range two-electron three-centered integrals $(\alpha\beta|x)_{fr}$ during the computation and storage of Pxy and O_{xy} matrices is computationally expensive. Therefore, we recommend against incorporating Δ corrections into our JK-Engine algorithm for future calculations, while continuing to utilize the metric matrix M_{xy}.

We can phrase this conclusion in a more positive fashion. Apparently, it is very hard to systematically improve on the metric matrix **M** that we derived by inserting an approximate resolution of the identity four times. This is not the result we anticipated, but it is a nice bolstering of the mathematical robustness of the applied RI technique.

f) Validation on Additional Molecular Systems

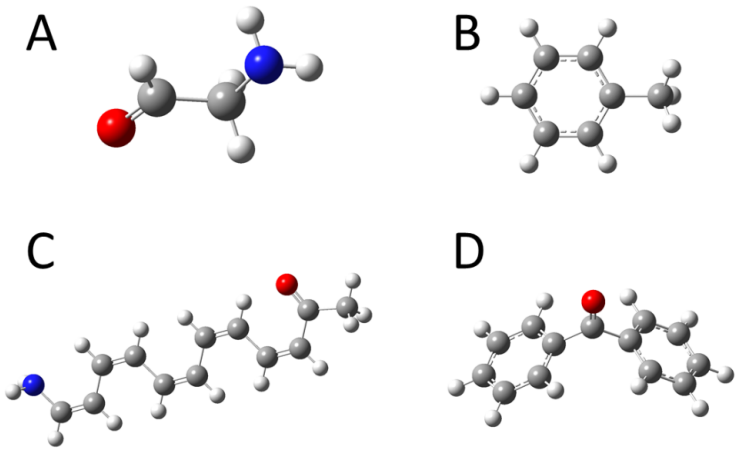


Figure 2.14: Further validation of the JK-Engine Algorithm is demonstrated through its application to four distinct molecules: A. Glycine($C_2H_5NO_2$), B. Toluene($C_6H_5CH_3$), C. (3Z,5Z,7Z,9Z,11Z)-12-aminododeca-3,5,7,9,11-pentaen-2-one($C_{12}H_{15}NO$), and D. Benzophenone($(C_6H_5)_2C{=}O$) [8].

The general applicability of the JK-Engine algorithm is demonstrated through its testing on additional molecular systems, including glycine ($C_2H_5NO_2$), toluene ($C_6H_5CH_3$), 12-aminododeca-3,5,7,9,11-pentaen-2-one($C_{12}H_{15}NO$), and Benzophenone($(C_6H_5)_2C{=}O$) shown in **Figure 2.14**. This broadening to various molecules, as illustrated in **Table 2.6**, highlights the algorithm's consistent accuracy, with differences between the JK-Engine and the Density Fitting approach in PySCF. Such results showcase the algorithm's adaptability and its potential for dependably precise applications across extensive molecular systems.

Molecules	$K_{\mathrm{M}} - K_{\mathrm{pyscf}}$	$J_{\mathrm{M}} - J_{\mathrm{pyscf}}$	$E_{\mathrm{ex,\ M}} - E_{\mathrm{ex}}$	$E_{\mathrm{elec,\ M}} - E_{\mathrm{elec}}$
$C_2H_5NO_2$	5.48×10^{-4}	2.63×10^{-2}	2.16×10^{-4}	5.17×10^{-3}
$C_6H_5CH_3$	4.47×10^{-4}	5.61×10^{-2}	2.45×10^{-4}	8.30×10^{-3}
$C_{12}H_{15}NO$	7.64×10^{-4}	6.95×10^{-2}	4.77×10^{-3}	1.77×10^{-2}
$(C_6H_5)_2C{=}O$	4.20×10^{-3}	3.21×10^{-2}	7.27×10^{-4}	5.04×10^{-3}
Molecules	$K_{\mathrm{Q}} - K_{\mathrm{pyscf}}$	$J_{\mathrm{Q}} - J_{\mathrm{pyscf}}$	$E_{\mathrm{ex,\ Q}} - E_{\mathrm{ex}}$	$E_{\mathrm{elec,\ Q}} - E_{\mathrm{elec}}$
$C_2H_5NO_2$	4.01×10^{-4}	2.68×10^{-2}	1.10×10^{-4}	1.58×10^{-3}
$C_6H_5CH_3$	6.47×10^{-4}	1.56×10^{-2}	2.75×10^{-4}	3.30×10^{-3}
$C_{12}H_{15}NO$	6.71×10^{-4}	6.49×10^{-2}	4.77×10^{-4}	1.66×10^{-2}
$(C_6H_5)_2C{=}O$	3.01×10^{-3}	3.02×10^{-2}	3.25×10^{-4}	3.40×10^{-3}

Table 2.6: Validation Analysis Between JK-Engine and Density Fitting Object in PySCF for Various Molecules on Additional Molecular System: This table shows the differences in computed values for exchange and Coulomb integrals, exchange energies, and electronic energies between the JK-Engine algorithm, and the Density Fitting approach in PySCF. These calculations were performed under the setting of $\alpha = 0.6$ with the cc-pVTZ basis set and the cc-pvtz-jkfit auxiliary basis set for molecules including glycine, toluene, 12-aminododeca-3,5,7,9,11-pentaen-2-one, and benzophenone.

In conclusion, this chapter emphasizes the critical role of parameter selection in optimizing Hartree-Fock calculations within the PySCF framework, particularly through the JK-Engine algorithm. The findings highlight several key insights for future analysis:

- α Parameter: Adjusting α impacts accuracy and efficiency. Lower α improves accuracy but increases the computational load, ideal for precise simulations. Higher α boosts efficiency, suitable for simulations with approximations.
- Δ Corrections: These corrections provide minimal accuracy improvements at a high computational cost of full-range three-index integrals $(\alpha\beta|x)_{fr}$. Therefore, we recommend excluding Δ corrections in metric matrix to reduce computational efforts.
- GTG Corrections: They can enhance accuracy for two-electron integrals and should be considered as default selection based on computational goals and molecule specifics.

The single parameter α plays the key role in the approximation and provides a trade-off between accuracy and efficiency. An interesting observation is that no single α parameter value is universally optimal; therefore, the selection of α parameters during implementation

might be customized to meet the unique demands of the computational task. Let us analyze the selection of the α value for specific tasks:

Initially, for the starting geometry optimization of the molecule, precision is less critical as it's a rough approximation. Therefore, to expedite the optimization process, we can opt for a larger α value, prioritizing speed over accuracy. Secondly, when calculating the total energy at optimized geometry or conducting single-point energy calculations, a high degree of accuracy is required, so we should employ a smaller α value to ensure precise results. Thirdly, the cluster-in-molecule (CiM) developed by the Nooijen group employs the exchange matrix $K_{\alpha\beta}$ to establish the orbital domain. Since the orbital selection scheme does not require highly accurate results, a larger α value can be selected to enhance computational efficiency.

In conclusion, no single α value is universally optimal; the appropriate choice varies based on the specific computational context. Our Int-Class package, which employs Object-Oriented Programming (OOP), makes adjusting α straightforward for diverse applications. To find an equilibrium between accuracy and computational efficiency, we will explore α values of 1.0, 5.0, and 10.0 in Chapter 4, focusing on their impact on timing and memory consumption.

Chapter 3

Calculation of Exchange Matrix using Short-range Integrals

Hartree-Fock (HF) theory approximates the system's wave function as a single determinant by determining the occupied molecular orbitals that minimize the total energy of the system. This leads to the formulation of the Self-Consistent Field (SCF) method, which iteratively solves the HF equations to find these occupied orbitals. The core of the SCF method involves computing the Fock matrix $F_{\alpha\beta}$ in the AO basis set, which is derived from a set of occupied orbitals [53]. The Fock matrix involves three terms: bare h $h_{\alpha\beta}$, Coulomb matrix $J_{\alpha\beta}$, and exchange matrix $K_{\alpha\beta}$. For closed-shell molecules the equations are given as:

$$F_{\alpha\beta} = h_{\alpha\beta} + 2J_{\alpha\beta} - K_{\alpha\beta} \tag{3.1}$$

$$K_{\alpha\beta} = \sum_{\gamma,\delta} (\alpha\gamma|\beta\delta) D_{\gamma,\delta} \tag{3.2}$$

$$J_{\alpha\beta} = \sum_{\gamma,\delta} (\alpha\beta|\delta\gamma) D_{\gamma,\delta} \tag{3.3}$$

Here we used that the density matrix is formed from the α-spin orbitals only, as it is a closed-shell system. As we mentioned in the previous chapter, the two-electron four-index repulsion integrals are simplified by representing the term of short-range three-index integral with DF approximation:

$$(\alpha\beta|\gamma\delta) \approx \sum_{x,y} (\alpha\beta|x)_{sr} Q_{xy} (\gamma\delta|y)_{sr} \tag{3.4}$$

Here $(\alpha\gamma|x)_{sr}$ are the short-range three-center integrals, and Q_{xy} (usually just M_{xy}) is the metric matrix with the RI approximation as discussed in the previous chapter.

When the matrix-RI approximation with short-range integrals is used for the exchange

calculations, one obtains:

$$K_{\alpha\beta} = \sum_{\gamma,\delta} \sum_{x,y} (\alpha\gamma|x)_{sr} Q_{xy} (\beta\delta|y)_{sr} D_{\gamma,\delta} \tag{3.5}$$

To achieve additional efficiencies in the calculation of the exchange matrix the density matrix $D_{\gamma,\delta}$ is decomposed as $D_{\gamma,\delta} = \sum_{\mu} L_{\gamma\mu} L^{T}_{\delta\mu}$, where $L_{\gamma\mu}$ are occupied localized molecular orbitals (LMO). There are various methods available for obtaining these LMOs, including the Foster-Boys localization technique, the Cholesky decomposition of the density matrix, and the Pipek-Mezey (PM) algorithm.

$$K_{\alpha\beta} = \sum_{\gamma,\delta} \sum_{x,y} \sum_{\mu} (\alpha\gamma|x)_{sr} Q_{xy} (\beta\delta|y)_{sr} L_{\gamma\mu} L_{\delta\mu} \tag{3.6}$$

Next, we introduce the intermediate integrals, denoted as $I(\alpha x|\mu)$, to streamline the calculation of the exchange contribution. These intermediates are defined as follows:

$$I(\alpha x|\mu) = \sum_{\gamma} (\alpha\gamma|x)_{sr} L_{\gamma\mu}, \tag{3.7}$$

To calculate this intermediate in practice, it is convenient to resort the primitive localized 3-index integrals, which will be reused many times in the process of SCF iterations, or in the process of a CiM calculation. We will indicate this through a permutation of indices, e.g.

$$I(\alpha x|\mu) = \sum_{\gamma} (\alpha x|\gamma)_{sr} L_{\gamma\mu}, \tag{3.8}$$

The purpose of these intermediate integrals is to simplify the exchange contribution, which can now be expressed as:

$$K_{\alpha\beta} = \sum_{x,y} \sum_{\mu} I(\alpha x|\mu) Q_{xy} I(\beta y|\mu) \tag{3.9}$$

At this point, it is of interest to analyse what has been achieved in the reformulation. If the LMO are well localized and efficient short-range integrals are used, the intermediate $I(\alpha x|\mu)$ has a localized nature, where all of the contributing AO's α and auxiliary basis functions x are in the vicinity of the localized orbital μ. The intermediate elements $I(\alpha x|\mu)$ can be calculated a few LMOs μ at a time (localized in a similar fashion), and the contributions to the exchange matrix can be obtained. Since x is localized only a subset of Q_{xy} are needed in the assembly of $\mathbf{K}$. We will use sparse matrix techniques to only sum over entries that fall above a threshold. The main challenge is to formulate a precise algorithm that is both memory- and time-efficient. This will be discussed in the remainder of this chapter.

Although the exchange algorithm has been improved through the application of short-range three-index integrals using RI approximations, challenges related to memory issues persist. The main cause of the "out-of-memory", Out-of-Memory (OOM) problem is the

computation of three-index integrals, specifically $(\alpha x|\beta)$ and $I(\alpha x|\mu)$, which require a large amount of memory. These integrals lead to a space complexity on the order of $O(n^3)$, resulting in a rapid increase in memory demand as the system size grows.

To address this memory issue, this chapter will dive into another main class in my algorithm, JK-Engine, by using short-range integrals. This class represents a cornerstone of our computational framework, which is designed to efficiently process two-electron integrals for quantum chemistry calculations. Previous efforts by our group, inspired by Mike Lecours' exploration of block-sparse structures in molecules, partitioned three-index integrals using a fixed set of atoms across a single layer. Our new approach expands on this by employing flexible grouping algorithms that partition large molecules into segments based on heavy atoms across all three layers, $(A_jB_i|X_k)$, encompassing α, β, and x. Additionally, we transform dense three-leg tensors into block-sparse two-leg matrices within our JK-Engine, which enhances both computational speed and memory efficiency. Through a detailed exploration of new data types and JK algorithms, this chapter outlines the challenges of handling a large amount of data and the strategies employed to manage memory usage effectively, ensuring the algorithm's applicability to large molecular systems.

3.1 New Data Type in Two-electron Integrals

In this section, we explore a novel approach to managing and calculating two-electron short-range integrals, a preparation stage of JK-Engine. At the heart of this method is the transition from traditional dense tensor representations of short-range integral to a more efficient and scalable sparse matrix format. This transformation is essential in addressing the intensive computational and memory challenges posed by the increasing size of molecular systems. There are a number of techniques we employed in our JK-Engine. We firstly investigated the grouping algorithm to organize the computational workload around the concept of heavy atoms and adjacent hydrogen atom clusters. The conversion process from three-dimensional tensors to two-dimensional sparse matrices is detailed, highlighting the clever use of index transformation and aggregation techniques to maintain the accuracy and block manipulation of the integral data. This methodology not only optimizes storage and computational resources but also sets the preliminary stage for advanced operations in the calculation of exchange and Coulomb matrices.

3.1.1 Dynamic Grouping Algorithm

The critical step in the preparation procedure is to convert the three-index two-electron short-range integral with the Gaussian correction, three-dimensional tensor, to a sparse matrix with COO. The three-dimensional tensor (or rank 3 tensor) can be considered as a cuboid with edges a, b, c and contains abc elements, shown in **Figure 3.1**. In this case, the short-range integral $(\alpha\beta|x)$ has $\mathbf{n_{ao}}\times\mathbf{n_{ao}}\times\mathbf{n_{naux}}$ elements, where $\mathbf{n_{ao}}$ represents the number of AO and $\mathbf{n_{naux}}$ represents the number of auxiliary orbitals. As the size of the input molecules increases, the memory requirements for calculating these short-range integrals exceed the available memory, leading to out-of-memory errors.

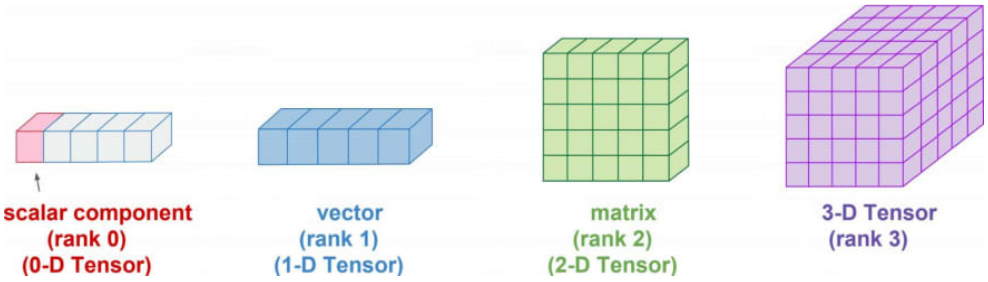


Figure 3.1: Tensors with different ranks [12]

To mitigate the issue of high RAM usage during calculations, the integrals will be sliced based on heavy atoms (non-hydrogen elements such as carbon, nitrogen, and oxygen) and their related adjacent hydrogen atoms. This method initially categorizes integral groups by the specific number of heavy atoms to determine the index range of AOs associated with the atoms within these segments. Each segment's integral will be centered around either a single heavy atom up to a few heavy atoms, depending on our preference for ease of handling.

To group the heavy atoms with adjacent hydrogen atoms, the preparation procedure is to compute the distance matrix for a set of points (atoms in this case) in a 3D space. The distance d between any two atoms $P_i : (x_i, y_i, z_i)$ and $P_j : (x_j, y_j, z_j)$ in a 3D Euclidean space is calculated by the Euclidean distance formula:

$$d(P_i, P_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \tag{3.10}$$

The **Algorithm A.1.1** computes a distance matrix d, where the element d_{ij} represents the pairwise distance between two atoms P_i and P_j, thus:

$$d = \begin{bmatrix} d(P_1, P_1) & d(P_1, P_2) & \cdots & d(P_1, P_n) \\ d(P_2, P_1) & d(P_2, P_2) & \cdots & d(P_2, P_n) \\ \vdots & \vdots & \ddots & \vdots \\ d(P_n, P_1) & d(P_n, P_2) & \cdots & d(P_n, P_n) \end{bmatrix} \tag{3.11}$$

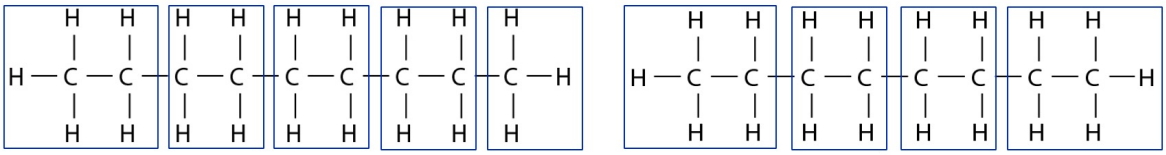


Figure 3.2: Segmentation of Alkane Chains: An Algorithmic Approach **A.1.2** to Identifying Groups within Nonane(C_9H_{20}) and Octane(C_8H_{18}) with $\mathbf{n_{heavy}}$=2

For each group of heavy atoms, the **Algorithm A.1.2** aims to identify chunks within $\mathbf{n_{heavy}}$ in the whole molecule, specifically looking for hydrogen atoms within a threshold distance d_{ij} from any atom in the heavy atom group. Upon identifying these segments, the algorithm consolidates each group, containing both the heavy atom or atoms and

their proximate hydrogen atoms, sorts them, and then incorporates these sorted groups into the overall list of atom groups. The **Figure 3.2** shows a grouping algorithm that categorizes the carbon atoms into slices, with each slice containing two carbon atoms, as indicated by the rectangles drawn around pairs of carbon atoms. The molecule nonane (C_9H_{20}) comprises a total of 5 slices, denoted as $\mathbf{n_{slice}}$, while the molecule octane (C_8H_{18}) comprises a total of 4 slices.

Notation	Description in the pseudo algorithm
α, β	atomic orbital functions
x, y	auxiliary basis functions
μ, ν	localized molecular occupied orbitals
A, B	atomic orbital functions within a specific segment
X, Y	auxiliary basis functions within a specific segment
V	localized molecular occupied orbitals within a specific segment

Table 3.1: List of symbols used in JK-Engine calculations

After slicing the molecule, the next step involves computing the short-range three-centered integrals for these segments, denoted as $(AB|X)_{sr}$, where A, B, and X represent the specific atomic orbitals and auxiliary orbitals within the slice, shown in **Table 3.1**. For the exchange algorithm, it is necessary to preliminary compute the short-range integrals $(\alpha B|x)_{sr}$ slicing along the β dimension. These three-dimensional integrals are aggregated by $(AB|X)_{sr}$ along the α and x dimension, then transformed into two-dimensional sparse matrices, denoted as $(\alpha x|B)_{sr}$. These matrices are eventually stored on a hard disk.

$$(\alpha B|x)_{sr} = \cup_{j=1}^{n_{slice}} \cup_{k=1}^{n_{slice}} (A_j B|X_k)_{sr} \tag{3.12}$$

The **Algorithm A.1.4** details the process of aggregating all segmented integrals $(AB|X)sr$ into larger segmented integrals, $(\alpha B|x)sr$, through the union of segments.

3.1.2 Conversion from Tensor to Sparse Matrix

Currently, the Python packages SciPy and NumPy do not offer a robust solution for the multiplication of three-dimensional and higher-dimensional sparse matrices. Although the 'einsum' function in NumPy allows for summation across dimensions in dense matrices, this method is not efficient enough for our specific needs. To address this challenge, Haobo Liu introduced a Tile structure by iterating over each auxiliary orbital x with three-index tensors [54]. In our innovative algorithm, we suggest using a new data structure to encapsulate these higher-dimensional tensors as sparse 2D matrices, with iteration over each block of heavy atoms.

The novel method for managing these higher-rank tensors entails converting them into 2D dense matrices through index reorganization. After the computation of $(AB|X)_{sr}$,

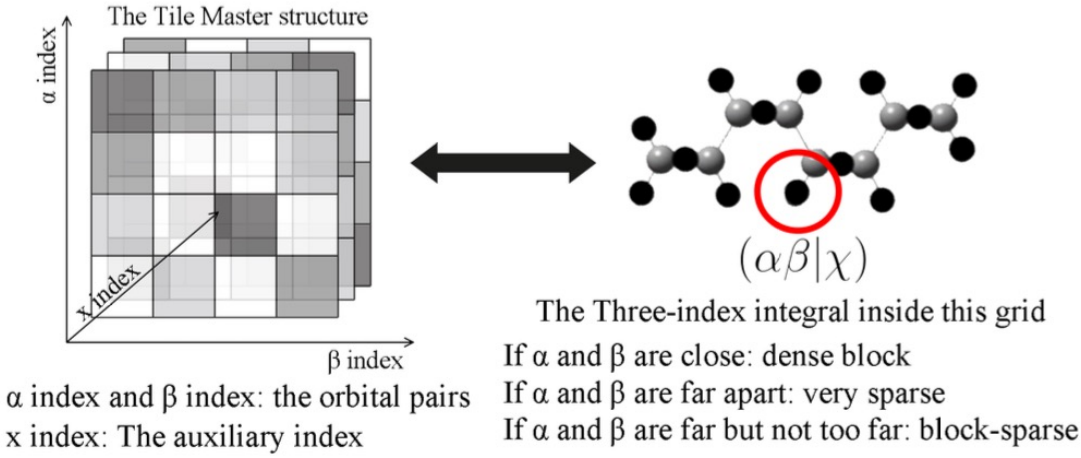


Figure 3.3: The three-centered block-sparse tensor quantity $(\alpha\beta|x)$ in preliminary stage[8].

these segment-specific three-dimensional tensors are first converted into sparse matrices via index transformation. Specifically, the indices α and β are amalgamated to act as a row index, while the index of x is designated as a column index. To represent the transformed compounded matrix $(AB;X)_{sr}$, we will introduce a new notation by employing a semicolon for its representation.

$$idx_{row} = idx_{\alpha} \cdot nao + idx_{\beta} \tag{3.13}$$

$$idx_{coln} = idx_{x} \tag{3.14}$$

3.1.3 Sparse Matrix Format and Storage

During implementation, some valuable contributions in our algorithms, such as short-range three-index integrals $(\alpha\beta;x)$ and some intermediate integrals, are always conducted in a large block-sparse matrix format, making computations expensive during operations. The sparse-block matrix has high sparsity, even up to 95%, and only 5% of elements have non-zero values in dense blocks. Specifically, an appropriate data structure for a block-sparse matrix, which includes element organization, conduction, and storage format, allows the algorithms of matrix operations like multiplication to be precise and economical.

Considering the importance of operational efficiency and memory utilization, the sparse matrix format is an important feature of the implementation. There are three basic data structures to store sparse matrix, COO, CSR, and CSC. The COO format is the most straightforward method where it only stores the information of non-zero elements. This includes the value of these elements along with their respective row and column indices, shown in **Figure 3.4**.

Although the COO format is user-friendly and easy to understand due to its straightforward storage of non-zero elements, it is not as efficient for matrix operations like multi-

plication or inversion. In contrast, the CSR format is much more economical for arithmetic operations and row slicing, whereas CSC format is optimized for column-wise operations. The CSR contains three arrays, shown in **Figure 3.5, 3.6**, an array of non-zero values, an array of column indices corresponding with non-zero elements, and an array of row index pointers. These row index pointers indicate the start index within the value array for each row, enhancing its efficiency for row-oriented operations. The CSC format utilizes a similar strategy, shown in **Figure 3.7, 3.8**.

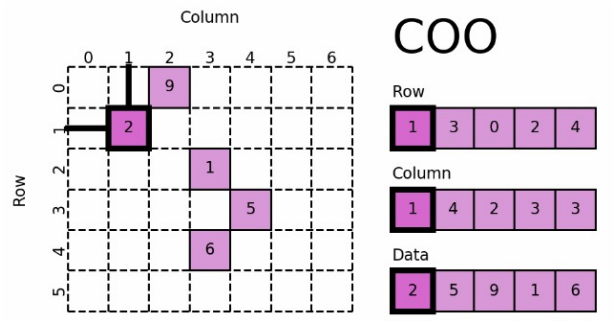


Figure 3.4: The COO format is utilized to represent non-zero elements within a sparse matrix. For instance, for a non-zero element valued at 2, it records the row index as 1 and the column index as 2 [13].

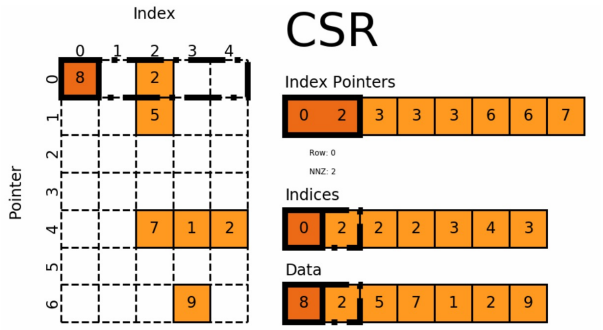


Figure 3.5: The CSR format is utilized to represent non-zero elements within a sparse matrix. For instance, for a non-zero element valued at 2, it records the row index as 1 and the index pointer as 2 [13].

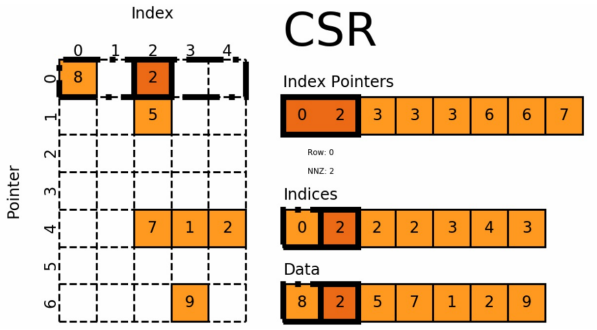


Figure 3.6: Another representation of non-zero elements in a sparse matrix using the CSR format [13].

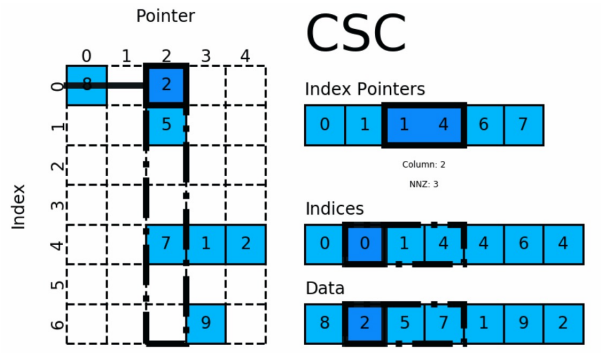


Figure 3.7: The CSC format is utilized to represent non-zero elements within a sparse matrix. For instance, for a non-zero element valued at 8, it records the column index as 0 and the index pointer as 1 [13].

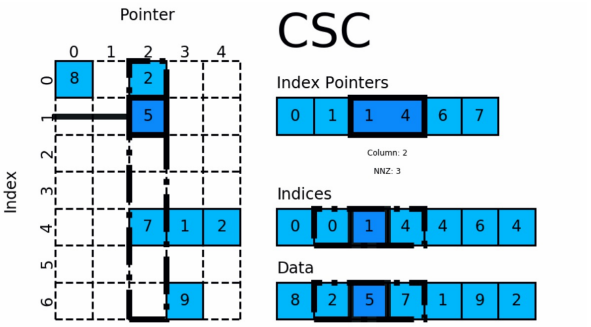


Figure 3.8: Another representation of non-zero elements in a sparse matrix using the CSC format [13].

In our algorithm, the intermediate short-range integrals $(\alpha B; x)$ will be stored on a hard disk in the COO format. This format is selected for its convenience in updating indices during the aggregation process. Additionally, as part of our preprocessing steps after converting from a dense to a sparse matrix, we implement a strategy to eliminate zero elements that fall below a certain threshold. This step is crucial for addressing numerical precision issues for tiny values (which are smaller than 1×10^{-5}) and optimizing storage efficiency. This elimination process is executed within our algorithm after every conversion of a dense matrix to a sparse matrix. **Algorithm A.1.3** details the specific steps involved in removing zero entities.

According to **Equation 3.12**, it is necessary to transform the row and column index of non-zero elements of each sparse matrix slice $(AB; X)_{sr}$, during the aggregation of these sparse matrices into a larger sparse matrix representing the entire $(aB; x)_{sr}$ integral set. The row indices for each slice's sparse matrix $(ABX.row)$ are adjusted by adding an offset, equal to the cumulative sum of the number of atomic orbitals along α dimension (n_A) in previous slices multiplied by the number of atomic orbitals along β dimension (n_B) in the slice, effectively mapping the local slice-based row indices to the global integral matrix (idx_{global_row}). In contrast, the column indices $(ABX.col)$ are adjusted by adding an offset equal to the cumulative sum of the number of auxiliary basis functions (n_X) in previous slices, mapping the local slice-based column indices to the global column indices (idx_{global_coln}) in the combined sparse matrix.

$$idx_{global\_row} = ABX.row + (\sum_{i=0}^{k-1} n_{A_i}) \times n_{B_k} \tag{3.15}$$

$$idx_{global\_coln} = ABX.coln + (\sum_{i=0}^{k-1} n_{X_i}) \tag{3.16}$$

This indices adjustment ensures that the local, slice-based indices are accurately mapped

to their global counterparts within the overarching sparse matrix. The object of this process is the construction of $(aB;x)_{sr}$, a COO format sparse matrix that embodies the aggregated integrals, ensuring that the structure and relationships inherent in the local matrix slices are preserved and represented on a global sparse matrix framework. This index transformation and aggregation strategy are pivotal features for the efficient computation and storage of the $(aB;x)_{sr}$ integrals, illustrated in **Algorithm A.1.4**.

In the exchange algorithm, preparation of the short-range integral slice, denoted as $(\alpha x;B)_{sr}$, is also required. Here, the indices α and x are combined to form a single row index, while the index B is extracted as a column index. The **Algorithm A.1.5** explains how the index re-compounds from $(\alpha B;x)$ to $(\alpha x;B)$.

Firstly, it generates a two-dimensional grid of indices for dimensions $'\alpha'$ and $'B'$, mapping each (α, B) pair to a unique linear index. This step uses *np.meshgrid* to create a coordinate matrix for these dimensions within the given sizes $'nao'$ and $'n'_B$. Secondly, for each non-zero element in the input matrix $(\alpha B;x)$, denoted as Z_{coo} in the algorithm, it identifies the corresponding $'\alpha'$ and $'B'$ values using the linear indices derived from the matrix's row indices. The *np.divmod* function is then applied to these selected linear indices, with n_B (the size of dimension $'B'$) as the divisor. This operation performs integer division and modulus simultaneously. The quotient (idx_a) represents the $'\alpha'$ index, and the remainder (idx_B) represents the $'B'$ index for each non-zero element.

Thirdly, the new row indices are re-compounded by combining these $'\alpha'$ values with the column indices of $(\alpha B;x)$, scaled by a factor $naux$. This operation effectively reorganizes the data, converting from an $(\alpha B;x)$ to $(\alpha x;B)$ configuration. Finally, a new COO matrix is constructed using these recalculated row indices and the original $'B'$ column indices, with the same non-zero data elements as the input matrix but with an updated shape to reflect the new arrangement. This function allows for the reorganization of sparse matrix representations, accelerating operations and analyses that benefit from the alternative structure.

At this stage, the necessary preparatory steps for computing the exchange matrix $K_{\alpha\beta}$ and Coulomb matrix $J_{\alpha\beta}$ are complete. The slice integrals, specifically in the COO format for $(\alpha B;x)$ and $(\alpha x;B)$, have been efficiently stored on the Hard Disk Drive (HDD).

3.2 JK-Engine: Exchange Matrix Algorithm

The exchange matrix $K_{\alpha\beta}$ plays a pivotal role in quantum chemistry, particularly in the HF method and post-HF calculations, where it contributes to the electron-electron repulsion and exchange phenomena. The exchange matrix $K_{\alpha\beta}$:

$$F_{\alpha\beta} = H^{core}_{\alpha\beta} + 2J_{\alpha\beta} - K_{\alpha\beta} \tag{3.17}$$

$$K_{\alpha\beta} = \sum_{\gamma,\delta} (\alpha\gamma|\beta\delta) D_{\gamma,\delta} \tag{3.18}$$

where $D_{\gamma\delta}$ is the electron density matrix. To reduce computational complexity, the matrix-RI approximation with short-range integrals is used for the exchange calculations, given

by:

$$K_{\alpha\beta} = \sum_{\gamma,\delta} \sum_{x,y} (\alpha\gamma|x)_{sr} Q_{xy} (\beta\delta|x)_{sr} D_{\gamma,\delta} \tag{3.19}$$

where $(\alpha\gamma|x)_{sr}$ are the short-range three-center integrals, and Q_{xy} is the metric matrix with the RI approximation.

The JK-Engine class is designed to offer a complete framework for calculating the exchange contribution, ensuring users select the most suitable approach for their specific requirements. It includes four distinct algorithmic approaches for exchange calculations, each designed to accommodate different requirements for memory usage and timing costs. To cater to various computational demands, the class offers four distinct algorithmic approaches for exchange calculations. Here is an overview of these approaches:

1. Exchange Slice Method: This technique involves slicing atomic orbitals according to a group of heavy atoms, identified by the parameter $\mathbf{n_{heavy}}$, and having a flexible number of grouping atoms ($\mathbf{n_{gatm}}$). It utilizes sparse matrix operations to perform the targeted exchange calculations. This method is the most effective for optimizing computations in large-scale systems with a number of heavy atoms, allowing for a more efficient calculation process.

2. Grouped Atom Exchange Slice Method: Similar to the Exchange Slice Method, this approach slices atomic orbitals by a specific group of atoms. However, it distinguishes itself by using a fixed number of atoms in each group ($\mathbf{n_{gatm}}$), combined with sparse matrix operations.

3. Sparse Exchange Method: This method adopts a general sparse matrix framework of the whole molecular system, balancing between efficiency and computational demand.

4. Dense Exchange Method: Applies a dense matrix methodology, this method is best suited for systems where the advantages of sparse matrix representations are minimal, such as in smaller molecular systems. This method ensures that calculations are straightforward, highly accurate, but they are also computationally expensive.

This flexibility ensures that users can select an approach that matches their computational resources, whether speed, accuracy, or memory usage, thereby enhancing the efficiency of their HF calculations. Algorithms 2-4 were developed in the course of this work, but the ultimate exchange slice method stands out as the most efficient strategy, combining the use of sparse matrix formats with flexible memory-efficient operations. Hence, this method will be employed to compute exchange matrices throughout our research project. We will explain in more detail how to calculate exchange contributions.

Initially, it is essential to segment the LMOs $C_{\beta,\mu}$ along both the AO, β dimension and the LMO μ dimension. The way to segment the LMOs in **Algorithm A.2.1** prepares for the exchange algorithm, which iterates through all the slice blocks $(\alpha x; B)$ spanning the β dimension, focusing on a selected subset of LMOs for the entire molecular ensemble.

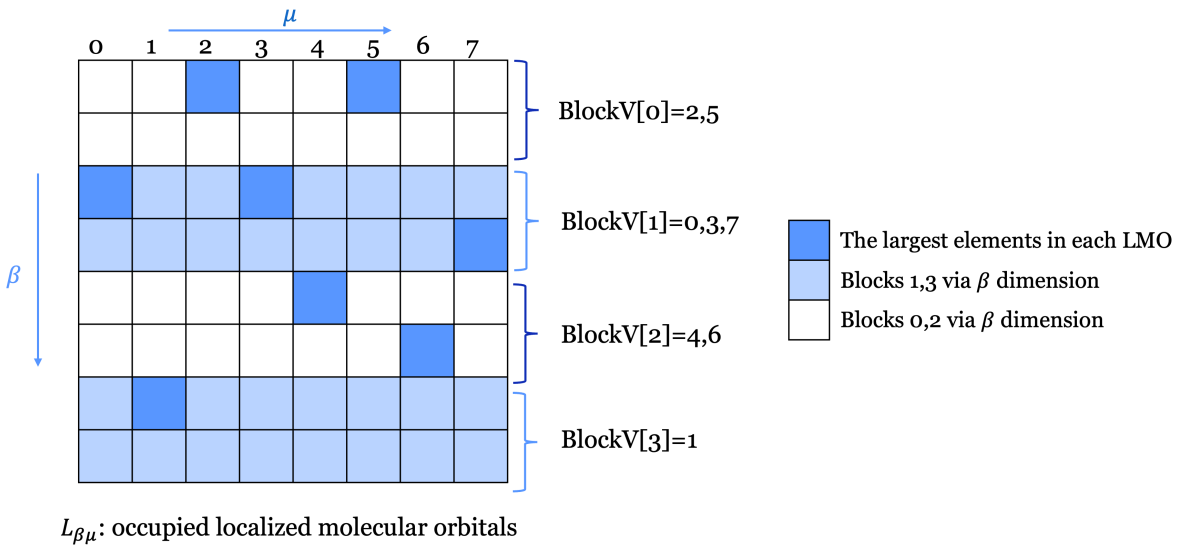


Figure 3.9: Sketch of Localized Molecular Orbital Grouping Strategy. This figure displays the method of segregating LMOs according to the β index of the most significant value in each LMO column.

Specifically, we construct two dictionaries, where they have the same key as the block index ($BlockIdx$), and the different values. One value, denoted as $BlockB$, holds the boundaries for AO along the β direction for each block, while the other value, $BlockV$, records the corresponding LMOs indices associated with each block. For each group of slice block $(\alpha x; B)$, it calculates the starting ($b0$) and ending ($b1$) AO's indices for the molecular slice associated with the first and last atom in the group, respectively. These AO indices are used to define the boundaries of a block ($BlockB$) and to associate LMOs' indices with the block ($BlockV$), shown in **Figure 3.9**. The association is based on whether the row (AO) index of the maximum value for a LMO falls within the AO boundary of the block. This **Algorithm A.2.1**, by identifying block boundaries and corresponding LMOs indices, plays the groundwork for focused computational analysis on specific target molecular regions.

Algorithm 9 Calculation of Exchange Integrals **K** by slicing

Function getExchangeSlice($int3cFolder$, Q_{xy}, dm, $threshZ$)
Initialize variables: $thresh\_Z \leftarrow 1e-5$, $L \leftarrow$ pivotedCholesky(dm)
Obtain dimensions nao, $naux$, $natm$, $ngatm$, and convert Q_{xy} to sparse matrix
Initialize $K_{\alpha\beta}$ as a sparse matrix of shape (nao, nao)
Get AO boundaries and associated LMO indices using $getBlocksBV(L)$
for each block V index in $blockV$ **do**
    $LMO_V \leftarrow BlockV[Idx_V]$ ▷ Assciated LMO index in each block V
    $L_{\beta V} \leftarrow L[:, LMO_V]$
    $I(\alpha x; V) \leftarrow$ CSC($nao * naux, len(L_V[0])$)
    for each atom block boundary index (β_0, β_1) in $blockB$ **do**
        $L_{BV} \leftarrow$ CSC($L_{\beta v}[\beta_0 : \beta_1, :]$) & EliminateZeros($L_{BV}, threshZ$)
        if $max(abs(L_{BV})) > threshZ$ **then**
            $(\alpha x, B) \leftarrow$ LoadSparseMatrix(int3cFolder, B)
            $I(\alpha x, V)$ += CSC$(\alpha x, B) \cdot L_{BV}$ & EliminateZeros($I(\alpha x, V), threshZ$)
        end if
    end for
    for $\nu \leftarrow 0$ **to** $range(len(L_V[0])$ **do**
        $I(\alpha x;) \leftarrow I[:, \nu]$
        $I(\alpha; x) \leftarrow$ Reshape($I(\alpha x;), nao, naux$)
        $Z \leftarrow$ CalculateSparsityMask($I(\alpha; x), threshZ$)
        $\tilde{Q} = Z * Q_{xy} * Z^T$ ▷ Hadamard Product
        $I_2 = \tilde{Q} \cdot I(\alpha; x)^T$
        $K_{\alpha\beta}$ += $I(\alpha; x) \cdot I_2$
    end for
end for
return K_{ab}

Figure 3.10: Pseudo Algorithm of Exchange Matrix $K_{\alpha\beta}$ by slicing heavy atoms (same as **Algorithm A.2.2**).

The primary algorithm for exchange, as illustrated in **Figure 3.10**, involves several steps. It begins by identifying the LMO through the application of a pivoted Cholesky decomposition to the density matrix (DM), represented by the equation $D_{\alpha\beta} = \sum_{\mu} L_{\alpha\mu} L_{\beta\mu}$. As the intermediate integral $I(\alpha x; \mu)$ is too large to fit in RAM, the algorithm segments the computation into block processing. According to the previous setup described in **Algorithm A.2.1**, we can get two dictionaries: one outlining the AO boundaries, denoted as $BlockB$, and another associated LMO indices with every block, denoted as $BlockV$.

The corresponding intermediate integrals $I(\alpha x; V)$ can be evaluated by:

$$I(\alpha x; V) = \sum_{B} (\alpha x; B)_{sr} L_{BV} \tag{3.20}$$

It can be seen that for each block V, one needs to read in all relevant 3-index integrals. For this reason we want to minimize the number of blocks. However, the intermediate $I(\alpha x; V)$ needs to fit completely in RAM, and this sets a limit on the block size of each V. By grouping the LMOs in groups that are localized on particular atom blocks B, we can skip certain block$B$ and this increases computational efficiency. Below we discuss the process in more detail.

For each block V, the algorithm computes the subset of the LMO. Then, it aims to load the converted sliced three-centered integrals from HDD, represented as $(\alpha x; B)_{sr}$ and compute intermediate integrals using the corresponding LMO slice. To optimize the CPU time, the algorithm incorporates a selection mechanism for loading these short-range integrals. This process involves slicing the LMO along the AO β dimension, denoted as L_{BV}, and evaluating the largest absolute value within the LMO segment. If this maximum absolute value is larger than a predefined threshold, $threshZ$, the algorithm proceeds with loading the respective short-range integrals for further computation. Conversely, if the value falls below the threshold, it skips the calculation for that particular block, only focusing on significant contributions.

The final computation of $K_{\alpha\beta}$ uses the intermediate integrals $I(\alpha x; V)$ for each LMO column, represented as $I(\alpha x;)$. This array is then transformed into a two-dimensional sparse matrix, $I(\alpha; x)$, which is convenient for further matrix operations.

$$\begin{aligned} K_{\alpha\beta} +&= \sum_{xy} I(\alpha x) Q_{xy} I(\alpha y)^T \\ +&= \sum_{xy} I(\alpha x) \tilde{Q}_{xy} I(\alpha y)^T \end{aligned} \tag{3.21}$$

$$\tilde{Q}_{xy} = Z \odot Q_{xy} \odot Z^T \tag{3.22}$$

To optimize the computation of matrix $K_{\alpha\beta}$, we employ a strategy of calculating sparsity mask in **Algorithm A.2.3**. This contains the construction of an array Z that will facilitate the conversion of a dense matrix Q_{xy} into a sparse matrix $\tilde{Q}_{xy}$. This array Z is composed solely of elements 1 and 0, where the value 1 in the array indicates the presence of non-zero columns in an intermediate matrix $I(\alpha; x)$. This approach is particularly beneficial

because it allows for the selective processing of only those elements that contribute to the final matrix, thereby reducing computational overhead.

All thresholds used in JK-Engine	**Default Value**
Short-range Integral Setting (α)	0.6
Identify Atom Groups ($threshG$)	3.0
Zero Elimination ($threshZ$)	1×10^{-5}
pivoted Cholesky ($threshZ$)	1×10^{-5}
sparsity Mask ($threshZ$)	1×10^{-5}
Matrix Inversion Threshold ($threshP$)	1×10^{-6}

Table 3.2: List of all thresholds in JK-Engine. These thresholds are integral to the operation of the JK-Engine, ensuring the balance between computational efficiency and the accuracy of Coulomb and Exchange matrix calculations.

3.3 JK-Engine: Coulomb Matrix Algorithm

The Coulomb matrix in the context of the HF method is another important component of the Fock matrix.

$$J_{\alpha\beta} = \sum_{\gamma,\delta} (\alpha\beta|\delta\gamma) D_{\gamma,\delta} \tag{3.23}$$

Similarly to RI-exchange $K_{\alpha\beta}$, the RI-Coulomb calculations can be encapsulated by short-range integrals $(\alpha\beta|x)_{sr}$:

$$J_{\alpha\beta} = \sum_{\gamma,\delta} \sum_{x,y} (\alpha\beta|x)_{sr} Q_{xy} (y|\gamma\delta)_{sr} D_{\gamma,\delta} \tag{3.24}$$

Our algorithm for $J_{\alpha\beta}$ efficiently operates by utilizing short-range integrals $(\alpha\beta|x)_{sr}$ and focuses on partitioning the computation into manageable slices corresponding to the number of heavy atoms.

The Coulomb **Algorithm A.3** firstly prepares by obtaining segments of the input molecular system and initializing an empty array for the total Coulomb matrix. It then iterates over all slices and determines the AO boundaries in every molecular slice. Also, the density matrix $D_{\alpha\beta}$ reshapes to the appropriate vector format for each DM slice, denoted as $D_{\alpha B;}$. Next, sparse matrices of three-centered integrals $(\alpha B|x)_{sr}$ are loaded from a specified folder in the HDD. These preliminary steps are crucial for the intermediate matrix $I2(x;)$ through matrix manipulations, transforming the loaded integrals into a format that can be directly used in the Coulomb matrix

$$I1(y;) += \sum_{x} \Big[\sum_{\alpha B} (\alpha B|y)^T D_{\alpha B;} \Big] \tag{3.25}$$

Algorithm 12 Calculation of Coulomb Matrix J in a Slice

Function getCoulombSlice($int3cFolder$, Q_{xy}, dm)
Obtain nao, $naux$, $natm$, $ngatm$
Initialize $groups \leftarrow identify\_atom\_groups$
$J_{\alpha\beta} \leftarrow$ EmptyArray$(nao, 0)$ ▷ Initialize total Coulomb matrix
for each $group$ in $groups$ **do**
    $(p0, \_) \leftarrow mol\_slice(group[0])$ ▷ Get start basis index for group
    $(\_, p1) \leftarrow mol\_slice(group[-1])$ ▷ Get end basis index for group
    $D_{\alpha B;} \leftarrow$ ReshapeAsVector$(dm[:, p_0 : p_1])$
    $(aB; x) \leftarrow$ LoadSparseMatrix$(int3c_folder, i)$
    $I1(y;) \mathrel{+}= \cdot(aB; x)^T \cdot D_{\alpha B;}$ ▷ Update intermediate matrix
end for
$I2(x) = Q_{xy} \cdot I1(y;)$
for each $group$ in $groups$ **do**
    $(p0, \_) \leftarrow mol\_slice(group[0])$
    $(\_, p1) \leftarrow mol\_slice(group[-1])$
    $n_B \leftarrow p_1 - p_0$
    $(aB; x) \leftarrow$ LoadSparseMatrix$(int3cFolder, i)$
    $J_{slice} \leftarrow (aB; x) \cdot I2(x;)$
    $J_{a;B} \leftarrow$ Reshape(nao, n_B)
    $J_{\alpha\beta} \leftarrow$ Concatenate$(J_{\alpha\beta}, J_{a;B})$ ▷ Concatenate along columns
end for
return $J_{\alpha\beta}$

Figure 3.11: Pseudo Algorithm of Coulomb Matrix $J_{\alpha\beta}$ by slicing heavy atoms (same as **Algorithm A.3**).

Next, multiply with metric matrix Q_{xy} outside of the iterations:

$$I2(x) = \sum_y Q_{xy} \cdot I1(y;) \tag{3.26}$$

Then, we start the second loop for each slice of molecular segments, computing $J_{\alpha\beta}$ by multiplying the corresponding sliced integral $(\alpha B; x)$ with the previously calculated intermediate integrals $I(x;)$. This operation is performed for each atomic slice to generate a portion of the Coulomb matrix.

$$J_{\alpha B;} = (\alpha B; x)_{sr} I(x;) \tag{3.27}$$

Finally, all computed slices $J_{\alpha B;}$ are converted to matrix format and concatenated along the β axis to form the complete Coulomb matrix $J_{\alpha\beta}$. This procedure aggregates the partial Coulomb matrix into the final matrix, which is a key output of the computation.

$$J_{\alpha\beta} = \cup_{j=1}^{n_{slice}} J_{\alpha B_j} \tag{3.28}$$

In the next chapter, we will investigate the computational efficiency and memory re-

quirements of JK-Engine.

Chapter 4

Linear-Scaling Exchange Matrix

To evaluate the computational performance of our algorithm, we will focus on three main features: accuracy, CPU process time, and memory consumption. Given that high-rank integral calculations are conducted by fragments across all dimensions, our 'int-class' and 'JK-Engine' classes are designed to be free from memory leaks. Thus, these algorithms are universally applicable to any molecular system of any size. In this chapter, we will concentrate on evaluating the performance of these algorithms using examples of weakly interacting polywater models, alkane chains, and cis-transoid polyacetylene chains, emphasizing how they scale with increasing molecular sizes.

The evaluation process begins with the completion of SCF procedures using conventional algorithms in PySCF, followed by obtaining the converged density matrix. Next, we perform a single calculation of the Coulomb and exchange matrices, as well as the energies, utilizing the JK-Engine. These results are then benchmarked against those obtained from PySCF.

4.1 Molecular System Setup

This section outlines the key components involved in Hartree-Fock (HF) calculations used for benchmarking, involving basis set, input molecular system, and important contributions for evaluating:

Basis set: cc-pVTZ or cc-pVDZ, with a particular focus on the cc-pVTZ basis set for its importance in accurately modeling these molecular systems.

Auxiliary basis set: cc-pvtz-jkfit

Systems for Benchmarking: To ensure efficiency, selected simulation systems must be large enough that some of the short-range integrals of $(\alpha\beta|x)_{sr}$ with a long distance between the center of an (α, β) pair and the center of the auxiliary basis function x approach zero. The following systems have been identified for benchmark analysis:

- (Weakly Interacting) Polywater Systems: This system provides a benchmark for understanding behavior in ideal molecular configurations, focusing on polywater models

where there is minimal or no interaction between the water molecules. In particular, the exchange matrix and exchange energy could be calculated in principle one monomer at a time. The critical test is to see if this scaling behaviour is reflected in the current general algorithm. The water dimer system acts as an exemplary model for testing our algorithm, by increasing the distance between the two oxygen atoms, $d_{O_1-O_2} = 20$Å. This method allows the short-range part of the potential to be effectively reduced to zero (below our threshold), making these extended molecular systems ideal for highlighting the capabilities and limitations of the algorithm.

- Alkane Chain (saturated carbon chain): Alkane chain examples (C_nH_{2n+2}) are hydrocarbons containing only single bonds between carbon atoms and are saturated with hydrogen atoms. Alkane chains in our tests are linear sequences of carbon atoms, connected without any branches and exclusively by single bonds, making them fully saturated. These models vary from a simple ethane (C_2H_6) structure to more complex chains extending up to $C_{30}H_{62}$, allowing for a comprehensive exploration of alkanes. The LMO's in linear alkanes are anticipated to be quite localized, which is important for the efficiency of the exchange algorithm.

- Cis-transoid Polyacetylene (unsaturated carbon chain): Cis-transoid polyacetylene examples (C_nH_{n+2}) represent a series of linear unsaturated hydrocarbons of increasing length, each featuring a special arrangement of carbon atoms linked by both single and double bonds. These models, ranging from a simple ethene (C_2H_4) structure to more complex chains extending up to $C_{30}H_{32}$, are designed for algorithmic testing to elucidate the impact of chain length in the conjugated system and algorithm computational expense. The LMO's in linear polyacetylenes are expected to be less localized than in alkanes, and we wish to see the effect on the efficiency of the exchange algorithm.

Figure 4.1: One example of water systems: water dimer molecular system and the distance between two oxygen atoms is 20Å

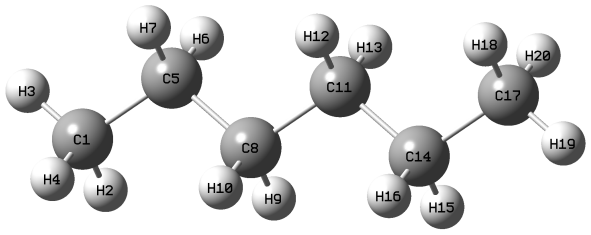


Figure 4.2: One example of alkane chains: C_6H_{14}

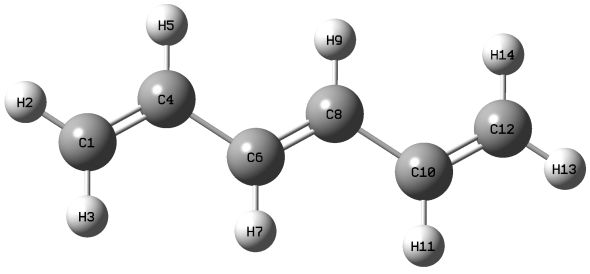


Figure 4.3: One example of cis-transoid polyacetylene chains: C_6H_8

Important Contributions for Benchmarking:

- Calculation of the K-matrix (Exchange matrix, $K_{\alpha\beta}$).
- Calculation of the J-matrix (Coulomb matrix, $J_{\alpha\beta}$).
- Reference benchmarking is conducted using PySCf with the density fitting (DF) approach, a method chosen for its effective error cancellation capabilities inherent in the resolution of identity (RI) approach. This involves the instantiation of a density-fitting object ($'dfobj = df.DF(Mol).build()'$).
- There are three critical stages in the computation of Exchange matrix using the JK-Engine. Each stage involves a series of detailed steps, which are disccussed in **Algorithm A.2.2**. To highlight the linear scaling property effectively, it is essential to analyze computational resources step by step. In the subsequent sections, we will discuss specific contributions in detail, shown in **Table 4.1**.

Initial Stage	
Descriptions	**Equation**
Short-Range Integral Slice	$(\alpha B; x)_{sr}$
Recompound Procedure	$(\alpha x; B)_{sr} \leftarrow (\alpha B; x)_{sr}$
Total Initial Stage	$(\alpha\beta; x)_{sr} \& (\alpha x; \beta)_{sr}$
Intermediate Stage	
Sparse Dot Multiplication for $I1$ Slice	$I(\alpha x; V) \mathrel{+}= \sum_B (\alpha x; B)_{sr} L_{BV}$
Excluding Zeros of $I1$ Slice	EliminateZeros$(I(\alpha x; V), threshZ)$
Total Intermediate Integral $(I1)$	$I(\alpha x; \mu)$
Build-K Stage	
Sparsity Mask Calculations for $I1$ per LMO	$Z =$ CalculateSparsityMask$I(\alpha; x)$
Transformed $\tilde{Q}_{xy}$ per LMO	$\tilde{Q}_{xy} = Z * Q_{xy} * Z^T$
Excluding Zeros of $\tilde{Q}_{xy}$ per LMO	EliminateZeros$(\tilde{Q}_{xy}, threshZ)$
Exchange $(K_\mu(\alpha\beta))$ per LMO	$K_{\alpha\beta} \mathrel{+}= I(\alpha; x)\tilde{Q}_{xy} I(\alpha; y)^T$
Total Exchange Slice Stage $(K_{\alpha\beta})$	$K_{\alpha\beta}$

Table 4.1: Summary of Contributions to the Analysis of Timing and Memory Usage in the JK-Engine

Illustrative example for benchmarking of parameter α

The optimal value of the α parameter for setting the reach of short-range regions depends on the task at hand, as detailed in Chapter 2. The timing for using the density fitting object from PySCF remains consistent across different α values. However, the time taken by the exchange and Coulomb operations in the JK-Engine decreases as the α parameter increases, as illustrated in **Figure 4.4** for the $C_{20}H_{42}$ molecular system. As seen the JK-engine can outperform The current DF algorithm implemented in PySCF. Conversely, the error in the exchange and Coulomb matrices between the JK-Engine and the DF Object within PySCF grows with an increase in the α parameter, as shown in **Figure 4.5**. The choice of α parameter for the short-range, three-centered integral is dictated by the specific type of simulations being conducted. Particularly, simulations that demand high accuracy benefit from a smaller α value, while simulations that can tolerate approximation tend to use larger α values to enhance efficiency. This chapter will explore various α parameters, such as $\alpha = 1.0$, $\alpha = 5.0$, and $\alpha = 10.0$, as examples to clarify their impact on accuracy, timing and memory usage.

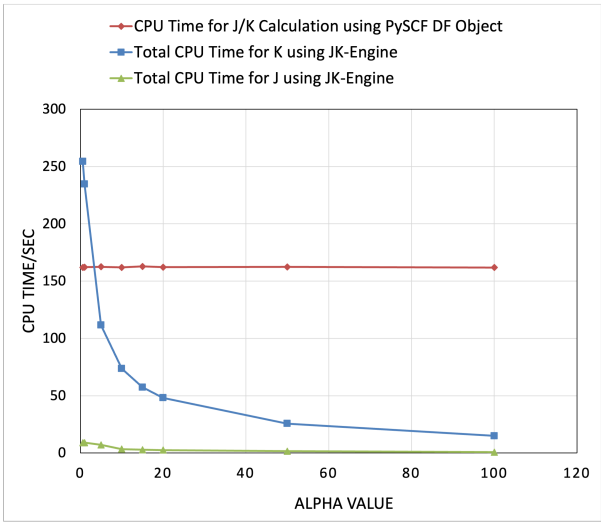


Figure 4.4: Comparative Analysis of CPU Time Usage Between JK-Engine Algorithm with the metric matrix $M_{\alpha\beta}$ and Density Fitting Object from PySCF for $C_{20}H_{42}$ under various α values in cc-pVDZ basis set and cc-pvtz-jkfit auxiliary basis set.

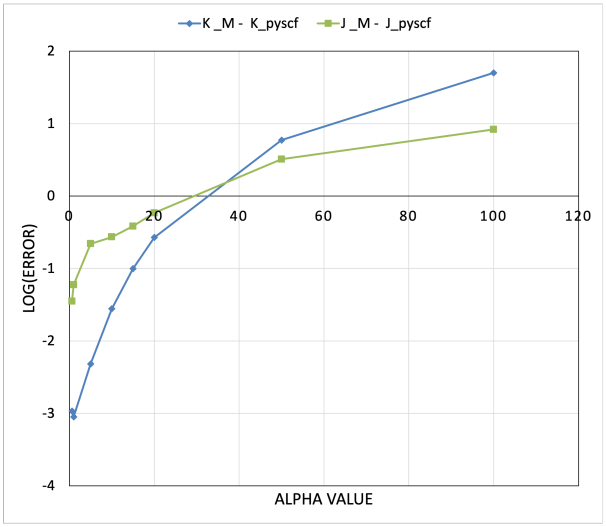


Figure 4.5: Comparative Analysis of Error Metrics Between JK-Engine Algorithm with the metric matrix $M_{\alpha\beta}$ and Density Fitting Object from PySCF for $C_{20}H_{42}$ under various α values in cc-pVDZ basis set and cc-pvtz-jkfit auxiliary basis set.

4.2 Linear-Scaling Property in Hartree-Fock Calculations

The performance of the JK-Engine algorithm is evaluated in terms of both time efficiency and memory consumption. Our analysis encompasses three molecular models: (weakly

interacting) polywater models, alkane chains, and cis-transoid polyacetylene chains, allowing us to thoroughly investigate the algorithm's linear scaling property in time and space complexity across these varied systems. Key contributions to the algorithm's performance are detailed in **Table 4.1**.

4.2.1 Timing Consumption in Exchange Algorithm

The timing consumption is the most straightforward analysis of computational resources. We will profile CPU time not just for the total J and K matrices but also for each step across the three main stages, as detailed in **Table 4.1**. We consider the weakly interacting polywater models system in great detail.

Polywater Models

The weakly interacting polywater models molecular system is the simplest and the most ideal case for analysis. This system is designed such that the calculation of the exchange matrix can be implemented in a linear scaling fashion, by just considering one water monomer at a time. The key question is to what extent we can achieve linear scaling using the JK-Engine that is applicable to general molecules. We assessed both CPU time and memory consumption for polywater systems ranging from a single water monomer up to sixteen monomers. To ensure consistency and eliminate outliers, each polywater model configuration was tested four times.

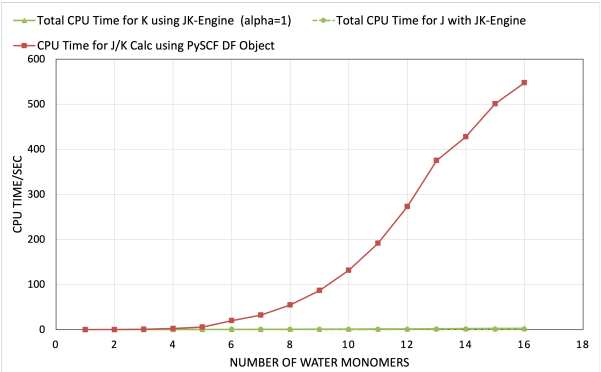


Figure 4.6: Preliminary Comparative CPU Time for J and K Calculations using PySCF DF Object, and JK-Engine for Polywater Models $(H_2O)_n$ in cc-pVTZ basis set and cc-pvtz-jkfit auxiliary basis set at α Parameters of 1.0.

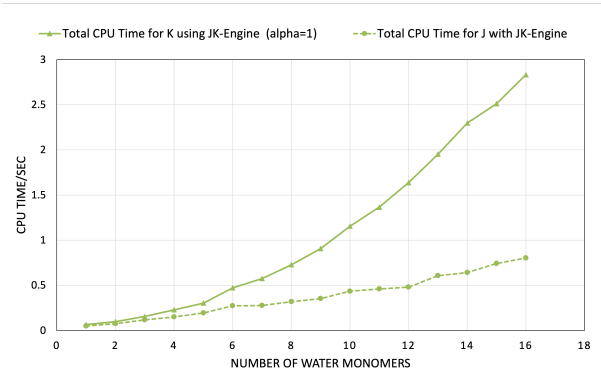


Figure 4.7: Preliminary Comparative CPU Time Analysis for J and K Calculations in Polywater Models $(H_2O)_n$ Using the JK-Engine with cc-pVTZ Basis Set and cc-pVTZ-jkfit Auxiliary Basis Set at α Parameters of 1.0.

We initially conducted a preliminary comparison of the performance between the JK-Engine and the PySCF DF approach. **Figure 4.6** and **Figure 4.7** present a comparison of CPU time required for J and K matrix calculations using both the JK-Engine and the

PySCF DF Object across a range of water monomer system sizes, from a single monomer up to sixteen. In the **Figure 4.6**, the CPU time for the J/K calculation using the PySCF DF object shows a significant quadratic increase as the number of water monomers increases, particularly noticeable beyond ten monomers. In contrast, the CPU time for calculations using our JK-Engine remains relatively flat for both J and K in green solid and green dash-dot lines, with a slight uptick in the case of K after five monomers. **Figure 4.7** offers a closer view of the CPU time for the polywater systems under $\alpha = 1.0$ value. For the JK-Engine, the time for J scales linearly with the size of the system, as anticipate, whereas the time for K is not quite linear. The time for K remains significantly lower than the PySCF DF object even for small polywater systems .

In summary, time consumption in **Figure 4.6** and **Figure 4.7** roughly summarizes the performance advantages of the JK-Engine over the PySCF DF object in terms of CPU time, particularly as the number of monomers increases. However, the JK-Engine does not show a consistent linear scaling in computational time for K, especially for larger water systems sizes. This is surprising and we will continue to more deeply analyze the operations within the JK-Engine. Let us explore the three stages within the JK-Engine, as outlined in **Table 4.1**.

At the initial stage, we generated slices of the short-range three-centered integrals $(\alpha B|x)_{sr}$ along the β dimension and saved these integrals on the hard disk. **Figure 4.8** depicts the linear timing relationship for calculating each $(\alpha B|x)_{sr}$ slice, a constant relationship for re-compound from $(\alpha B|x)_{sr}$ to $(\alpha x|B)_{sr}$, and a quadratic relationship for the total number of $(\alpha B|x)_{sr}$ slices. The quadratic relationship arises because the integral slice $(\alpha B|x)_{sr}$ iterates over all slices $(AB|X)_{sr}$ across both α and x dimensions without employing any screening techniques. Consequently, some integral blocks are computed and subsequently will be filtered in the next step. This initial process presents an opportunity for optimization through the exclusion of calculating zero blocks in future work. For the scope of this project, we will not delve further into this initial stage but will instead utilize pre-computed short-range integrals stored on the hard disk.

In detailed exchange matrix calculations (**Algorithm A.2.2**), we focus on two primary stages: calculating the intermediate contribution $I1(\alpha x|V)$ for each B block, and computing the exchange contribution for each localized molecular orbital $K_{\alpha\beta;\mu}$. This latter step we refer to as the build-K step, as in **Equation 4.2**. The time consumption associated with these two steps is illustrated in **Figure 4.9** and **Figure 4.10**, respectively. As is seen the calculation of the intermediate $I1$ shows a linear scaling behavior. To our surprise, the subsequent build-K step shows a non-linear behavior. There are a number of small subroutines into these two steps, and further analysis is needed to identify which step(s) cause a problem.

$$I(\alpha x; V) = \sum_{B} (\alpha x; B)_{sr} L_{BV} \tag{4.1}$$

$$K_{\alpha\beta} += \sum_{xy} I(\alpha x) \tilde{Q}_{xy} I(\alpha y)^T \tag{4.2}$$

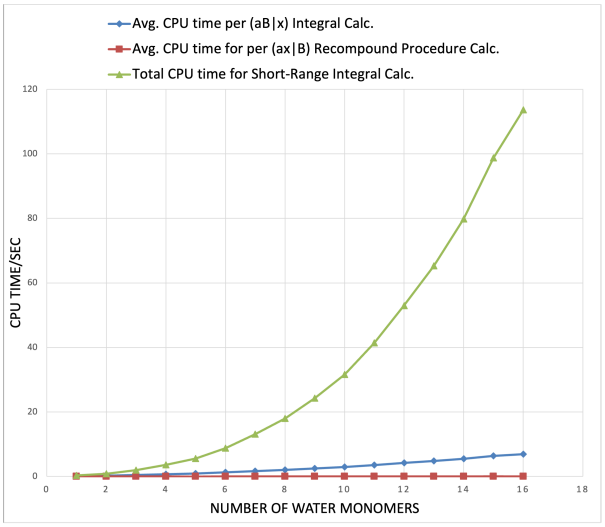


Figure 4.8: Average and Total CPU Time for Calculating Short-Range Integrals $(\alpha x|B)_{sr}$ at the Initial Stage in Polywater Systems $(H_2O)_n$ using the cc-pVTZ Basis Set and cc-pvtz-jkfit Auxiliary Basis Set ($\alpha = 1.0$)

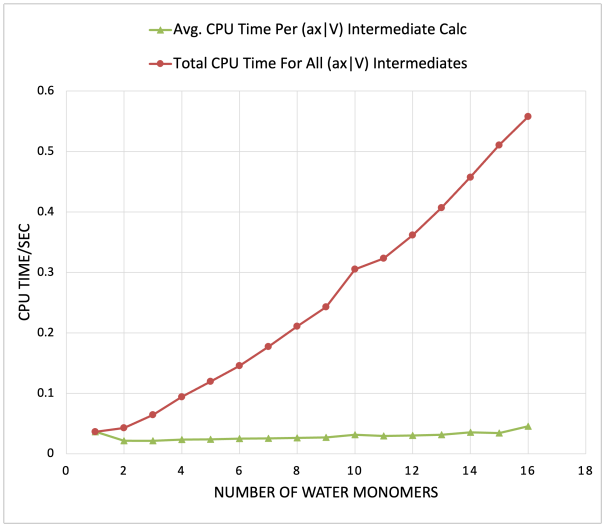


Figure 4.9: Average and Total CPU Time for Intermediate Integral $(\alpha x|V)$ Calculations in polywater Systems $(H_2O)_n$ using the cc-pVTZ Basis Set and cc-pvtz-jkfit Auxiliary Basis Set ($\alpha = 1.0$)

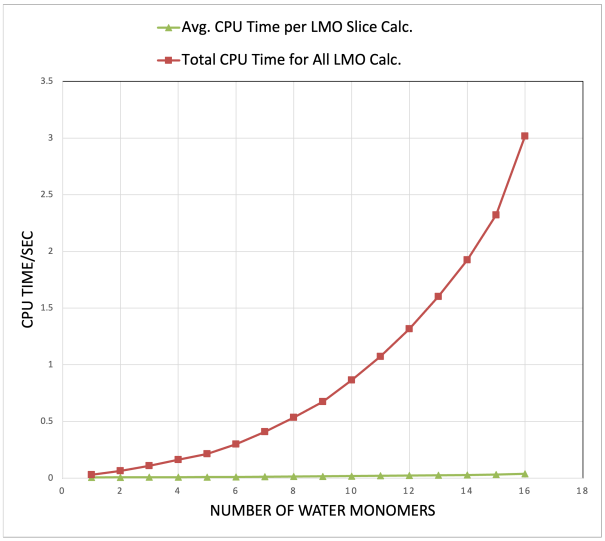


Figure 4.10: Average and Total CPU Time for each LMO Analysis for K-built in polywater Systems $(H_2O)_n$ using the cc-pVTZ Basis Set and cc-pvtz-jkfit Auxiliary Basis Set ($\alpha = 1.0$)

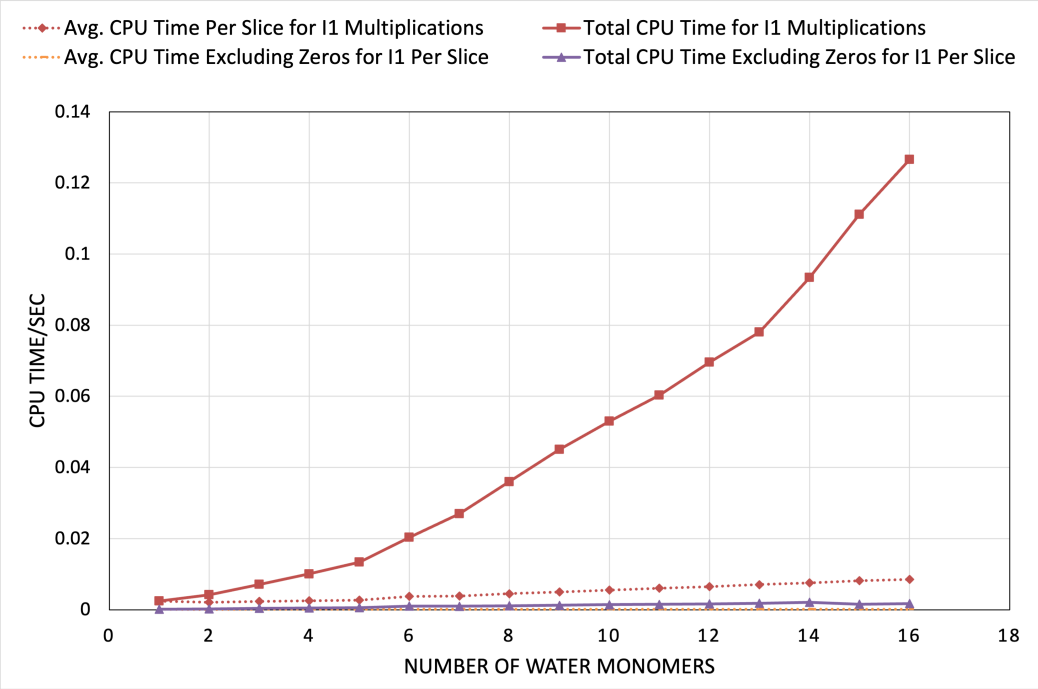


Figure 4.11: Comparative CPU Time Analysis for Intermediate Calculations in polywater Systems $(H_2O)_n$. The graph illustrates the average and total CPU times per LMO block (L_{BV}) for the calculations of intermediate integral $I1(\alpha x|V)$ and elimination of zero elements process of $I1$, as the number of water molecule count increases from one to ten (noted as $(H_2O)_1$ to $(H_2O)_{16}$) using the cc-pVTZ Basis Set and cc-pvtz-jkfit Auxiliary Basis Set ($\alpha = 1.0$).

To delve into the details of intermediate calculations, **Figure 4.11** illustrates the time consumed during sparse matrix dot product operations as outlined in **Equation 4.1**, and the process of removing zero elements from the sparse intermediate $I(\alpha x; V)$. These two steps are carried out after importing the three-centered integral $(\alpha B|x)$ and identifying its corresponding non-zero block in L_{BV} slice.

Moreover, from the **Figure 4.11**, it is clear that while the total CPU time for both processes increases with the number of water monomers, the average CPU time per slice remains relatively stable. This suggests that the computational effort for each individual slice does not significantly increase with system size for these steps. The linear increase illustrated by the solid red line from a single water molecule to system of 16 water molecules reflects a significant escalation in the total CPU time required for sparse matrix multiplication for $I1$. Additionally, the stability of the average time per slice, depicted by the dotted lines, shows the algorithm's consistent efficiency on a per-slice basis, even as the overall system grows. This observation suggests that while the total computational load increases with system size, the per-unit processing efficiency of the sparse matrix multiplication algorithm and zero-entries elimination algorithm remains unaffected, highlighting its potential benefits for larger systems.

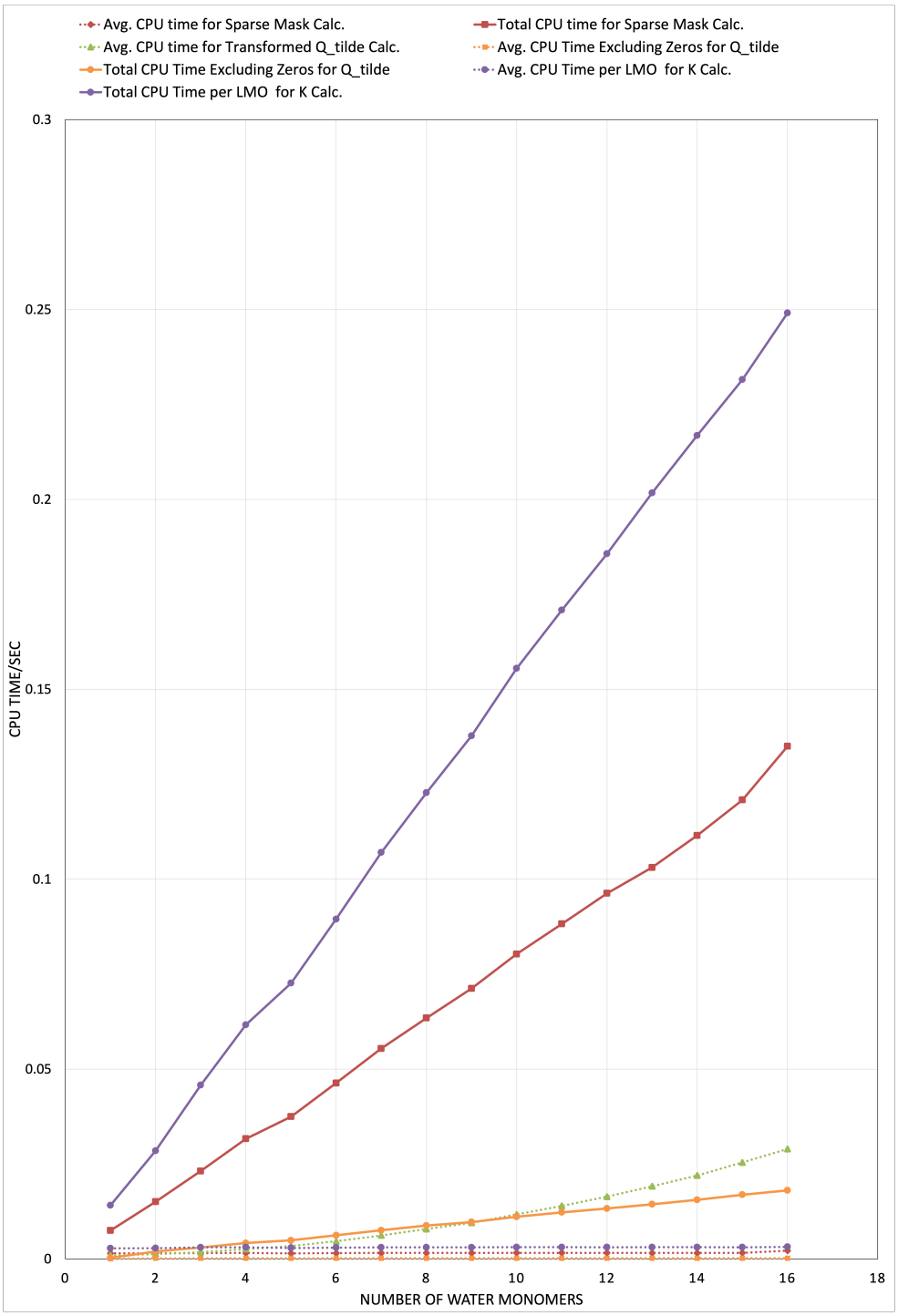


Figure 4.12: Comparative CPU Time Analysis for Calculations Polywater Systems $(H_2O)_n$. The graph illustrates the average and total CPU times per LMO for a variety of computational processes, including sparse mask algorithm, transformed $\tilde{Q}_{xy}$ calculations, exchange contribution $K_{\alpha\beta;\mu}$ calculations, and elimination of zero elements process of $\tilde{Q}_{xy}$, as the number of water molecule count increases from one to ten (noted as $(H_2O)_1$ to $(H_2O)_{16}$)using the cc-pVTZ Basis Set and cc-pvtz-jkfit Auxiliary Basis Set ($\alpha = 1.0$).

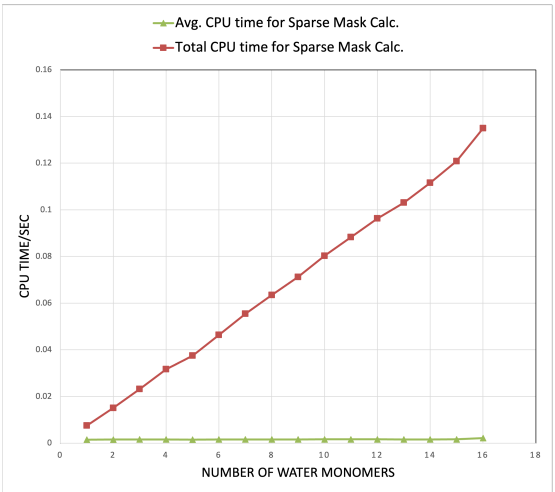


Figure 4.13: Average and Total CPU Time for the Per LMO Sparse Mask Process Calculations in Polywater Systems $(H_2O)_n$ using the cc-pVTZ Basis Set and cc-pvtz-jkfit Auxiliary Basis Set ($\alpha = 1.0$).

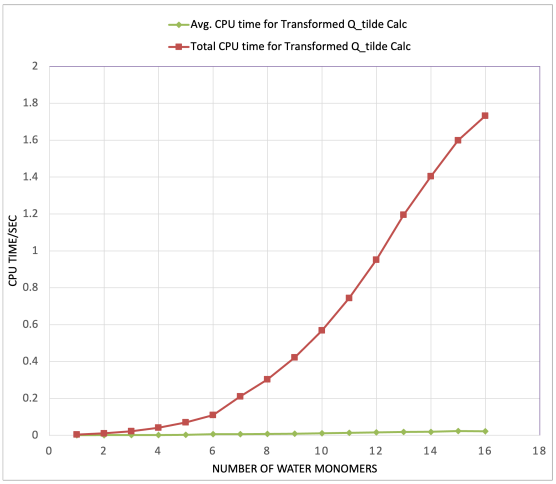


Figure 4.14: Average and Total CPU Time for the Per LMO Transformed $\tilde{Q}_{xy}$ Calculations in Polywater Systems $(H_2O)_n$ using the cc-pVTZ Basis Set and cc-pvtz-jkfit Auxiliary Basis Set ($\alpha = 1.0$).

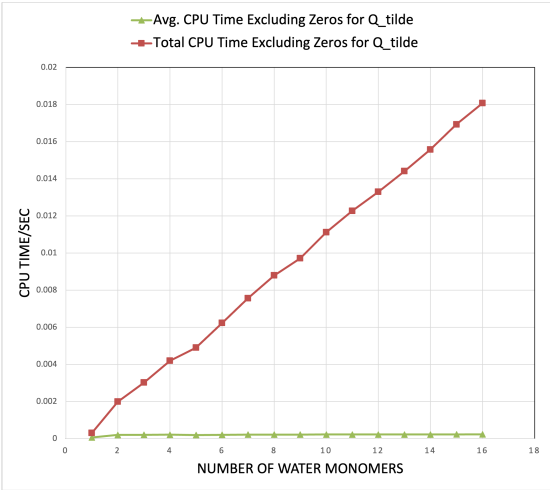


Figure 4.15: Average and Total CPU Time for Excluding Zeros in $\tilde{Q}_{xy}$ Per LMO Slice in Polywater Systems $(H_2O)_n$ using the cc-pVTZ Basis Set and cc-pvtz-jkfit Auxiliary Basis Set ($\alpha = 1.0$).

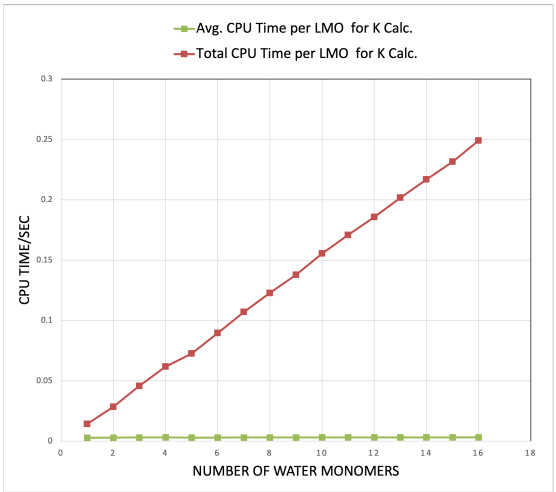


Figure 4.16: Average and Total CPU Time for the Exchange Contribution $K_{\alpha\beta;\mu}$ Calculations Per LMO Slice in Polywater Systems $(H_2O)_n$ using the cc-pVTZ Basis Set and cc-pvtz-jkfit Auxiliary Basis Set ($\alpha = 1.0$)

In another primary stage for evaluating the K-build, **Figure 4.12** aggregates the CPU time for key computational processes. This dataset spans multiple steps, with each step's details further illustrated in **Figure 4.13**, **Figure 4.14**, **Figure 4.15**, and **Figure 4.16**. These figures correspond to the calculation of the sparse mask, the transformation of $\tilde{Q}xy$, the elimination of zeros in $\tilde{Q}xy$, and the computation of the exchange contributions $K_{\alpha\beta;\mu}$, respectively. Specifically, the sparsity mask algorithm in **Algorithm A.2.3** refers to constructing a mask array Z to identify the non-zero columns in $I(\alpha; x)$. This mask is used to convert the dense Q_{xy} matrix into a sparse $\tilde{Q}_{xy}$ via a Hadamard product:

$$\tilde{Q}_{xy} = Z \odot Q_{xy} \odot Z^T = (Z \cdot Z^T) \odot Q_{xy} \tag{4.3}$$

Following the removal of zero elements in $\tilde{Q}_{xy}$, the exchange contribution for each LMO as follows:

$$K_{\alpha\beta} += I(\alpha x)\tilde{Q}_{xy}I(\alpha y)^T \tag{4.4}$$

We noticed the linear scaling behavior of each step as the system expands from a single water molecule to a system of sixteen in K-build procedures, except the non-linear scaling in transformed $\tilde{Q}$ from **Figure 4.14**. This step is attributed to the sparse Hadamard product performed between the dense matrix Q_{xy} and the sparse mask matrix $(Z \cdot Z^T)$, resulting in a more complex interaction that deviates from linearity, leading to a quadratic relationship. This step looks like a small innocent step in the algorithm, but it is the reason that we do not see linear scaling in the calculation of the exchange matrix for water systems. It can be improved using the following simple algorithm, given the binary matrix $Z \cdot Z^T$ to identify the row and column indices of non-zero elements. This involves the construction of new COO sparse matrix $\tilde{Q}$, using data from the dense Q matrix along with the indices derived from the binary $Z \cdot Z^T$ matrix. **Figure 4.17**, and **Figure 4.18** illustrates how the manually optimized Hardmard multiplication affect the efficiency of the transformed $\tilde{Q}$ matrix and K-built per LMO procedures. The optimization of the Hadamard product results in evident linear scaling in both cases.

To sum up, after the fix of the transformed $\tilde{Q}$ matrix, these detailed examinations of JK-Engine's stages from **Figure 4.19**, and **Figure 4.20** reveal a linear relationship in the exchange matrices computations, involving the initial calculation of short-range integrals, implementation of intermediate integrals, and evaluations of K-build per LMO. **Figure 4.20** illustrates the HF J and K components under different values of α, which overlap between each other.This overlapping is attributed to the activation of short-range integrals at short distances, indicating that the NNZ in short-range RI integrals remain relatively the same across different α parameters in non-interacting long-distance polywater systems. Additionally, there exists a single step that presents some challenges, but it is optimized by computing Hadamard product manually in non-interacting water systems. It should be noted that for other molecular models, the sparse Hadamard product implementation from the SciPy package was retained, as the problem was only removed at a very late stage of writing of this thesis. The analysis of an ideal polywater system lays a foundation for further analysis of alkane chains and cis-transoid polyacetylene chains.

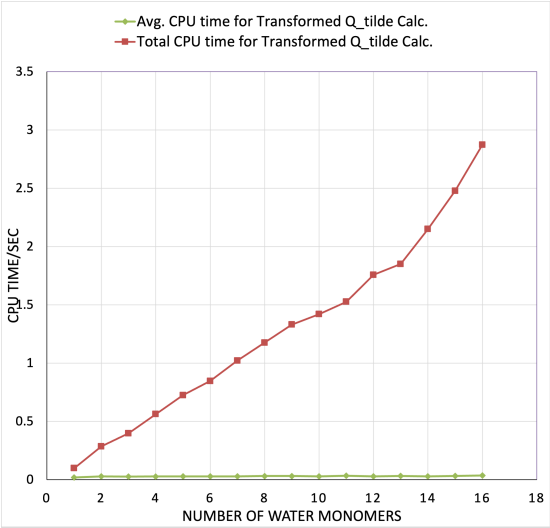


Figure 4.17: Average and Total CPU Time for the Optimized Transformed $\tilde{Q}_{xy}$ Calculations Per LMO in Polywater Systems $(H_2O)_n$ using the cc-pVTZ Basis Set and cc-pvtz-jkfit Auxiliary Basis Set ($\alpha = 1.0$).

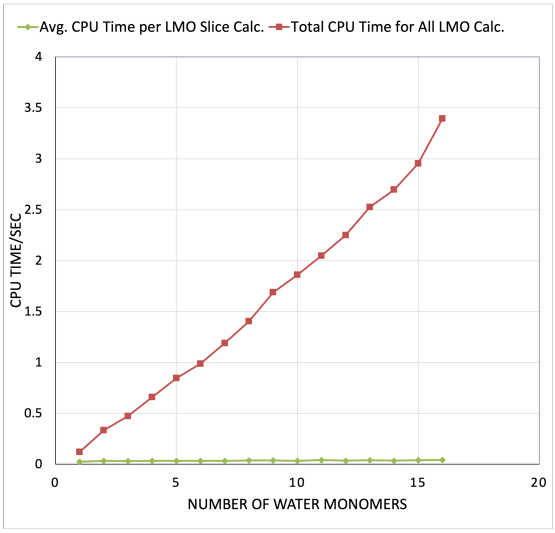


Figure 4.18: Average and Total CPU Time for Optimized K-built for each LMO in Polywater Systems $(H_2O)_n$ using the cc-pVTZ Basis Set and cc-pvtz-jkfit Auxiliary Basis Set ($\alpha = 1.0$)

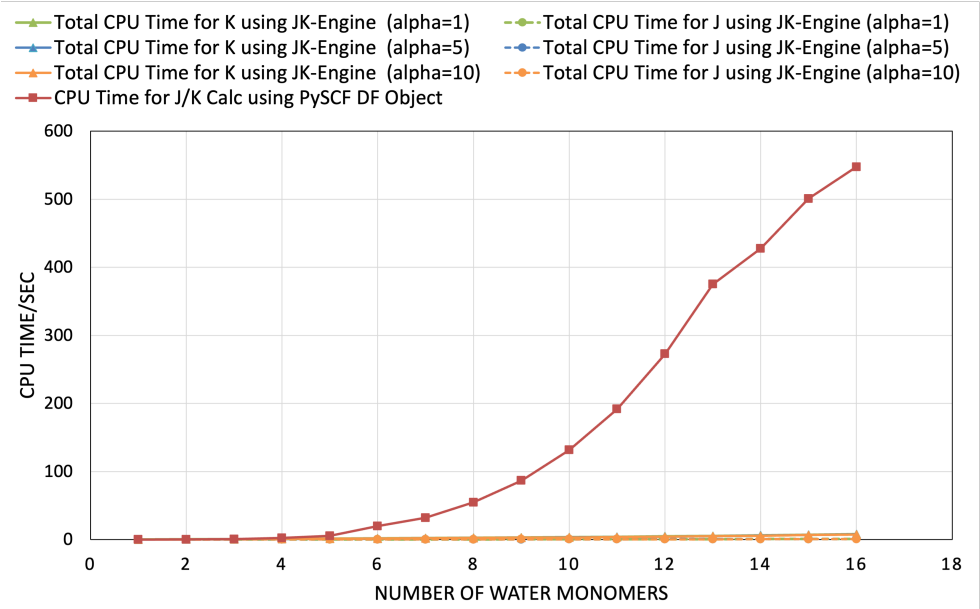


Figure 4.19: Comparative CPU Time Analysis for J and K Calculations in Polywater Systems $(H_2O)_n$ Using the JK-Engine with cc-pVTZ Basis Set and cc-pVTZ-jkfit Auxiliary Basis Set at α Parameters of 1.0, 5.0, and 10.0

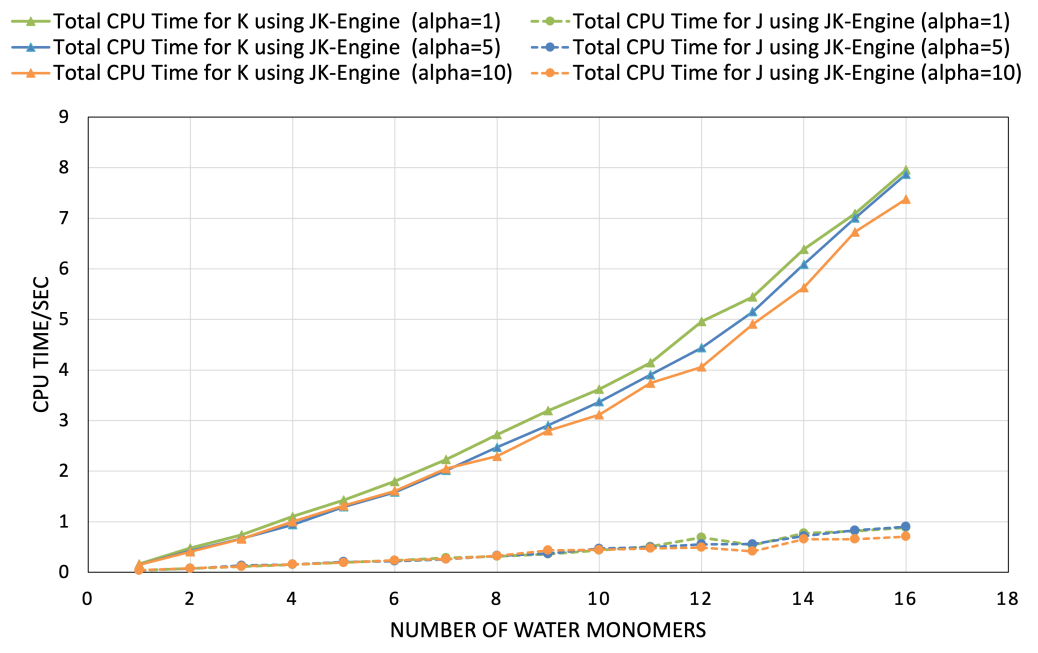


Figure 4.20: Comparative CPU Time Analysis for J and K Calculations in Polywater Systems $(H_2O)_n$ Using the JK-Engine with cc-pVTZ Basis Set and cc-pVTZ-jkfit Auxiliary Basis Set at α Parameters of 1.0, 5.0, and 10.0

Linear Alkane Chains

Figure 4.21 and **Figure 4.22** present a comparison of the total CPU time for computing J and K matrices, illustrated with green, blue, and orange lines, against the time taken using the PySCF DF method, depicted in red, across various values of α. It is evident that our JK-Engine significantly outperforms the PySCF DF approach, even in computationally demanding scenarios where $\alpha = 1.0$. The linear relationship of the JK-engine exchange algorithm becomes significantly clearer beyond 18 carbon atoms in green solid line of **Figure 4.22**.

Figure 4.23 to Figure 4.25 represent the CPU timing required for initial, intermediate, and exchange slices stages respectively. Despite the timing for short-range integral slices increasing quadratically, this step is pre-computed and stored in a hard disk, and will not affect calculations of exchange matrices. The other two stages in **Figure 4.24** **Figure 4.25** demonstrate a linear relationship when the carbon chains are sufficiently long resulting in a well-sparse matrix format. The intermediate stage begins to show a linear correlation with the system size starting at approximately 18 carbon atoms, while the build-K stage demonstrates this linear behavior from an earlier point, around 10 carbon atoms. We will discuss this later, supported by data on memory usage.

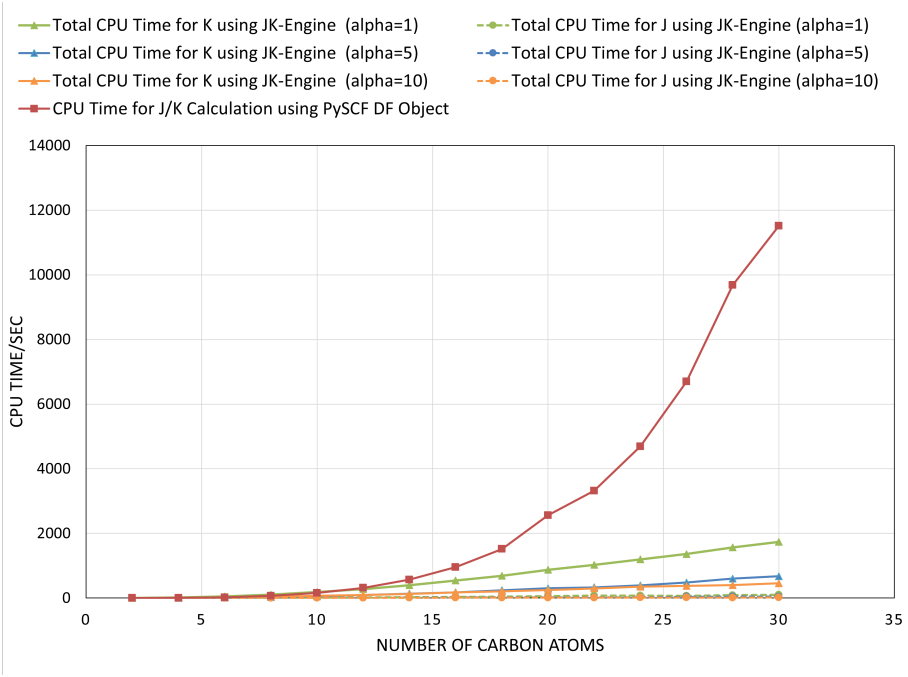


Figure 4.21: Comparative CPU Time for J and K Calculations using PySCF Accurate, PySCF DF Object, and JK-Engine for Alkane Chain C_nH_{2n+2} with cc-pVTZ Basis Set and cc-pVTZ-jkfit Auxiliary Basis Setat α Parameters of 1.0, 5.0, and 10.0.

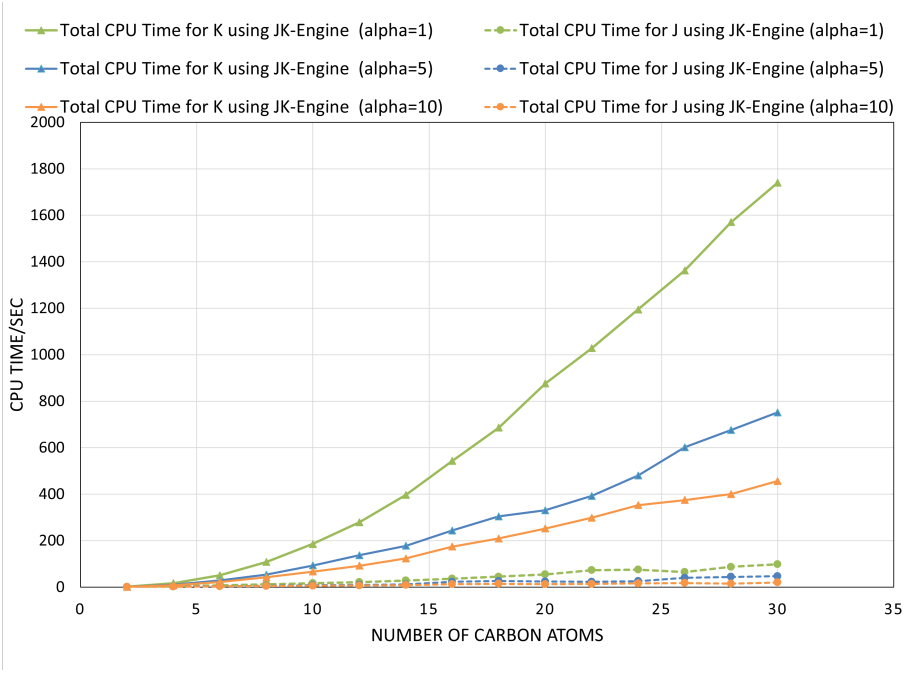


Figure 4.22: Comparative CPU Time for J and K Calculations using PySCF DF Object, and JK-Engine for Alkane Chain C_nH_{2n+2} with cc-pVTZ Basis Set and cc-pVTZ-jkfit Auxiliary Basis Set at α Parameters of 1.0, 5.0, and 10.0.

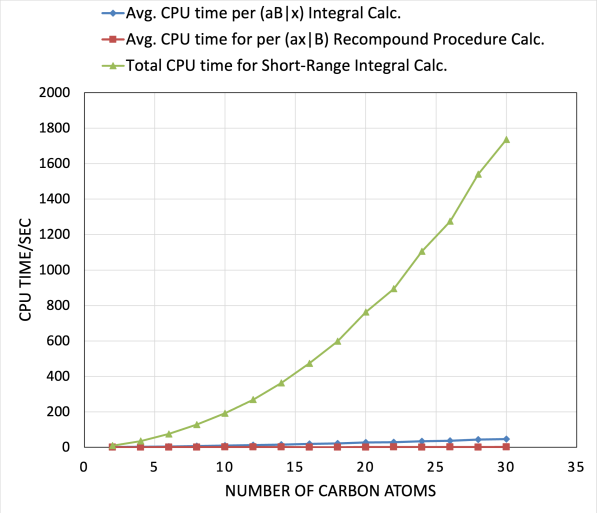


Figure 4.23: Average and Total CPU Time for Short-Range Integral $(\alpha x|B)_{sr}$ Calculations in Alkane Chain C_nH_{2n+2} using the cc-pVTZ Basis Set and cc-pvtz-jkfit Auxiliary Basis Set ($\alpha = 1.0$).

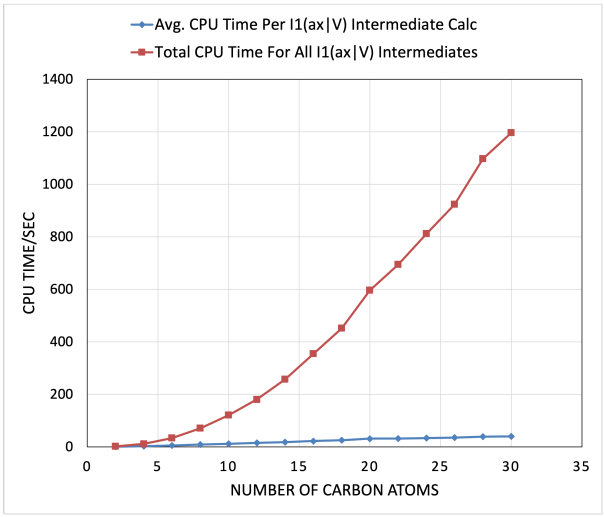


Figure 4.24: Average and Total CPU Time for Intermediate Integral $(\alpha x|V)$ Calculations in Alkane Chain C_nH_{2n+2} using the cc-pVTZ Basis Set and cc-pvtz-jkfit Auxiliary Basis Set ($\alpha = 1.0$).

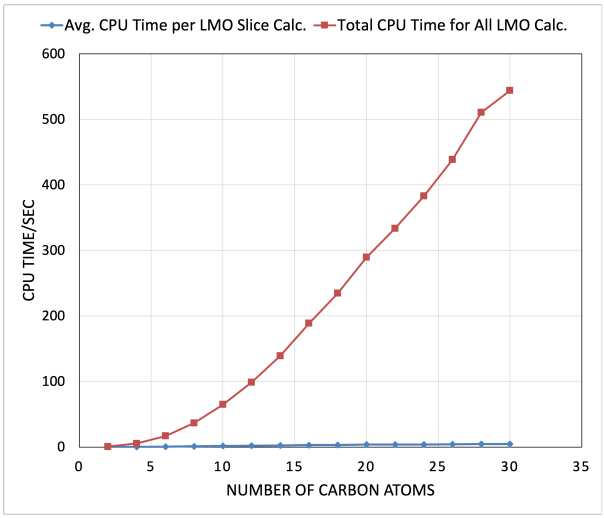


Figure 4.25: Average and Total CPU Time for each LMO Analysis for K Algorithm in Alkane Chain C_nH_{2n+2} using the cc-pVTZ Basis Set and cc-pvtz-jkfit Auxiliary Basis Set ($\alpha = 1.0$).

Cis-transoid Polyacetylene

Similarly, **Figure 4.26 and Figure 4.27** show a comparison of the total CPU time for computing J and K matrices for cis-transoid polyacetylene chains, depicted with green, blue, and orange lines, against the time taken using the PySCF DF method, illustrated in red, across a range of α values. Interestingly, a clear linear relationship was not observed in the intermediate stage from **Figure 4.29**. This deviation can be attributed to the selection procedures in the intermediate stage not being effective, as we loaded all integrals associated with L_{BV} for the cis-transoid polyacetylene chain, which impacted the expected performance. Despite the inherent challenges associated with cis-transoid polyacetylene chains, which generally do not perform as well as alkane chains in computational efficiency, the build-K stage has a good shape in a linear relationship after around ten carbon atoms, shown in **Figure 4.30**. This aspect will be also explored in a later memory analysis. Overall, our JK-Engine of cis-transoid polyacetylene analysis demonstrates a clear advantage over the PySCF DF method.

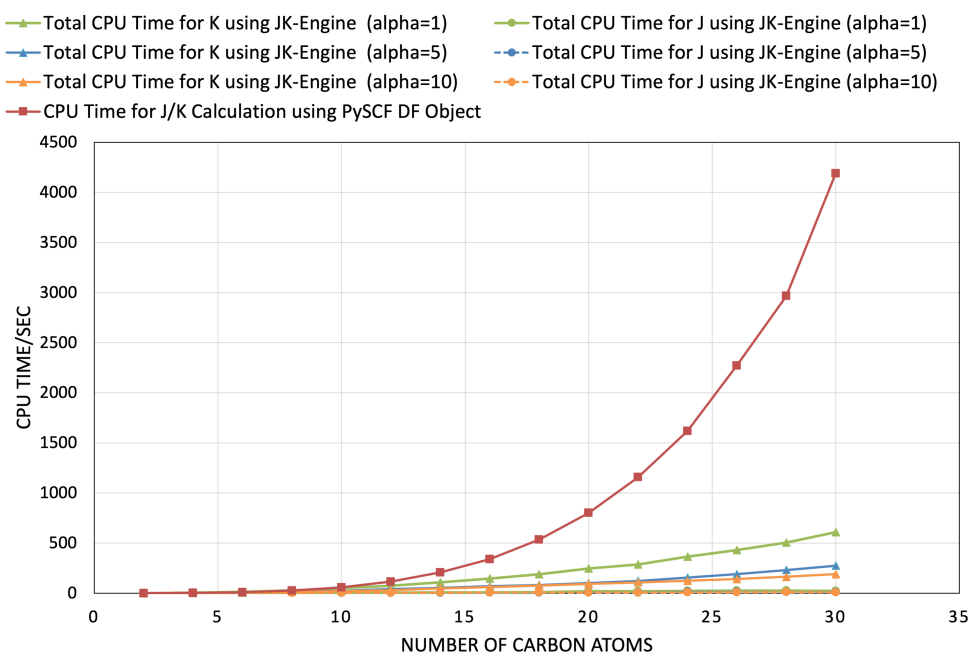


Figure 4.26: Comparative CPU Time for J and K Calculations using PySCF Accurate, PySCF DF Object, and JK-Engine for Cis-transoid Polyacetylene Chain C_nH_{n+2} with cc-pVTZ Basis Set and cc-pVTZ-jkfit Auxiliary Basis Set under α Parameters of 1.0, 5.0, and 10.0.

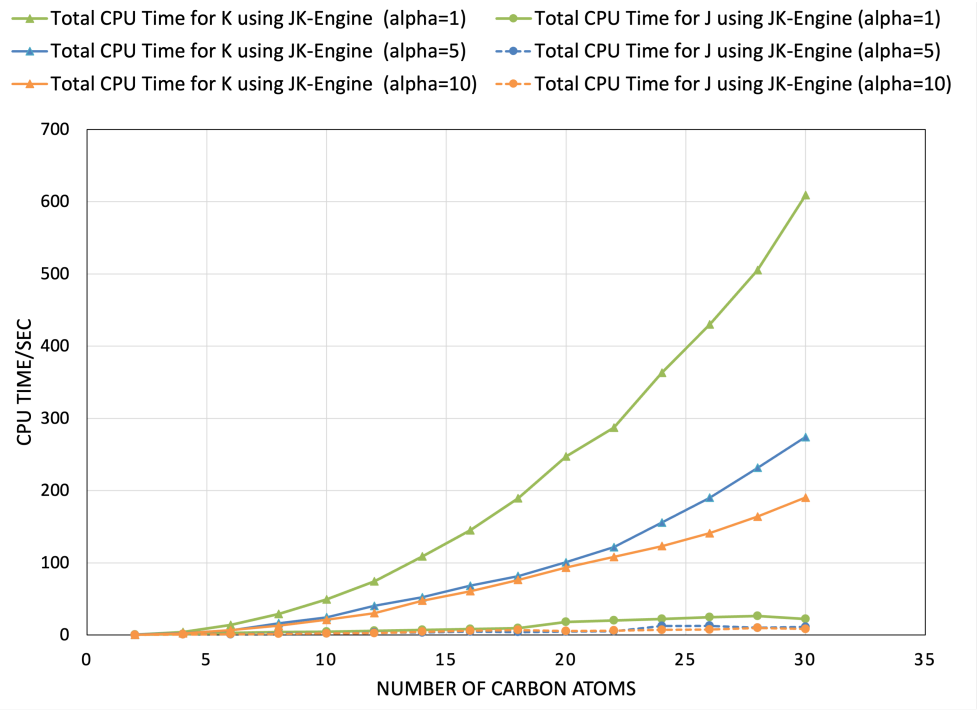


Figure 4.27: Comparative CPU Time for J and K Calculations using PySCF DF Object, and JK-Engine for Cis-transoid Polyacetylene Chain C_nH_{n+2} with cc-pVTZ Basis Set and cc-pVTZ-jkfit Auxiliary Basis Set at α Parameters of 1.0, 5.0, and 10.0.

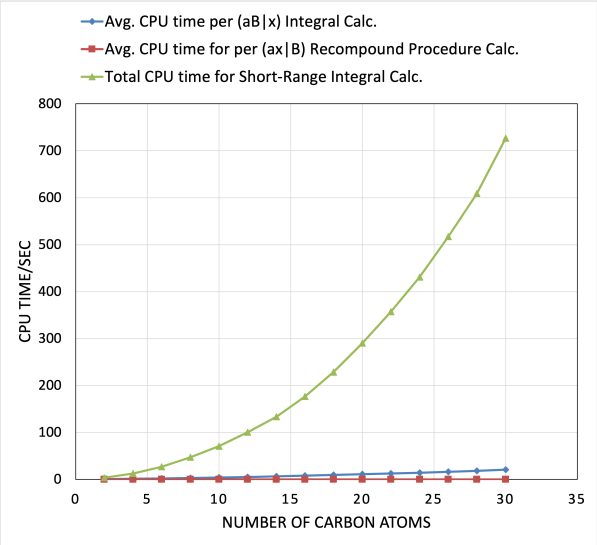


Figure 4.28: Average and Total CPU Time for Short-Range Integral $(\alpha x|B)_{sr}$ Calculations for Cis-transoid Polyacetylene Chain C_nH_{n+2} with cc-pVTZ Basis Set and cc-pVTZ-jkfit Auxiliary Basis Set under α Parameters of 1.0.

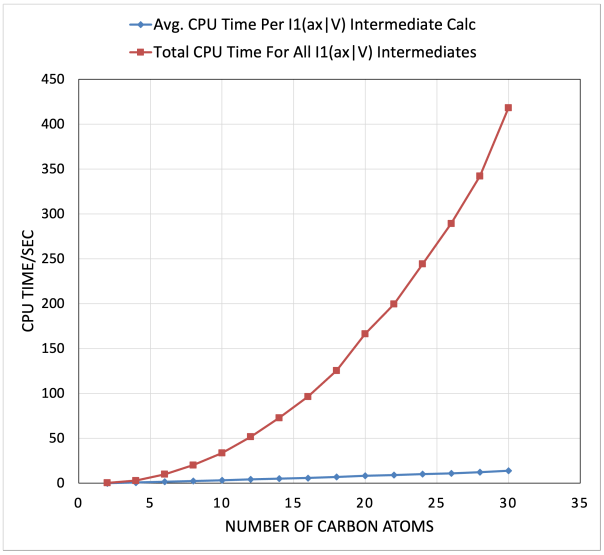


Figure 4.29: Average and Total CPU Time for Intermediate Integral $(\alpha x|V)$ Calculations for Polyacetylene Chain C_nH_{n+2} with cc-pVTZ Basis Set and cc-pVTZ-jkfit Auxiliary Basis Set under α Parameters of 1.0.

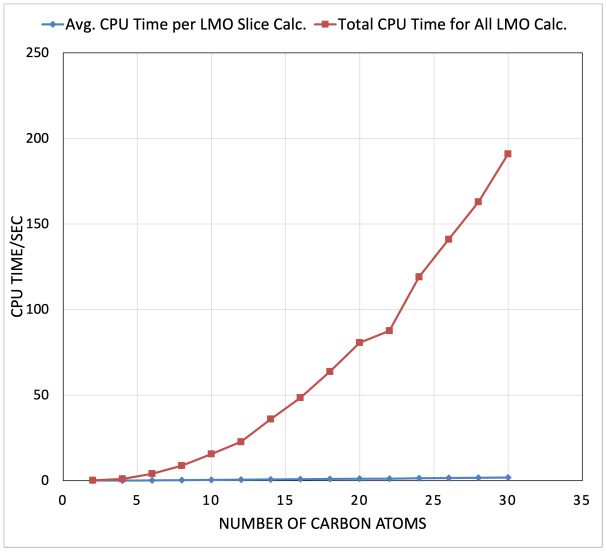


Figure 4.30: Average and Total CPU Time for each LMO Analysis for K Algorithm for Polyacetylene Chain C_nH_{n+2} with cc-pVTZ Basis Set and cc-pVTZ-jkfit Auxiliary Basis Set under α Parameters of 1.0.

4.2.2 Memory Consumption in Exchange Algorithm

In light of the requirements for RAM and hard disk space, the strategies implemented allow for the efficient execution of large-scale calculations without encountering OOM issues. In particular, it's important to note that certain operations generate a substantial volume of intermediate results. Initially, these operations require high RAM usage due to the size of the data being processed. To mitigate this, the approach of processing three-index calculations in slices, particularly for heavy atoms, significantly benefits the management of peak RAM usage, as discussed in Chapter 3. These considerations and optimizations collectively facilitate the implementation of extensive computations, ensuring that the system is well-equipped to handle the demands of such tasks without compromising performance.

Furthermore, the application of several filtering techniques enhances efficiency and reduces RAM demands. These techniques include the elimination of zero entries in every sparse matrix, the use of a dense block in L_{BV} to determine the necessity of loading integrals from the hard disk, and the conversion of the dense metric matrix M_{xy} into a sparse metric matrix $\tilde{M}_{xy}$.

One option to monitor memory consumption involves the use of profiling tools, such as Python's CProfile, to track the memory usage of each function. Given that certain steps in our methodology are designed to decrease data size, peak memory usage alone may not sufficiently reflect the efficiency of these optimizations. To provide a more accurate assessment, we consider additional metrics such as the size of the reduced data and the number of non-zero entries(NNZ). By serving as indicators of data sparsity, these metrics provide us with a clearer understanding of the impact that our data structures and processing steps have on memory consumption. In this context, we have conducted analyses on three molecular systems:

Polywater Systems

Figure 4.31 shows the number of iterations at the intermediate stage and exchange slice stage in our exchange algorithm. Specifically, the blue line represents the total number of slices in localized molecular orbitals (LMOs), which also refers to the count of $(\alpha x|B)$ slices before selecting, the green line indicates the total number of LMOs, and the red line denotes the count of loaded $(\alpha x|B)_{sr}$ integrals. The maximum absolute element in each slice in LMOs (L_{BV}) depends on whether the algorithm loads the corresponding short-range integral slice $(\alpha x|B)_{sr}$ or not. If there is a zero block in L_{BV}, the algorithm is designed to automatically omit the calculations involving all zeros. The presented data in red line describes that with increasing complexity of the molecular system, the frequency of loaded short-range integrals maintains a linear correlation. The choice of a polywater model as the test example is optimal due to the considerable separation between two water monomers, resulting in a matrix for $L_{\beta\mu}$ that exhibits an exemplary block-sparse structure. Hence, our algorithm selectively concentrates on processing the dense blocks.

Following the computation of intermediate integrals $I(\alpha x; V)$, attention shifts towards the subsequent calculation about each LMO. Moreover, **Figure 4.31** also describes a

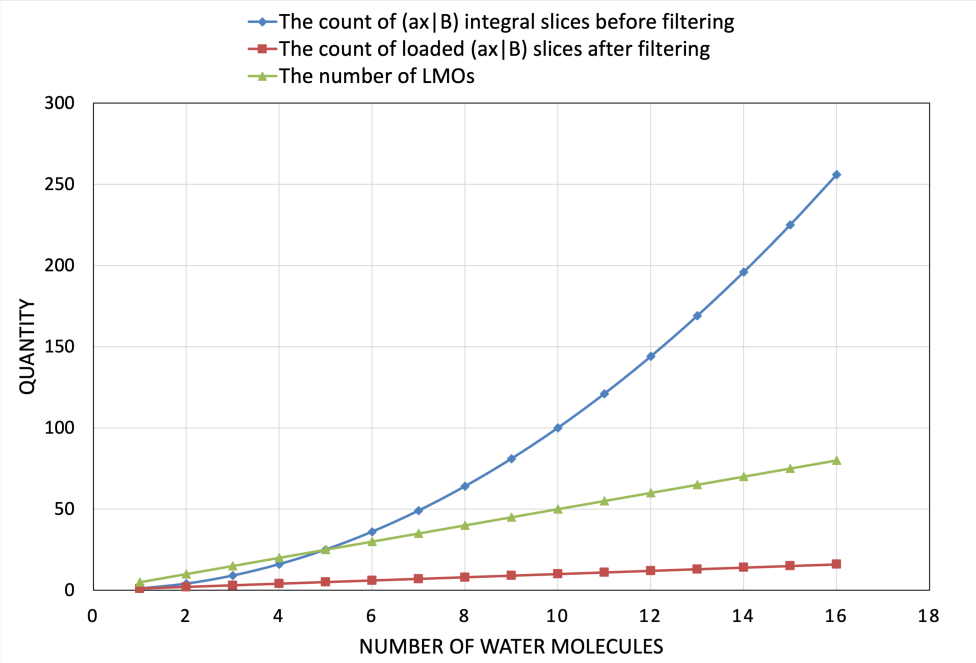


Figure 4.31: Correlation between the number of water molecules and quantity of slices. The graph illustrates the linear increase in both the number of loaded short-range $(\alpha x|B)$ slices and the quantity of LMOs, as well as the quadratic increase in the total number of slices within L_{BV}, as the number of water molecule count increases from one to ten (noted as $(H_2O)_1$ to $(H_2O)_{16}$) using the cc-pVTZ Basis Set and cc-pvtz-jkfit Auxiliary Basis Set $(\alpha = 1.0)$.

linear correlation regarding the number of LMOs, as shown by the green dot line, with the augmentation of the number of water monomers. This indicates a consistent linear scaling behavior of both dense-block processing and LMO calculations with system size, underscoring the efficiency and scalability of our algorithm in handling extensive molecular systems.

The exploration of space complexity in our short-range integral preparation algorithm is essential. We initially faced a scenario where the traditional storage requirements for the three-centered short-range integrals, denoted as $(\alpha\beta|x)$, scale with $O(n_{ao}^2 \cdot n_{naux})$. To avoid memory leaks, we adopted a cutting procedure along with the sparse representation. This approach successfully reduced the space complexity to $O(1)$ of each slice, involving integral slice $(\alpha B|x)$, intermediate slice $I_1(\alpha x|V)$, transformed metric matrix $\tilde{M}_{xy}$, and intermediate contribution $I_2(x\alpha) = \tilde{M}_{xy} \cdot I(\alpha x)^T$, making it independent of the size of the input molecule, illustrated in **Figure 4.32**. The central strategy of this cutting procedure involves calculating the short-range three-centred integrals for a system of atoms and subsequently mapping these subsystem integrals to whole molecule integrals. Crucially, all integrals are stored in a sparse format, ensuring that only non-zero elements in the short-range integrals are retained.

Let us begin with polywater models to explain. We categorized atom groups within these systems by identifying each group as having a single heavy atom from **Algorithm**

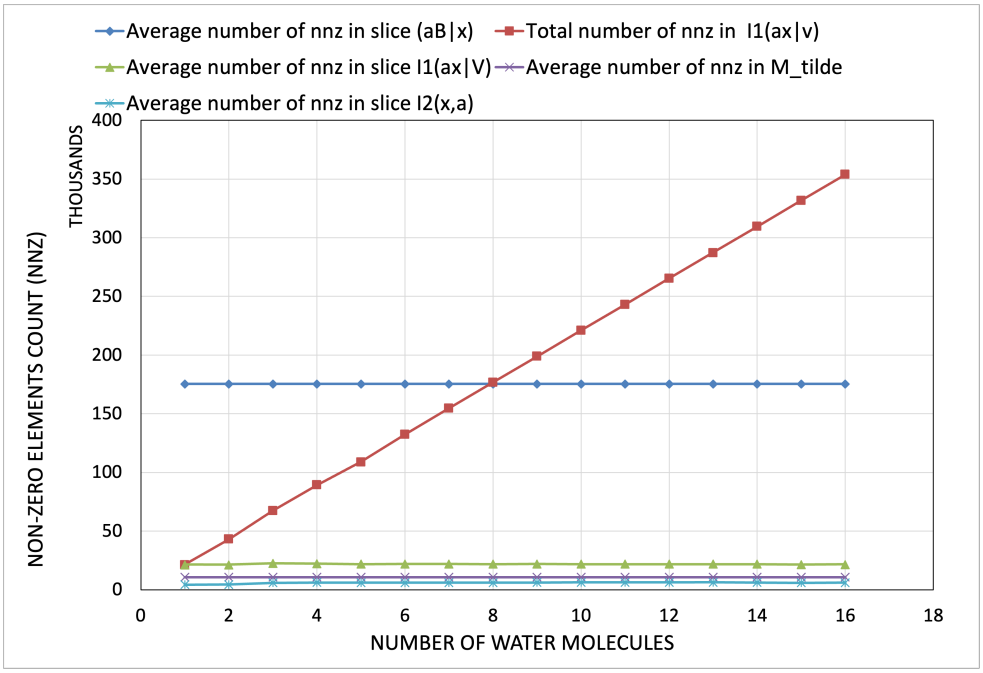


Figure 4.32: Correlation between the number of water molecules and non-zero elements of intermediate contribution slices. The graph illustrates the linear increase in total number of NNZ in intermediate $I_1(\alpha x|\mu)$. In contrast, the average number of NNZ in other contribution slices remains relatively constant, as the number of water molecule count increases from one to ten (noted as $(H_2O)_1$ to $(H_2O)_{16}$) using the cc-pVTZ Basis Set and cc-pvtz-jkfit Auxiliary Basis Set ($\alpha = 1.0$).

A.1.2. After this procedure, we determined that the number of atoms in each group, denoted by $\mathbf{n_{gatm}}$, is 3. This implies that we calculate the integrals for one water molecule at a time:

$$
\begin{aligned}
W1(\alpha\beta|x)_{sr} &= (\alpha B_1|x) \\
W2(\alpha\beta|x)_{sr} &= (\alpha B_1|x) + (\alpha B_2|x) \\
&= \sum_{A,X}(AB_1|X) + \sum_{A,X}(AB_2|X) \\
&= (A_1B_1|X_1) + (A_2B_2|X_2) \\
W3(\alpha\beta|x)_{sr} &= (\alpha B_1|x) + (\alpha B_2|x) + (\alpha B_3|x) \\
&= \sum_{A,X}(AB_1|X) + \sum_{A,X}(AB_2|X) + \sum_{A,X}(AB_3|X) \\
&= (A_1B_1|X_1) + (A_2B_2|X_2) + (A_3B_3|X_3) \\
&\cdots\cdots \\
WN(\alpha\beta|x)_{sr} &= (\alpha B_1|x) + (\alpha B_2|x) + \cdots + (\alpha B_N|x) \\
&= \sum_{A,X}(AB_1|X) + \sum_{A,X}(AB_2|X) + \cdots + \sum_{A,X}(AB_N|X) \\
&= (A_1B_1|X_1) + (A_2B_2|X_2) + \cdots + (A_NB_N|X_N)
\end{aligned}
\tag{4.5}
$$

As **Equation 4.5**, the preparation procedures store the non-zero data in three-index dense tensor $(A_N B_N|X_N)$ which contributes the whole molecule integrals with their row and column indices into the hard disk. According to **Figure 4.32**, although the three-index short-range integrals $(\alpha\beta|x)$ grow cubically as the molecular system increases, the data reveals a constant pattern in the total number of non-zero elements for the dense slices of intermediate integrals $I_1(\alpha x|\mu)$ (depicted by the red line), meanwhile the average number of non-zero elements in each slice $(\alpha B|x)$ and slice $I_1(\alpha x|V)$ remains relatively unchanged (illustrated by the dark blue and green lines).

Besides, the dense metric matrix M_{xy} also represents a large portion of memory usage, scaling with $O(n_{aux}^2)$. The matrix multiplication of sparse matrix and dense matrix will get the dense intermediate matrix $I_2(x\alpha) = M_{xy} \cdot I(\alpha x)^T$, resulting in expensive computational cost. To maintain linear memory usage throughout molecule calculations, **Algorithm A.2.3** is implemented to conduct transformed metric matrix $\tilde{M}_{xy}$ within the exchange algorithm, focusing exclusively on the non-zero matrix elements in the molecule. As illustrated in **Figure 4.32**, there is a consistent relationship between the NNZ elements in the transformed metric matrix $\tilde{M}_{xy}$ and the number of water molecules, represented by the purple line. Similarly, this relationship extends to the intermediate slice $I_2(x\alpha)$, which is represented by the shallow blue line.

The polywater simulations serve as ideal models, yielding perfectly block-sparse formats for contributions to the algorithm. Now, let's shift our focus to more realistic molecular systems, specifically alkane and cis-transoid polyacetylene chains.

Alkane Chains

As shown in **Figure 4.33**, the screening procedure of dense L_{BV} blocks becomes effective starting from 18 carbon atoms in the red line. This effectiveness is attributed to the locality of the localized molecular orbitals, in contrast to the unfiltered count shown by the blue line. Additionally, the significant reduction in the number of loaded three-index integrals markedly enhances the efficiency of our exchange algorithm, as it eliminates the need to compute the unselected sparse blocks. This substantial decrease in the number of loaded integral slices is expected to greatly improve the efficiency of our JK-Engine's exchange algorithm for larger molecular systems.

Furthermore, **Figure 4.34** illustrates the number of non-zero entries in every slice of integrals in every step. At first, the average number of NNZ increases as the number of carbon atoms rises from 1 to 10. Beyond this point, the average number of non-zero elements plateaus, maintaining a consistent level, which suggests that a well-defined block-sparse matrix format emerges after the tenth carbon atom. This saturation indicates that the efficiency of the matrix operations for each slice, especially for K-build, will not diminish with further increases in the size of the alkane molecule, as the sparsity pattern remains stable. This explains the linear correlation observed after 10 carbon atoms in the K-build stage, as depicted in Figure 4.25.

These different onsets of linear behaviour reveal an interesting feature. We require LMO coefficients to drop below a small threshold before we can discard computing contributions to the intermediate $I(\alpha, x|\nu)$, and apparently efficiencies only arise at a quite large onset of 18 carbon atoms for simple alkanes. If we analyse the resulting NNZ elements we see a much earlier onset of perhaps even as small as 6 carbon atoms. It is not entirely clear if one can capitalize further on this. The value of the LMO threshold may be increased, or a more sophisticated screening is needed.

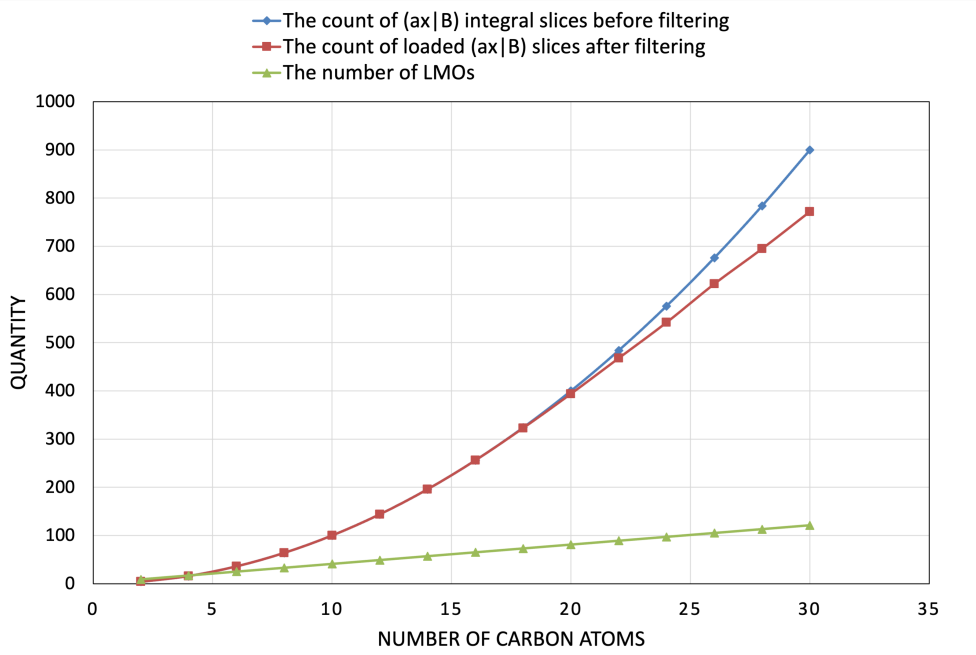


Figure 4.33: Correlation between the number of carbon atoms of alkane chain and quantity of slices. The graph illustrates the linear increase in both the number of loaded short-range $(\alpha x|B)$ slices and the quantity of LMOs, as well as the quadratic increase in the total number of slices within L_{BV}, as the number of carbon atoms count increases from two to thirty for alkane system C_nH_{2n+2} using the cc-pVTZ Basis Set and cc-pvtz-jkfit Auxiliary Basis Set ($\alpha = 1.0$).

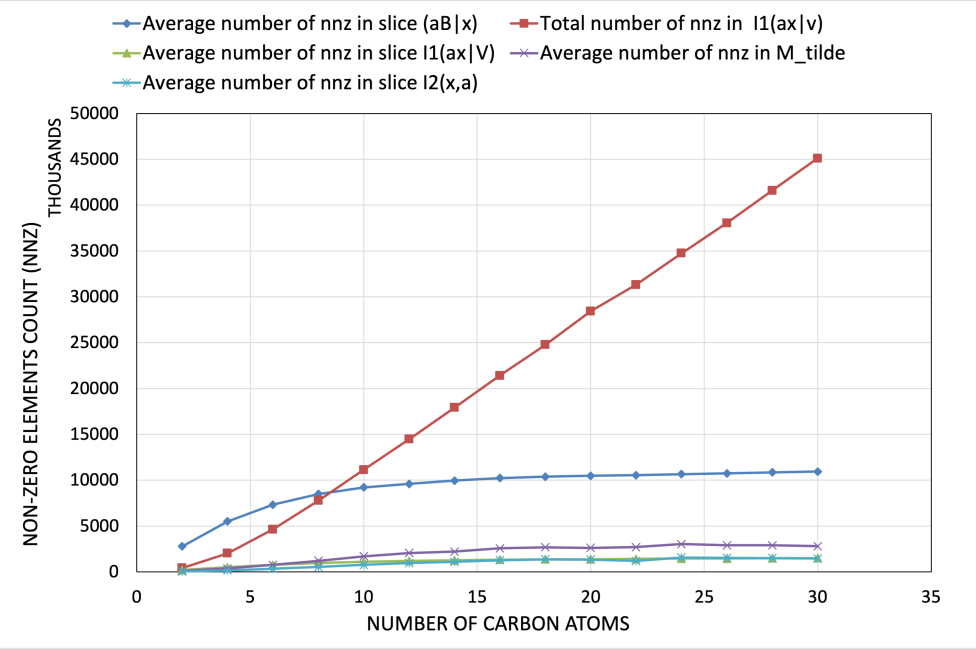


Figure 4.34: Correlation between the number of carbon atoms of alkane chain and non-zero elements of intermediate contribution slices. The graph illustrates the linear increase in total number of NNZ in intermediate $I_1(\alpha x|\mu)$. In contrast, the average number of NNZ in other contribution slices remains relatively constant, as the number of carbon atoms counts increases from two to thirty for alkane system C_nH_{2n+2} using the cc-pVTZ Basis Set and cc-pvtz-jkfit Auxiliary Basis Set ($\alpha = 1.0$).

Cis-transoid Polyacetylene Chains

Unlike the alkane case, from the provided data, the cis-transoid polyacetylene chain does not benefit from a reduction in the number of loaded integral slices as the number of carbon atoms increases in **Figure 4.35**. This is because polyacetylene systems exhibit electron-distribution across both σ and π bonds. Sigma (σ) bonds establish the structural framework, enabling atomic connections and ensuring molecular stability, while the pi (π) bonds, formed by sideways overlap of p orbitals, introduce extra electron density above and below the molecular plane. This configuration not only makes polyacetylenes more chemically reactive compared to alkanes but also allows for the delocalization of π electrons in conjugated systems. The requirement for loaded integral slices is influenced by the localized molecular orbitals, and in the case of polyacetylenes, the presence of dense L_{BV} blocks necessitates the calculation of all associated three-index integrals, reflecting the high-demand calculations within these molecules. This directly explains the non-linear relationship observed in CPU timing for polyacetylenes chain systems, as seen in computational analyses in **Figure 4.29**.

Moreover, as depicted in **Figure 4.36**, the average count of non-zero elements within the contributions at every step stabilizes after reaching 6-8 carbon atoms. This consistent sparsity within each slice, regardless of the size of the molecular system, ensures that

the computational expense associated with each slice remains constant. Consequently, we achieved a linear-scaling property in molecular systems after 8 carbons for the K-build stage, shown in **Figure 4.30**. Interestingly this behaviour is more or less consistent between both alkane and alkene chains. It appears that the tails of the orbitals drop off more closely for cis-transoid polyacetylene chains, but the generic locality of the orbitals is not so different and this is reflected in the behaviour of NNZ in the $I(\alpha x|\nu)$ intermediates.

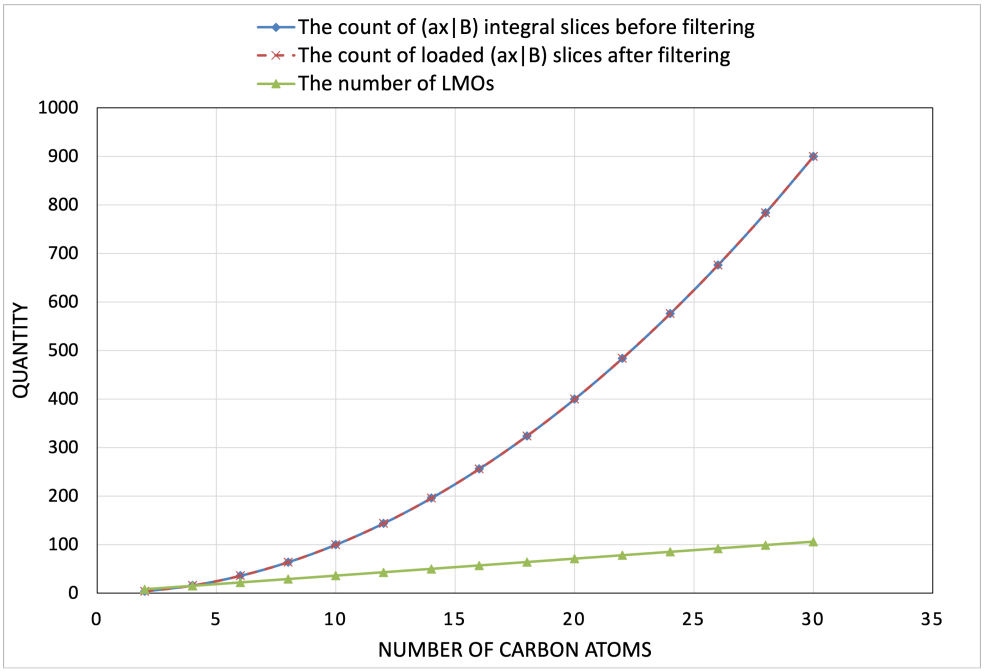


Figure 4.35: Correlation between the number of carbon atoms of alkene chain and quantity of slices. The graph illustrates the linear increase in both the number of loaded short-range $(\alpha x|B)$ slices and the quantity of LMOs, as well as the quadratic increase in the total number of slices within L_{BV}, as the number of carbon atoms count increases from two to thirty for cis-transoid polyacetylene system C_nH_{n+2} using the cc-pVTZ Basis Set and cc-pvtz-jkfit Auxiliary Basis Set ($\alpha = 1.0$).

In this chapter, we've undertaken a comprehensive evaluation of our JK-Engine algorithm's computational performance, paying close attention to accuracy, CPU processing time, and memory consumption. Emphasizing its versatility, we've ensured that the algorithm is robust against memory leaks, making it suitable for molecular systems of varying sizes. Our assessment spans polywater models, alkane chains, and cis-transoid polyacetylene chains, demonstrating the algorithm's scalability with molecular size.

For our polywater benchmarks, we observed a stable and efficient computation as the number of water molecules increased, thanks to a well-structured block-sparse matrix format, particularly for extended systems with minimal molecular interaction.

From the exploration of the timings and RAM requirements for the alkane and polyacetylene chains, we can reach a noteworthy conclusion. The computation of the intermediate $I1(\alpha x|V)$ demonstrates that the linear-scaling property may only be feasible at a

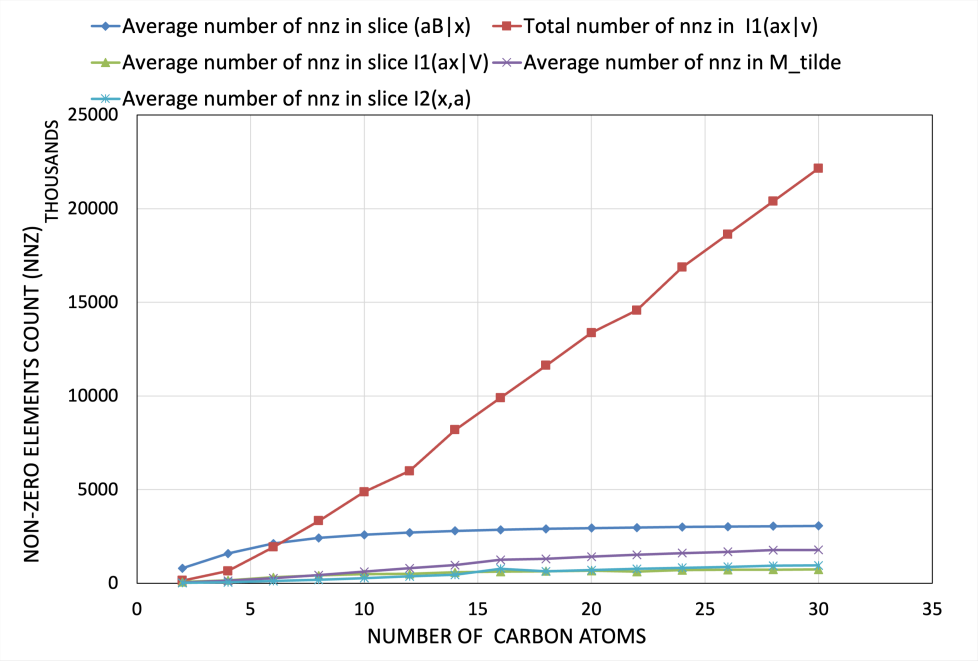


Figure 4.36: Correlation between the number of carbon atoms of alkene chain and non-zero elements of intermediate contribution slices. The graph illustrates the linear increase in total number of NNZ in intermediate $I_1(\alpha x|\mu)$. In contrast, the average number of NNZ in other contribution slices remains relatively constant, as the number of carbon atoms count increases from two to thirty for cis-transoid polyacetylene system C_nH_{n+2} using the cc-pVTZ Basis Set and cc-pvtz-jkfit Auxiliary Basis Set ($\alpha = 1.0$).

fairly late stage. specifically, the linear correlation starts at 18 carbon atoms for alkanes, with polyacetylenes yet to show this behavior. The reason for this behaviour depends on the threshold $ThreshS$ we use to identify non-zero L_{BV} blocks. Despite this, the sparse nature of the data in $I1(\alpha x|V)$ allows for linear scaling to be observed as early as 6 carbon atoms for both alkane and polyacetylene chains, shown in **Figure 4.34** and **Figure 4.36**. This is an excellent onset for linear scaling, and as a consequence also the timings in the Build-K part of the calculation scale in a linear fashion with this early onset. This is illustrated in **Figure 4.25** and **Figure 4.30**.

Overall our JK-engine algorithm shows significant advantages in computational speed over the PySCF DF method, suggesting its effectiveness across different molecular frameworks. Further improvements may be possible, given the interesting features we identified above.

Chapter 5

Construction of Orbital Domain

The cluster-in-molecule (CiM) approach is developed for calculating localized electron correlation energy using a foundation of orthogonal occupied and virtual localized molecular orbitals (LMOs).

$$\begin{aligned} E_{corr} &= \sum_{i,j,a,b} (ia|jb)\tau_{ab}^{ij} \\ &= \sum_{I} \sum_{i \in I} \sum_{j,a,b} (ia|jb)\tau_{ab}^{ij} \\ &\equiv \sum_{I} \sum_{i \in I} E_i \end{aligned} \tag{5.1}$$

where the notations of i, j represent localized occupied orbitals, and the notations of a, b represent localized virtual orbitals. The quantity $(ia|jb)$ is two-electron integral in terms of spatial occupied (i,j) and virtual (a,b) orbitals.

Crucially, the core idea of the CiM method is to break down the correlation energy calculations of large molecular system into smaller subsystems with specific LMOs. Hence, the CiM method performs linear scalability in computation time through the transformed two-electron repulsion integrals (ERIs) over LMOs with efficient truncation methods. This approach emphasizes the crucial step of selecting virtual and occupied orbitals within each subset of the system for post-Hartree-Fock(HF) calculations. In Nooijen group, our CiM approach incorporates a scalable and size-independent methodology for the controllable construction of orbital domain (COD) from transformed exchange matrices under domain central I, ensuring efficiency across various molecular sizes.

The key idea underlying the construction of orbital domains can be rationalized as follows. The largest contributions to the correlation energy would come from diagonal $i = j$ contributions. These integrals are the same integrals that enter the HF exchange matrix, $K_{ab} = \sum_i (ia|ib)$. If one performs a diagonalization or a Cholesky decomposition of the matrix K_{ab}, corresponding to the occupied orbitals i in the domain I one performs an orbital selection of important virtual orbitals in the domain. The use of the exchange matrix is the key idea underlying the orbital selection scheme. We can use a similar strategy to select important occupied orbitals, constructing the exchange matrix over occupied

orbitals: $K_{kl} = \sum_{i \in I}(ki|li)$. This approach has been used successfully in the Laplace MP2 approach by Demel and Nooijen to screen important occupied (ij) pairs in exchange $MP2_x$ contributions [44, 7]. While this is the essential idea, additional details need to be worked out in the non-orthogonal AO basis. We will discuss the algorithm for orbital selection and provide some elementary examples. The next step would be to use these orbital domains to perform CCSD cluster in molecule calculations. This is beyond the scope of the thesis unfortunately.

This chapter will explore in detail the methodology for constructing orbital domains for both occupied and virtual spaces centered around the specific domain central I. This orbital selection algorithm is applied in parallel across different central LMOs, effectively identifying the relevant occupied and virtual orbitals in their vicinity.

The most important aspect of the orbital domain construction shown in this chapter is that the expensive step in the calculation is always the construction of the exchange matrix corresponding to some input density matrix, depending on the step of the calculation. The main contribution in this thesis is an efficient algorithm to construct the exchange matrix. In this algorithm we prepare a number of short-range integrals, sorted exactly to optimize the efficiency of the exchange matrix calculation. This algorithm is deployed many times, once for each central orbital, while reusing these integrals. Likewise the 3-center integrals can be used in the next step in a CiM calculation that is a major bottleneck in current implementations: the transformation of integrals.

Below the algorithm to construct orbital domains is described in detail. The development of this algorithm was part of a senior Chem494 research project, and the material is obtained from this source.

5.1 AO-based Construction of Orbital Domain

A fundamental approach to alleviating computational demands in large molecular systems involves the construction of orbital domains COD. Specifically, the COD algorithm eliminates irrelevant Localized Molecular Orbitals (LMOs) and retains the essential LMOs for each domain central I. The reduction process in the orbital selection scheme mainly relies on pivoted Cholesky Decomposition and Löwdin orthonormalization. Additionally, AO-based construction of domain orbitals provides a general solution, since it transfers any orbitals from AO-based space to an orthonormal basis set. Another advantage is that the number of selected orbitals scales in zero-order with respect to the size of the molecular subsystem.

Notation	**Description in the pseudo algorithm**
$\alpha, \beta, \gamma, \delta$	atomic orbital functions
a, b, c	virtual atomic orbitals or unoccupied orbitals
i, j, k	occupied orbitals or holes
p, q, r, s	generic orthonormal orbitals
z	transformed exchange basis or Cholesky basis

μ, ν	localized molecular occupied orbitals

Table 5.1: List of symbols used in COD calculations

Algorithm A.4.2 outlines detailed steps, with the symbols used detailed in **Table 5.1**. The inputs of COD contain localized occupied molecular orbitals, representing the domain central, alongside overlap matrix $S_{\alpha\beta}$, projector matrix $P_{\alpha\beta}$, and adjustable thresholds. This algorithm was a key component of my 494 project, where a notable enhancement involved the introduction of adjustable thresholds in the pivoted Cholesky Decomposition process. The selection of orbitals is finely tuned through these thresholds, allowing flexible control over their number.

Algorithm 14 Construct Orbital Domain

$L \leftarrow S_{\alpha\beta} = L_{\alpha\mu} L_{\alpha\mu}^T$ ▷ Cholesky decomposition of $S_{\alpha\beta}$
Procedure CDO($C_{\alpha\mu}$, $P_{\alpha\beta}$, $L_{\alpha\mu}$, η, $thresh$)
$D_{new} \leftarrow C_{\alpha\mu} C_{\alpha\mu}^T$
$K \leftarrow getExchangeSlice(int3cFolder, Q_{xy}, D_{new}, threshZ)$
$\tilde{K}_{zz} \leftarrow (L^T P) K (PL)$
$\tilde{Y}_{zk} \leftarrow \tilde{K}_{zz} = \tilde{Y}_{zk} \tilde{Y}_{zk}^T$ ▷ Cholesky decomposition of $\tilde{K}_{zz}$
$max\_per\_column \leftarrow \max(\tilde{Y},$ along axis $= 0)$
if $max\_per\_column > \eta$ **then**
    Identify corresponding eigenvector in $\tilde{Y}_{zk}$
    Keep this eigenvector $\tilde{Y}_{zk}[:, max\_per\_column]$
end if
$\tilde{X}_{\alpha k} \leftarrow \tilde{X}_{\alpha k} = \sum_z (L^T)_{\alpha z}^{-1} \tilde{Y}_{zk}$ ▷ Transform to AO basis
$M \leftarrow M = \tilde{X}^T S \tilde{X}$
$X \leftarrow X = \tilde{X} M^{-1/2}$ ▷ Löwdin Orthonormalization
return X, n ▷ X: selected orbitals, n: number of selected orbitals
EndProcedure

Figure 5.1: Pseudo Algorithm of Construction of Orbital Domain Subroutine (same with **Algorithm A.4.2**).

The algorithm begins by constructing a new density matrix, $D_{new} = C_{\alpha\mu} C_{\alpha\mu}^T$, using the coefficient matrix of one domain central I, one of the localized occupied orbitals. This step is essential for identifying the relevant space of the electron density distribution. It then proceeds with the pivoted Cholesky decomposition of the overlap matrix $S_{\alpha\beta}$, resulting in a lower triangular matrix $L_{\alpha\mu}$. This decomposition is important for transforming the exchange integral calculations to a more manageable form.

The core computational step involves calculating the transformed exchange integrals, $\tilde{K}$, within the specified projector space. Each subsystem of LMO constructs its own subset of occupied space and virtual space associated with efficient exchange integrals, $K_{\alpha\beta}$. Next,

Project AO-based $K_{\alpha\beta}$ to the orthonormal orbital space of interest, $\tilde{K}_{zz}$. This transformation is crucial for localizing the computation within the relevant orbital space.

$$K_{\alpha\beta} = \sum_{i \in P} (\alpha i | \beta i) \tag{5.2}$$

$$\tilde{K}_{zz} = (L^T_{\alpha\mu} P_{\alpha\beta}) K (P_{\alpha\beta} L_{\alpha\mu}) \tag{5.3}$$

A selective pivoted Cholesky decomposition of the transformed exchange matrix, $\tilde{K}_{zz}$, is performed to identify significant Cholesky vectors, $\tilde{Y}_{zk}$, i.e., those eigenvalues with contributions above an eigenvector significance threshold η. The goal of this threshold η is to screen out meaningless eigenvectors within a Cholesky matrix based on their contribution, and this step effectively reduces the computational complexity by focusing on significant interactions.

$$\tilde{K}_{zz} = \tilde{Y}_{zk} \tilde{Y}^T_{zk} \tag{5.4}$$

The selected Cholesky vectors are then transformed back to the AO basis using the inverse of the Cholesky decomposed overlap matrix. This transformation is necessary for aligning the selected orbitals with the original molecular basis. Lastly, Löwdin orthonormalization is applied to these transformed vectors to ensure they form an orthonormal basis set, essential for further quantum chemistry calculations.

$$\tilde{X}_{\alpha k} = \sum_{z} (L^T)^{-1}_{\alpha z} \tilde{Y}_{zk} \tag{5.5}$$

The algorithm outputs a set of selected domain orbitals (X) and the number of these orbitals (n).

Controllable Thresholds in COD	**Default Value**
Eigenvector significance threshold(η)	1×10^{-5}
pivoted Cholesky of density matrix ($threshZ$)	1×10^{-5}
pivoted Cholesky of overlap matrix	1×10^{-8}
Primary virtual space (V_1)	1×10^{-3}
Secondary occupied space (O_2)	1×10^{-3}
Secondary virtual space (V_2)	1×10^{-3}

Table 5.2: List of all thresholds in COD Algorithm. The controllable thresholds decide how many orbitals we select.

5.2 Overall Construction of Orbital Domain

The **Algorithm A.4.1** illustrates a systematic approach for constructing an orbital domain by utilizing AO representations to get overall virtual and occupied space. This method focuses on partitioning the orbital space into whole occupied and whole virtual spaces for each subspace centered around I, corresponding to a set of localized molecular orbitals (LMOs).

This algorithm starts with defining a projector matrix $P_{\alpha\beta}$, noted as $\bar{D}$ for primary virtual space V_1, defined as the difference between the inverse of the overlap matrix $S^{-1}_{\alpha\beta}$ and the density matrix $D_{\alpha\beta}$. This matrix will identify which orbitals should be considered in the construction of a specific domain, effectively dividing the space into occupied and virtual domains.

$$P_{\alpha\beta} = \bar{D}_{\alpha\beta} = S^{-1}_{\alpha\beta} - D_{\alpha\beta} \tag{5.6}$$

For each central I within a cluster, which means an iterative loop over all LMOs, each LMO acts as a focal point for constructing subsets of virtual and occupied spaces. The primary subset of virtual space, V_1, is identified through the previous COD subroutine, which utilizes the projector $\bar{D}_{\alpha\beta}$ aimed at identifying virtual orbitals, $X_{\alpha a}$, that have a close connection with the respective LMO.

Following the orbital selection of V_1, the projector matrix is updated to reflect the subtraction of these orbitals, hence focusing the algorithm on the remaining virtual space, V_2, for further selection. Apply projectors on remaining virtual space ($\bar{\bar{D}}$):

$$\bar{\bar{D}} = \bar{D} - \sum_{i \in I} X_{\alpha a} X_{\beta a} \tag{5.7}$$

Similarly, the projector on occupied space O_1, denoted as ($\tilde{D}$) to the AO-based COD algorithm, excluding the central I orbital.

$$\tilde{D} = D - \sum_{i \in I} C_{\alpha i} C_{\beta i} \tag{5.8}$$

To obtain the complete virtual space, a secondary selection process on virtual space, V_2, aims to capture virtual orbitals not identified in the primary round. This is achieved through employing subroutine COD a projector on the residual virtual space, $\bar{\bar{D}}$.

At last, we combine both the primary virtual orbital V_1 and the secondary virtual orbitals V_2 together as an entire virtual space V. Likewise, the entire occupied space encompasses the central I orbital $L_{\alpha i}$ and occupied space O_1.

The **Table 5.3** provides an overview of the characteristics of occupied and virtual orbitals for a $C_{12}H_{26}$ molecule analyzed within a 3-21G basis set, contextualized within the framework of the AO-based Construct Orbital Domain algorithm previously described. The table lists orbitals centered around the first localized molecular orbital (center I), showing each orbital's average position (R_μ) and radial extent (σ_μ).

Algorithm 13 AO-based Construct Orbital Domain

$P_{\alpha\beta} \leftarrow S^{-1}_{\alpha\beta} - D_{\alpha\beta}$
for $i \leftarrow 0$ **to** n_v **do** ▷ n_v: number of LMO, $L_{\alpha\mu}$
  $V_1, n_a \leftarrow \mathrm{CDO}(L_{\alpha i}, P_{\alpha\beta}, S_{\alpha\beta}, thresh)$ ▷ V_1: Virtual1, n_a: number of virturals
  $\bar{\bar{D}} = P_{\alpha\beta} - V_1 V_1^T$
  $O_1, n_j \leftarrow \mathrm{CDO}(L_{\alpha i}, \tilde{D}, S_{\alpha\beta}, thresh)$ ▷ O_1: Occupied1, n_j: number of occupieds
  $\tilde{D} = D_{\alpha\beta} - L_{\alpha i} L^T_{\alpha i}$
  $V_2, n_b \leftarrow \mathrm{CDO}(L_{\alpha i}, \bar{\bar{D}}, S_{\alpha\beta}, thresh)$ ▷ V_2: Virtual2, n_b: number of virtuals
  $O = L_{\alpha i} + O_1$
  $V = V_1 + V_2$
end for
return O, V ▷ O: occupied space, V: virtual space

Figure 5.2: Pseudo Algorithm of Overall Construction of Domain Orbital, including entire virtual and occupied space for each center I (same with **Algorithm A.4.1**).

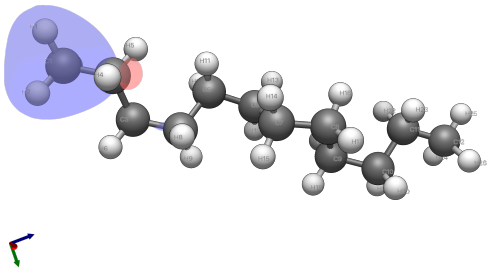
The center I orbital has an average position of 15.119 with a radial extent of 0.366, serving as the reference value for the first central I. The occupied orbitals, numbered from 01 to 05, display average positions ranging from 10.526 to 11.349, with radial extents varying from 0.500 to 2.324. These occupied orbitals represent the local electron density regions closely associated with the first center I. Similarly, the virtual orbitals, identified as 01 to 09, show average positions in a similar range to the occupied orbitals, but with generally broader radial extents, reaching up to 3.337. This indicates that virtual orbitals span larger spaces around the center I with a wider range of electron density fluctuations. These virtual and occupied orbitals significantly contribute to electron correlation for small pieces of fragments.

Alternatively, **Figure 5.3** displays molecular orbital surfaces for a $C_{12}H_{26}$ molecule in a 3-21G basis set, focusing on specific center $I23$ and center $I25$. It provides visualizations for two different central orbitals with examples of their respective occupied and virtual orbitals.

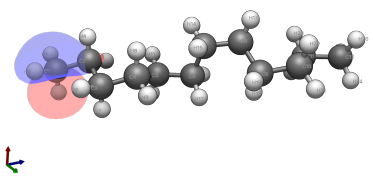
The COD algorithm effectively illustrates the separation and identification process of useful orbitals for further localized electron correlation energy calculations. This selection algorithm enables a more efficient computational approach by concentrating on orbitals that contribute to the subspace, based on their spatial attributes to the central domain, I.

Orbitals	Average Positions R_μ	Radial Extent σ_μ
Center I	15.119	0.366
Occupied Orbital #01	10.853	0.500
Occupied Orbital #02	10.847	2.028
Occupied Orbital #03	10.526	2.110
Occupied Orbital #04	11.349	2.186
Occupied Orbital #05	10.685	2.324
Virtual Orbital #01	10.834	1.947
Virtual Orbital #02	10.861	2.086
Virtual Orbital #04	10.894	1.967
Virtual Orbital #04	10.871	1.949
Virtual Orbital #05	10.714	3.337
Virtual Orbital #06	10.661	2.582
Virtual Orbital #07	11.285	2.568
Virtual Orbital #08	10.767	2.680
Virtual Orbital #09	9.958	3.102

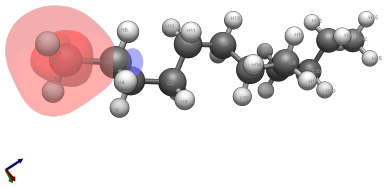
Table 5.3: The example of $C_{12}H_{26}$ molecule in 3-21G basis set: the characteristics (average position and radial extent) for the occupied orbitals (upper) and virtual orbitals (lower) under first localized molecular orbital (center I)



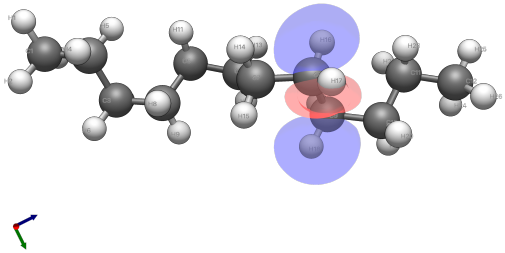
(a) Central Orbital $I23$



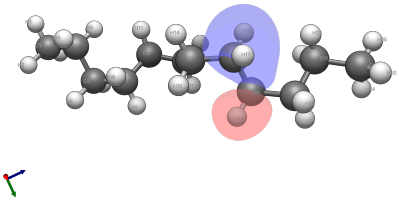
(b) One of the occupied orbitals under $I23$



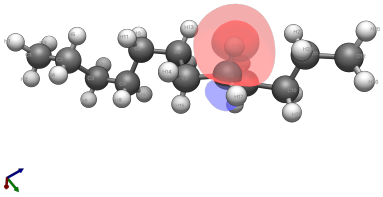
(c) One of the virtual orbitals under $I23$



(d) Central Orbital $I25$



(e) One of the occupied orbitals under $I25$



(f) One of the virtual orbitals under $I25$

Figure 5.3: The molecular orbital surface for selected occupied orbitals under different central I for $C_{12}H_{26}$ molecule in 3-21G basis sets.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This project successfully developed and analyzed efficient JK-Engine and Construction of Orbital Domain (COD), as foundational components for the cluster-in-molecule (CiM) approach. The JK-Engine, leveraging range-separated three-centered integrals complemented by Gaussian correction, was conducted for the evaluation of two-electron repulsion integrals critical to the computation of exchange and Coulomb matrices.

The performance and accuracy of the JK-Engine were rigorously benchmarked across various molecular models, including water clusters, alkene chains, and alkane chains, comparing favorably with the density-fitting methods implemented in PySCF. The analysis proved that the JK-Engine maintains minor computational errors from the resolution of identity (RI) and high efficiency in both random-access memory (RAM) and central processing unit (CPU) time particularly notable in the handling of larger molecular systems. Additionally, the algorithm displays linear scaling properties in timing and space complexities, marking a significant advancement in the efficiency of electronic structure calculations within the CiM framework.

Moreover, the COD method illustrates how to select the occupied and virtual spaces around a domain center i using the exchange matrix. This approach aims to ensure that the chosen orbital domains are significantly related to the occupied localized molecular orbitals, thereby improving the efficiency of ensuing calculations. By emphasizing the efficient segmentation of molecular systems into smaller, controllable subsystems, this method enables the scalable execution of further CiM calculations across extensive molecular structures.

6.2 Future Work and Recommendations

While empirical tests on the JK-Engine algorithm suggest an $O(1)$ space complexity for individual slices with varying input data sizes, this approach alone does not conclusively prove that the algorithm's additional memory usage remains constant. This limitation

arises because we applied data pruning techniques, including the elimination of zero elements, to reduce the data size manually and improve the efficiency for further calculations. In future stages, it will be essential to develop innovative strategies that inherently reduce memory demands while preserving computational integrity, without relying on manual reduction for data management.

Furthermore, there is a need for optimization in the initial phase where short-range integral slices $(\alpha B|x)_{sr}$ are computed and stored on the hard disk drive (HDD). The focus should be on effectively screening out dense integrals, aiming to achieve linear time complexity in these preparing steps, as opposed to the current quadratic time complexity.

Additionally, the integration between the JK-Engine and the Construction of Orbital Domains algorithm requires further enhancement. This improvement can be achieved by concentrating on computations for each occupied molecular orbital, using high-performance computing techniques such as transitioning to C++ and employing multi-threaded calculations.

All in all, the JK-Engine and the Construction of Orbital Domains stand out as essential tools in the initial phases of analysis. They are instrumental in efficiently calculating linear-scaling exchange matrices, and determining the occupied and virtual spaces surrounding a domain center i, particularly through the use of the exchange matrix for each localized molecular orbital. Moving forward, the Nooijen group intends to utilize these domains through a new algorithm in the cluster-in-molecule Coupled Cluster approach, aiming to enhance the computation of correlation energy while reducing computational costs and improving accuracy.

This thesis focused on obtaining J and K matrices in the context of calculations at a single molecular geometry. That is the usual situation when considering CiM calculations. However, in HF and (hybrid) DFT calculations the focus is often on geometry optimizations using analytical gradients, or the calculation of analytical Hessians to obtain vibrational frequencies. This requires analytical derivative formulas for the effective RI one-and two-electron integrals. The procedure that has been developed here is a clean mathematical procedure and it is relatively straightforward to obtain the relevant integrals, capitalizing on existing implementations. The additional complexity would arise from neglecting contributions that fall below a threshold, which arise at a number of instances in the algorithm. All of these issues can presumably be resolved, but it will take time and effort. The saga continues.

Bibliography

[1] Wei Li, Piotr Piecuch, and Jeffrey R Gour. Local correlation calculations using standard and renormalized coupled-cluster methods. In *AIP Conference Proceedings*, volume 1102, pages 68–113. American Institute of Physics, 2009.

[2] Wei Li and Piotr Piecuch. Improved design of orbital domains within the cluster-in-molecule local correlation framework: Single-environment cluster-in-molecule ansatz and its application to local coupled-cluster approach with singles and doubles. *The Journal of Physical Chemistry A*, 114(33):8644–8657, 2010.

[3] Wei Li and Piotr Piecuch. Multilevel extension of the cluster-in-molecule local correlation methodology: Merging coupled-cluster and møller- plesset perturbation theories. *The Journal of Physical Chemistry A*, 114(24):6721–6727, 2010.

[4] Zoltán Rolik and Mihály Kállay. A general-order local coupled-cluster method based on the cluster-in-molecule approach. *The Journal of chemical physics*, 135(10):104111, 2011.

[5] Zoltán Rolik, Lóránt Szegedy, István Ladjánszki, Bence Ladóczki, and Mihály Kállay. An efficient linear-scaling ccsd (t) method based on local natural orbitals. *The Journal of chemical physics*, 139(9):094105, 2013.

[6] Péter R Nagy, Gyula Samu, and Mihály Kállay. An integral-direct linear-scaling second-order møller–plesset approach. *Journal of chemical theory and computation*, 12(10):4897–4914, 2016.

[7] Ondřej Demel, Michael J Lecours, Richard Habrovskỳ, and Marcel Nooijen. Toward laplace mp2 method using range separated coulomb potential and orbital selective virtuals. *The Journal of Chemical Physics*, 155(15):154104, 2021.

[8] Michael Lecours. *Compact Sparse Coulomb Integrals using a Range-Separated Potential*. PhD thesis, UWSpace, 2021. Accessed: date-of-access.

[9] OpenAI. Chatgpt: Optimizing language models for dialogue, 2023. Software available from OpenAI.

[10] Wei Li, Zhigang Ni, and Shuhua Li. Cluster-in-molecule local correlation method for post-hartree–fock calculations of large systems. *Molecular Physics*, 114(9):1447–1460, 2016.

[11] Joonho Lee, Lin Lin, and Martin Head-Gordon. Systematically improvable tensor hypercontraction: Interpolative separable density-fitting for molecules applied to exact exchange, second-and third-order møller–plesset perturbation theory. *Journal of chemical theory and computation*, 16(1):243–263, 2019.

[12] Juan Carlos. Tensors for busy people. https://dev.to/juancarlospaco/tensors-for-busy-people-315k, May 2019. Accessed: [insert date of access].

[13] Matt Eding. Sparse matrices. https://matteding.github.io/2019/04/25/sparse-matrices/, April 2019. Accessed: [insert date of access].

[14] Masato Kobayashi and Hiromi Nakai. Extension of linear-scaling divide-and-conquer-based correlation method to coupled cluster theory with singles and doubles excitations. *The Journal of chemical physics*, 129(4):044103, 2008.

[15] N Flocke and Rodney J Bartlett. A natural linear scaling coupled-cluster method. *The Journal of chemical physics*, 121(22):10935–10944, 2004.

[16] Claudia Hampel and Hans-Joachim Werner. Local treatment of electron correlation in coupled cluster theory. *The Journal of chemical physics*, 104(16):6286–6297, 1996.

[17] Martin Schütz and Hans-Joachim Werner. Low-order scaling local electron correlation methods. iv. linear scaling local coupled-cluster (lccsd). *The Journal of Chemical Physics*, 114(2):661–681, 2001.

[18] E J Baerends, DE Ellis, and PJCP Ros. Self-consistent molecular hartree—fock—slater calculations i. the computational procedure. *Chemical Physics*, 2(1):41–51, 1973.

[19] Chr Møller and Milton S Plesset. Note on an approximation treatment for many-electron systems. *Physical review*, 46(7):618, 1934.

[20] Wei Li and Shuhua Li. Divide-and-conquer local correlation approach to the correlation energy of large molecules. *The Journal of chemical physics*, 121(14):6649–6657, 2004.

[21] Jun Yang, Yuki Kurashige, Frederick R Manby, and Garnet KL Chan. Tensor factorizations of local second-order møller–plesset theory. *The Journal of chemical physics*, 134(4):044123, 2011.

[22] Jun Yang, Garnet Kin-Lic Chan, Frederick R Manby, Martin Schütz, and Hans-Joachim Werner. The orbital-specific-virtual local coupled cluster singles and doubles method. *The Journal of chemical physics*, 136(14):144105, 2012.

[23] Hans-Joachim Werner and Martin Schütz. An efficient local coupled cluster method for accurate thermochemistry of large systems. *The Journal of chemical physics*, 135(14):144116, 2011.

[24] Attila Szabo and Neil S Ostlund. *Modern quantum chemistry: introduction to advanced electronic structure theory*. Courier Corporation, 2012.

[25] Rodney J Bartlett and John F Stanton. Applications of post-hartree—fock methods: A tutorial. *Reviews in computational chemistry*, pages 65–169, 1994.

[26] RJ Bartlett and M Musial. Rev. *Mod. Phys.*, 79:291, 2007.

[27] T Daniel Crawford and Henry F Schaefer. An introduction to coupled cluster theory for computational chemists. *Reviews in computational chemistry*, 14:33–136, 2000.

[28] Svein Sæbø and Peter Pulay. Local configuration interaction: An efficient approach for larger molecules. *Chemical physics letters*, 113(1):13–18, 1985.

[29] Frank Neese, Andreas Hansen, and Dimitrios G Liakos. Efficient and accurate approximations to the local coupled cluster singles doubles method using a truncated pair natural orbital basis. *The Journal of chemical physics*, 131(6):064103, 2009.

[30] Frank Neese, Frank Wennmohs, and Andreas Hansen. Efficient and accurate local approximations to coupled-electron pair approaches: An attempt to revive the pair natural orbital method. *The Journal of chemical physics*, 130(11):114108, 2009.

[31] Shuhua Li, Jing Ma, and Yuansheng Jiang. Linear scaling local correlation approach for solving the coupled cluster equations of large systems. *Journal of computational chemistry*, 23(2):237–244, 2002.

[32] Jerry L Whitten. Coulombic potential energy integrals and approximations. *The Journal of Chemical Physics*, 58(10):4496–4501, 1973.

[33] JA Jafri and JL Whitten. Electron repulsion integral approximations and error bounds: Molecular applications. *The Journal of Chemical Physics*, 61(5):2116–2121, 1974.

[34] Nelson HF Beebe and Jan Linderberg. Simplifications in the generation and transformation of two-electron integrals in molecular calculations. *International Journal of Quantum Chemistry*, 12(4):683–705, 1977.

[35] Richard A Friesner. Solution of self-consistent field electronic structure equations by a pseudospectral method. *Chemical physics letters*, 116(1):39–43, 1985.

[36] Richard A Friesner. Solution of the hartree–fock equations by a pseudospectral method: Application to diatomic molecules. *The Journal of chemical physics*, 85(3):1462–1468, 1986.

[37] Richard A Friesner. Solution of the hartree–fock equations for polyatomic molecules by a pseudospectral method. *The Journal of chemical physics*, 86(6):3522–3531, 1987.

[38] Frank Neese, Frank Wennmohs, Andreas Hansen, and Ute Becker. Efficient, approximate and parallel hartree–fock and hybrid dft calculations. a 'chain-of-spheres' algorithm for the hartree–fock exchange. *Chemical Physics*, 356(1-3):98–109, 2009.

[39] Sara IL Kokkila Schumacher, Edward G Hohenstein, Robert M Parrish, Lee-Ping Wang, and Todd J Martínez. Tensor hypercontraction second-order møller–plesset perturbation theory: Grid optimization and reaction energies. *Journal of chemical theory and computation*, 11(7):3042–3052, 2015.

[40] Robert M Parrish, Edward G Hohenstein, Todd J Martínez, and C David Sherrill. Tensor hypercontraction. ii. least-squares renormalization. *The Journal of chemical physics*, 137(22), 2012.

[41] Edward G Hohenstein, Robert M Parrish, C David Sherrill, and Todd J Martínez. Communication: Tensor hypercontraction. iii. least-squares tensor hypercontraction for the determination of correlated wavefunctions. *The Journal of chemical physics*, 137(22), 2012.

[42] Robert M Parrish, Edward G Hohenstein, Todd J Martínez, and C David Sherrill. Discrete variable representation in electronic structure theory: Quadrature grids for least-squares tensor hypercontraction. *The Journal of Chemical Physics*, 138(19), 2013.

[43] Jianfeng Lu and Lexing Ying. Compression of the electron repulsion integral tensor in tensor hypercontraction format with cubic scaling cost. *Journal of Computational Physics*, 302:329–335, 2015.

[44] Ondřej Demel, Michael J Lecours, and Marcel Nooijen. Further investigations into a laplace mp2 method using range separated coulomb potential and orbital selective virtuals: Multipole correction, osv extrapolation, and critical assessment. *The Journal of Chemical Physics*, 158(11), 2023.

[45] Frank Neese. An improvement of the resolution of the identity approximation for the formation of the coulomb matrix. *Journal of computational chemistry*, 24(14):1740–1747, 2003.

[46] Paul P Ewald. Die berechnung optischer und elektrostatischer gitterpotentiale. *Annalen der physik*, 369(3):253–287, 1921.

[47] Tomas Hansson, Chris Oostenbrink, and WilfredF van Gunsteren. Molecular dynamics simulations. *Current opinion in structural biology*, 12(2):190–196, 2002.

[48] BI Dunlap, JWD Connolly, and JR Sabin. On first-row diatomic molecules and local density models. *The Journal of Chemical Physics*, 71(12):4993–4999, 1979.

[49] Brett I Dunlap, JWD Connolly, and JR Sabin. On some approximations in applications of xα theory. *The Journal of Chemical Physics*, 71(8):3396–3402, 1979.

[50] BI Dunlap. Robust and variational fitting: Removing the four-center integrals from center stage in quantum chemistry. *Journal of Molecular Structure: THEOCHEM*, 529(1-3):37–40, 2000.

[51] Qiming Sun. Libcint: An efficient general integral library for g aussian basis functions. *Journal of computational chemistry*, 36(22):1664–1671, 2015.

[52] Qiming Sun, Timothy C Berkelbach, Nick S Blunt, George H Booth, Sheng Guo, Zhendong Li, Junzi Liu, James D McClain, Elvira R Sayfutyarova, Sandeep Sharma, et al. Pyscf: the python-based simulations of chemistry framework. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 8(1):e1340, 2018.

[53] Erin R Johnson and Axel D Becke. A post-hartree–fock model of intermolecular interactions. *The Journal of chemical physics*, 123(2), 2005.

[54] Haobo Liu. Design of efficient block-sparse data structures and associated tensor multiplications for applications in cluster in molecule electronic structure calculations, 2022. Accessed: date-of-access.

[55] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[56] Yanqing Chen, Timothy A Davis, William W Hager, and Sivasankaran Rajamanickam. Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. *ACM Transactions on Mathematical Software (TOMS)*, 35(3):1–14, 2008.

[57] Rodney J Bartlett. Coupled-cluster theory: An overview of recent developments. *Modern Electronic Structure Theory: Part II*, pages 1047–1131, 1995.

[58] Isaiah Shavitt and Rodney J Bartlett. *Many-body methods in chemistry and physics: MBPT and coupled-cluster theory*. Cambridge university press, 2009.

[59] Krishnan Raghavachari, Gary W Trucks, John A Pople, and Martin Head-Gordon. A fifth-order perturbation comparison of electron correlation theories. *Chemical Physics Letters*, 157(6):479–483, 1989.

[60] Aydin Buluç, Jeremy T Fineman, Matteo Frigo, John R Gilbert, and Charles E Leiserson. Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. In *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, pages 233–244, 2009.

[61] Gian-Carlo Wick. The evaluation of the collision matrix. *Physical review*, 80(2):268, 1950.

[62] Beatrice Weston Van Der Goetz. *Improving Wavefunction Efficiency by Tessellating Correlation Factors and Coupled State-Specific Optimization*. PhD thesis, University of California, Berkeley, 2021.

[63] Manuel Hodecker. *Development and application of Hermitian methods for molecular properties and excited electronic states*. PhD thesis, 2020.

APPENDICES

Appendix A

Pseudo Algorithm

A.1 Two-electron Three-index Integrals

A.1.1 Calculation Distances between Atoms

Algorithm 1 Calculation distances between atoms

Function: calc_distances($print\_opt = False$)
$coords \leftarrow self.mol.atom\_coords()$ ▷ Retrieve atom coordinates
Compute distance matrix
$distance\_matrix \leftarrow \sqrt{((coords[:, None, :] - coords[None, :, :])^2).sum(axis = 2)}$
if *print_opt* **then**
    print("distance check:", *distance_matrix*) ▷ Optional print
end if
return *distance_matrix*

A.1.2 Dynamic Grouping: Identify Atom Groups Based on Distance

Algorithm 2 Identify groups of atoms within a threshold distance

Function: identify_atom_groups($thresholdG = 3.0$)
$distances \leftarrow self.calc\_distances()$ ▷ Calculate pairwise distances between atoms
$groups \leftarrow [\,]$ ▷ Initialize list to store atom groups
$heavy\_atoms \leftarrow [i \text{ for } i, atom \text{ in } enumerate(mol.\_atom) \text{ if } atom! = H]$ ▷ Identify heavy atoms
for $start$ **in** $range(0, len(heavy\_atoms), self.n\_heavy)$ **do**
    $heavy\_group \leftarrow heavy\_atoms[start : start + self.n\_heavy]$
    $group \leftarrow heavy\_group.copy()$
    for $j, atom\_j$ **in** $enumerate(self.mol.\_atom)$ **do**
        if $atom\_j[0] == H$ **then** ▷ Consider only hydrogens for appending
            if any($distances[heavy][j] < thresholdG$ for heavy atoms) **then**
                $group.append(j)$ ▷ Add hydrogen close to any heavy atom
            end if
        end if
    end for
    $groups.append(sorted(group))$ ▷ Sort and add the group to groups list
end for
return $groups$ ▷ Return list of atom groups

A.1.3 Function: Eliminate Small Entries from Sparse Matrices

Algorithm 3 Elimination of Small Entries in Sparse Matrices

Function EliminateZeros_COO(*sparse*, *threshold*) ▷ Applicable for COO, CSR, CSC formats
Input:
    sparse - A sparse matrix in COO, CSR, or CSC format.
    threshold - A threshold value below which elements are considered negligible.
Output:
    A COO format sparse matrix with all elements below the threshold eliminated.
Begin
Identify elements of *sparse.data* that are less than *threshold* in magnitude.
Set these elements to 0 to effectively ignore them.
Invoke *sparse.eliminate_zeros*() to remove these zeroed elements from the data.
Convert *sparse* to COO format, if not already, to standardize the output.
return The resulting sparse matrix in COO format.
End

A.1.4 Recompound Sparse Matrix from aBX to aXB Format

Algorithm 4 Recompound Sparse Matrix from (aB;x) to (ax;B)Format

1: **Function:** recompound_aBx_to_axB($Z_{coo}, nao, n_B, naux$)
2: Generate a 2D grid of indices for a and B dimensions using *np.meshgrid*
3: Flatten the grid to create a linear index mapping for (a, B) pairs
4: For each non-zero element in Z_{coo}, find the corresponding a and B indices
5: Calculate the new row index by combining a index and Z_{coo} column index with $naux$
6: Create a new COO matrix R_{coo} with the updated indices and the same data as Z_{coo}
7: **return** R_{coo}

A.1.5 Computation of the Short-range Integrals $(\alpha B|x)$

Algorithm 5 Compute (aB|x) integral using sparse matrix format

Function: integral_aBx($p0, p1$)
Initialize $data\_aBx, rows\_aBx, cols\_aBx$ as empty lists
Initialize nA_lst with [0] ▷ $\mathbf{n_{ao}}$ in α dimension
for $j \leftarrow 0$ to $natm$ with step $ngatm$ **do** ▷ slicing by A in α dimension
    Determine AO slice range and shell slice range for atom block A
    $n_X\_slice \leftarrow [0]$ ▷ $\mathbf{n_{naux}}$ in X dimension
    for $k \leftarrow 0$ to $natm$ with step $ngatm$ **do** ▷ slicing by X in x dimension
        Determine aux AO slice range and shell slice range for atom block X
        Determine slice integral, $(A; B; X) \leftarrow slice\_int2e3c\_sr\_gtg(slice\_range)$
        Determine dimensions of slice integrals: nA, nB, nX
        Reshape to a 2D matrix: $(AB; X) \leftarrow slice\_int.reshape((nA \times nB, nX))$
        Convert $(AB; X)$ to COO format: ABX_coo
        Append data to integral data list:
            $data\_aBx.append(ABX\_coo.data)$
        Update row indices:
            $rows\_aBx.append(ABX\_coo.row + np.sum(nA\_lst) \times nB)$
        Update column indices:
            $cols\_aBx.append(ABX\_coo.col + np.sum(nX\_lst))$
        Update $\mathbf{n_{naux}}$, n_X
    end for
    Update $\mathbf{n_{ao}}$, n_A
end for
Combine aggregated data, rows, and cols
Create new COO matrix for (aB|x) integral
$aBx\_coo \leftarrow coo\_matrix((combined\_data, (combined\_row, combined\_col)), shape)$
return aBx_coo

A.1.6 Processing Atom Groups for Integral Calculations

Algorithm 6 Process Atom Groups and Calculate Integrals

$groups = identify\_atom\_groups(\ )$
for each $group$ in $groups$ **do**
    $(p0, \_) \leftarrow mol\_slice(group[0])$ ▷ Get start basis index for group
    $(\_, p1) \leftarrow mol\_slice(group[-1])$ ▷ Get end basis index for group
    $(ao_0, \_) \leftarrow mol\_slice\_ao(group[0])$ ▷ Get AO start index
    $(\_, ao_1) \leftarrow mol\_slice\_ao(group[-1])$ ▷ Get AO end index
    $aBx\_coo \leftarrow integral\_aBx(p0, p1)$ ▷ Calculate $(aB|x)$ integrals
    $aBx\_data.append(aBx\_coo.nnz)$ ▷ Store non-zero elements count
    Save aBx_coo to file in '$folder/int3c2e\_sr/aBx\_group[0].npz$'
    $axB\_coo \leftarrow recompound\_aBx\_to\_axB(aBx\_coo, nao, (ao_1 - ao_0), naux)$
    Save axB_coo to file in '$folder/int3c2e\_sr/axB\_group[0].npz$'
end for

A.1.7 Computation and Storage of P and O

Algorithm 7 Construction of P and O Matrices

Function: Construction P&O
Initialize P matrix: $P \leftarrow 0$
Initialize O matrix: $O \leftarrow 0$
for $i \leftarrow 0$ to $natm$ with step $ngatm$ **do**
    Determine shell slice range for block B: b_0, b_1
    for $j \leftarrow 0$ to $natm$ with step $ngatm$ **do**
        Determine shell slice range for block A: a_0, a_1
        Define slice range: $(a0, a1, b0, b1, mol.nbas, mol.nbas + auxmol.nbas)$
        Calculate full-range integrals: $FR \leftarrow slice\_int2e3c\_fr(slice\_range)$
        Calculate long-range integrals: $LR \leftarrow slice\_int2e3c\_lr(slice\_range)$
        Calculate Gaussian correction: $GC \leftarrow slice\_int2e3c\_gc(slice\_range)$
        Compute short-range integrals: $(AB;x)_{sr} \leftarrow FR - LR + GC$
        Determine dimensions: $n_A, n_B \leftarrow sr.shape$
        Reshape to matrix: $(AB;x)_{sr} \leftarrow sr.reshape((nA \times nB, naux))$
        Convert to CSC format: $csc\_matrix((AB;x)_{sr})$
        Reshape full-range integrals: $(AB;x)_{fr} \leftarrow FR.reshape((n_A \times n_B, naux))$
        Convert to CSC format: $csc\_matrix((AB;x)_{fr})$
        Update P matrix: $P \mathrel{+}= (AB;x)_{sr}^T \cdot (AB;x)_{sr}$
        Update O matrix: $O \mathrel{+}= (AB;x)_{fr}^T \cdot (AB;x)_{sr}$
    end for
end for
return P, O

A.1.8 Pseudo Inverse of Matrix

Algorithm 8 Pseudo Inverse Calculation

Function: pinverse(*matrix*, $threshP = 1e-6$)
Compute the eigenvalues and eigenvectors of *matrix*:
$[eigenvalues, eigenvectors] \leftarrow$ np.linalg.eig(*matrix*)
Initialize the diagonal matrix for inverse eigenvalues: $diagonal\_matrix \leftarrow 0$
for each eigenvalue λ in *eigenvalues* **do**
    if $|\lambda| < thresh$ **then**
        Treat the eigenvalue as 0: $\lambda \leftarrow 0$
    else
        Invert the eigenvalue: $\lambda \leftarrow 1/\lambda$
    end if
end for
Diagonal matrix of modified eigenvalues: $diagonal\_matrix \leftarrow$ np.diag(*eigenvalues*)
Pseudo inverse: $matrix\_inv \leftarrow eigenvectors \cdot diagonal\_matrix \cdot eigenvectors^T$
return *matrix_inv*

A.2 Algorithm: Exchange Integrals

A.2.1 Algorithm: Association of LMOs with Atom Blocks

Algorithm 9 Determination of Block Boundaries and Associated LMO Indices

Function getBlocksBV(lmo)
Obtain $natm$, $ngatm$ from the molecular structure
Initialize $groups \leftarrow identify\_atom\_groups$
Initialize $BlockB \leftarrow \{\}$, $BlockV \leftarrow \{\}$ ▷ Dictionaries for block B and block V
Compute $max\_ind \leftarrow argmax(|lmo|)$ ▷ Index with max absolute value in each LMO
for each $group$ in $groups$ **do**
    $b0, \_ \leftarrow$ MolSliceAO($group[0]$)
    $\_, b1 \leftarrow$ MolSliceAO($group[-1]$)
    $BlockB[BlockIdx] \leftarrow (b0, b1)$ ▷ Store AO boundaries for the current block
    $BlockV[BlockIdx] \leftarrow [j$ **for** j in $range(len(max\_ind))$ **if** $b0 \leq max\_ind[j] < b1]$ ▷ LMO-index associated with the block
end for
return $BlockB, BlockV$

A.2.2 Algorithm: Exchange Integrals by Slicing

Algorithm 10 Calculation of Exchange Integrals **K** by slicing

Function getExchangeSlice($int3cFolder$, Q_{xy}, dm, $threshZ$)
Initialize variables: $thresh\_Z \leftarrow 1e-5$, $L \leftarrow$ Localize Orbitals
Obtain dimensions nao, $naux$, $natm$, $ngatm$, and convert Q_{xy} to sparse matrix
Initialize $K_{\alpha\beta}$ as a sparse matrix of shape (nao, nao)
Get AO boundaries and associated LMO indices using $getBlocksBV(L)$
for each block V index in $blockV$ **do**
    $LMO_V \leftarrow BlockV[Idx_V]$ ▷ Assciated LMO index in each block V
    $L_{\beta V} \leftarrow L[:, LMO_V]$
    $I(\alpha x; V) \leftarrow \text{CSC}(nao * naux, len(L_V[0]))$
    for each atom block boundary index (β_0, β_1) in $blockB$ **do**
        $L_{BV} \leftarrow \text{CSC}(L_{\beta v}[\beta_0 : \beta_1, :])$ & EliminateZeros$(L_{BV}, threshZ)$
        if $max(abs(L_{BV})) > threshZ$ **then**
            $(\alpha x, B) \leftarrow$ LoadSparseMatrix(int3cFolder, B)
            $I(\alpha x, V)$ += CSC$(\alpha x, B) \cdot L_{BV}$ & EliminateZeros$(I(\alpha x, V), threshZ)$
        end if
    end for
    for $\nu \leftarrow 0$ **to** $range(len(L_V[0]))$ **do**
        $I(\alpha x;) \leftarrow I[:, \nu]$
        $I(\alpha; x) \leftarrow \text{Reshape}(I(\alpha x;), nao, naux)$
        $Z \leftarrow$ CalculateSparsityMask$(I(\alpha; x), threshZ)$
        $\tilde{Q} = Z * Q_{xy} * Z^T = (Z \cdot Z^T) * Q_{xy}$ ▷ Hadamard Product
        $I_2 = \tilde{Q} \cdot I(\alpha; x)^T$
        $K_{\alpha\beta}$ += $I(\alpha; x) \cdot I_2$
    end for
end for
return $K_{\alpha\beta}$

A.2.3 Calculate Sparsity Mask

Algorithm 11 Calculate Sparsity Mask

Procedure CalculateSparsityMask($I(\alpha; X)$, $naux$, $threshZ$)
$mask \leftarrow$ CreateEmptyArray($Size : naux$)
for $i \leftarrow 0$ **to** $naux$ **do**
    if $max(|e|_{\alpha,i}) < threshZ$ **then**
        $mask[i] \leftarrow 0$ ▷ Mark element as sparse
    else
        $mask[i] \leftarrow 1$ ▷ Mark element as significant
    end if
end for
return $mask$
EndProcedure

A.3 Algorithm: Coulomb Integrals

Algorithm 12 Calculation of Coulomb Matrix **J** in a Slice

Function getCoulombSlice($int3cFolder$, Q_{xy}, dm)
Obtain nao, $naux$, $natm$, $ngatm$
Initialize $groups \leftarrow identify\_atom\_groups$
$J_{\alpha\beta} \leftarrow \text{EmptyArray}(nao, 0)$ ▷ Initialize total Coulomb matrix
for each $group$ in $groups$ **do**
    $(p0, \_) \leftarrow mol\_slice(group[0])$ ▷ Get start basis index for group
    $(\_, p1) \leftarrow mol\_slice(group[-1])$ ▷ Get end basis index for group
    $D_{\alpha B;} \leftarrow \text{ReshapeAsVector}(dm[:, p_0 : p_1])$
    $(aB; x) \leftarrow \text{LoadSparseMatrix}(int3c\_folder, i)$
    $I1(x;) \mathrel{+}= \cdot(aB; x)^T \cdot D_{\alpha B;}$ ▷ Update intermediate matrix
end for
$I2(x) = Q_{xy} \cdot I1(y;)$
for each $group$ in $groups$ **do**
    $(p0, \_) \leftarrow mol\_slice(group[0])$
    $(\_, p1) \leftarrow mol\_slice(group[-1])$
    $n_B \leftarrow p_1 - p_0$
    $(aB; x) \leftarrow \text{LoadSparseMatrix}(int3cFolder, i)$
    $J_{slice} \leftarrow (aB; x) \cdot I2(x;)$
    $J_{a;B} \leftarrow \text{Reshape}(nao, n_B)$
    $J_{\alpha\beta} \leftarrow \text{Concatenate}(J_{\alpha\beta}, J_{a;B})$ ▷ Concatenate along columns
end for
return $J_{\alpha\beta}$

A.4 Algorithm: Construction of Orbital Domain

A.4.1 AO-based Construct Orbital Domain

Algorithm 13 AO-based Construct Orbital Domain

$P_{\alpha\beta} \leftarrow S^{-1}_{\alpha\beta} - D_{\alpha\beta}$
for $i \leftarrow 0$ **to** n_v **do** ▷ n_v: number of LMO, $L_{\alpha\mu}$
  $V_1, n_a \leftarrow \mathrm{CDO}(L_{\alpha i}, P_{\alpha\beta}, S_{\alpha\beta}, thresh)$ ▷ V_1: Virtual1, n_a: number of virturals
  $\bar{\bar{D}} = P_{\alpha\beta} - V_1 V_1^T$
  $O_1, n_j \leftarrow \mathrm{CDO}(L_{\alpha i}, \tilde{D}, S_{\alpha\beta}, thresh)$ ▷ O_1: Occupied1, n_j: number of occupieds
  $\tilde{D} = D_{\alpha\beta} - L_{\alpha i} L_{\alpha i}^T$
  $V_2, n_b \leftarrow \mathrm{CDO}(L_{\alpha i}, \bar{\bar{D}}, S_{\alpha\beta}, thresh)$ ▷ V_2: Virtual2, n_b: number of virtuals
  $O = L_{\alpha i} + O_1$
  $V = V_1 + V_2$
end for
return O, V ▷ O: occupied space, V: virtual space

A.4.2 Construct Orbital Domain

Algorithm 14 Construct Orbital Domain

$L \leftarrow S_{\alpha\beta} = L_{\alpha\mu} L_{\alpha\mu}^T$ ▷ Cholesky decomposition of $S_{\alpha\beta}$
Procedure CDO($C_{\alpha\mu}$, $P_{\alpha\beta}$, $L_{\alpha\mu}$, η, $thresh$)
$D_{new} \leftarrow C_{\alpha\mu} C_{\alpha\mu}^T$
$K \leftarrow getExchangeSlice(int3cFolder, Q_{xy}, D_{new}, threshZ)$
$\tilde{K}_{zz} \leftarrow (L^T P) K (P L)$
$\tilde{Y}_{zk} \leftarrow \tilde{K}_{zz} = \tilde{Y}_{zk} \tilde{Y}_{zk}^T$ ▷ Cholesky decomposition of $\tilde{K}_{zz}$
$max\_per\_column \leftarrow \max(\tilde{Y},\ \text{along axis } = 0)$
if $max\_per\_column > \eta$ **then**
    Identify corresponding eigenvector in $\tilde{Y}_{zk}$
    Keep this eigenvector $\tilde{Y}_{zk}[:, max\_per\_column]$
end if
$\tilde{X}_{\alpha k} \leftarrow \tilde{X}_{\alpha k} = \sum_z (L^T)_{\alpha z}^{-1} \tilde{Y}_{zk}$ ▷ Transform to AO basis
$M \leftarrow M = \tilde{X}^T S \tilde{X}$
$X \leftarrow X = \tilde{X} M^{-1/2}$ ▷ Löwdin Orthonormalization
return X, n ▷ X: selected orbitals, n: number of selected orbitals
EndProcedure

Appendix B

Additional Data

B.1 Variations on Alpha Values

α	$K_{\mathrm{M}} - K_{\mathrm{pyscf}}$	$J_{\mathrm{M}} - J_{\mathrm{pyscf}}$	$E_{\mathrm{ex,\ M}} - E_{\mathrm{ex,\ pyscf}}$	$E_{\mathrm{elec,\ M}} - E_{\mathrm{elec,\ pyscf}}$
0.1	3.68×10^{-6}	4.81×10^{-5}	4.36×10^{-8}	9.49×10^{-7}
0.2	6.57×10^{-5}	1.01×10^{-3}	9.21×10^{-7}	3.23×10^{-5}
0.3	2.53×10^{-4}	4.13×10^{-3}	2.25×10^{-6}	2.09×10^{-4}
0.4	4.62×10^{-4}	8.17×10^{-3}	2.03×10^{-5}	5.70×10^{-4}
0.5	5.87×10^{-4}	1.13×10^{-2}	6.47×10^{-5}	1.01×10^{-3}
0.6	6.47×10^{-4}	1.34×10^{-2}	1.37×10^{-4}	1.43×10^{-3}
0.7	7.05×10^{-4}	1.50×10^{-2}	2.21×10^{-4}	1.78×10^{-3}
0.8	7.97×10^{-4}	1.65×10^{-2}	2.92×10^{-4}	2.05×10^{-3}
0.9	9.22×10^{-4}	1.81×10^{-2}	3.44×10^{-4}	2.27×10^{-3}
1.0	1.06×10^{-3}	1.97×10^{-2}	3.77×10^{-4}	2.47×10^{-3}
1.1	1.20×10^{-3}	2.13×10^{-2}	3.99×10^{-4}	2.67×10^{-3}
1.2	1.31×10^{-3}	2.28×10^{-2}	4.18×10^{-4}	2.86×10^{-3}
1.3	1.42×10^{-3}	2.43×10^{-2}	4.38×10^{-4}	3.06×10^{-3}
1.4	1.53×10^{-3}	2.58×10^{-2}	4.57×10^{-4}	3.26×10^{-3}
1.5	1.64×10^{-3}	2.73×10^{-2}	4.77×10^{-4}	3.46×10^{-3}
1.6	1.75×10^{-3}	2.88×10^{-2}	4.96×10^{-4}	3.65×10^{-3}
1.7	1.86×10^{-3}	3.03×10^{-2}	5.16×10^{-4}	3.85×10^{-3}
1.8	1.97×10^{-3}	3.18×10^{-2}	5.36×10^{-4}	4.05×10^{-3}
1.9	2.08×10^{-3}	3.33×10^{-2}	5.55×10^{-4}	4.24×10^{-3}
2.0	2.19×10^{-3}	3.48×10^{-2}	5.75×10^{-4}	4.44×10^{-3}

Table B.1: Comparative Analysis of Error Metrics Between JK-Engine Algorithm with the metric matrix $M_{\alpha\beta}$ and Density Fitting Object from PySCF for Water Monomer H_2O under various α values in cc-pVTZ basis set and cc-pvtz-jkfit auxiliary basis set: This table presents the differences in calculated values for exchange integrals ($K_{\mathrm{M}} - K_{\mathrm{pyscf}}$), Coulomb integrals ($J_{\mathrm{M}} - J_{\mathrm{pyscf}}$), exchange energies ($E_{\mathrm{ex,\ M}} - E_{\mathrm{ex,\ pyscf}}$), and electronic energies ($E_{\mathrm{elec,\ M}} - E_{\mathrm{elec,\ pyscf}}$) utilizing different α parameters.

α	$K_{\mathrm{M}} - K_{\mathrm{pyscf}}$	$J_{\mathrm{M}} - J_{\mathrm{pyscf}}$	$E_{\mathrm{ex,\ M}} - E_{\mathrm{ex,\ pyscf}}$	$E_{\mathrm{elec,\ M}} - E_{\mathrm{elec,\ pyscf}}$
0.1	2.96×10^{-5}	9.61×10^{-4}	4.50×10^{-6}	4.99×10^{-5}
0.2	6.57×10^{-5}	2.15×10^{-3}	1.73×10^{-6}	2.77×10^{-4}
0.3	2.53×10^{-4}	5.57×10^{-3}	4.62×10^{-6}	6.91×10^{-4}
0.4	4.62×10^{-4}	9.90×10^{-3}	4.06×10^{-5}	1.47×10^{-3}
0.5	5.87×10^{-4}	1.33×10^{-2}	1.30×10^{-4}	2.39×10^{-3}
0.6	6.47×10^{-4}	1.56×10^{-2}	2.75×10^{-4}	3.30×10^{-3}
0.7	7.05×10^{-4}	1.74×10^{-2}	4.41×10^{-4}	4.07×10^{-3}
0.8	7.97×10^{-4}	1.92×10^{-2}	5.85×10^{-4}	4.67×10^{-3}
0.9	9.22×10^{-4}	2.10×10^{-2}	6.88×10^{-4}	5.16×10^{-3}
1.0	1.06×10^{-3}	2.28×10^{-2}	7.53×10^{-4}	5.62×10^{-3}
1.1	1.20×10^{-3}	2.46×10^{-2}	7.99×10^{-4}	6.06×10^{-3}
1.2	1.31×10^{-3}	2.63×10^{-2}	8.36×10^{-4}	6.50×10^{-3}
1.3	1.42×10^{-3}	2.79×10^{-2}	8.75×10^{-4}	6.94×10^{-3}
1.4	1.53×10^{-3}	2.95×10^{-2}	9.14×10^{-4}	7.38×10^{-3}
1.5	1.64×10^{-3}	3.11×10^{-2}	9.53×10^{-4}	7.82×10^{-3}
1.6	1.75×10^{-3}	3.27×10^{-2}	9.92×10^{-4}	8.26×10^{-3}
1.7	1.86×10^{-3}	3.43×10^{-2}	1.03×10^{-3}	8.70×10^{-3}
1.8	1.97×10^{-3}	3.59×10^{-2}	1.07×10^{-3}	9.14×10^{-3}
1.9	2.08×10^{-3}	3.75×10^{-2}	1.11×10^{-3}	9.58×10^{-3}
2.0	2.19×10^{-3}	3.91×10^{-2}	1.15×10^{-3}	1.00×10^{-2}

Table B.2: Comparative Analysis of Error Metrics Between JK-Engine Algorithm with the metric matrix M_{xy} and Density Fitting Object from PySCF for Water Dimer $(H_2O)_2$ under various α values in cc-pVTZ basis set and cc-pvtz-jkfit auxiliary basis set: This table presents the differences in calculated values for exchange integrals ($K_{\mathrm{M}} - K_{\mathrm{pyscf}}$), Coulomb integrals ($J_{\mathrm{M}} - J_{\mathrm{pyscf}}$), exchange energies ($E_{\mathrm{ex,\ M}} - E_{\mathrm{ex,\ pyscf}}$), and electronic energies ($E_{\mathrm{elec,\ M}} - E_{\mathrm{elec,\ pyscf}}$) utilizing different α parameters.

α	$K_{\mathrm{M}} - K_{\mathrm{pyscf}}$	$J_{\mathrm{M}} - J_{\mathrm{pyscf}}$	$E_{\mathrm{ex,\ M}} - E_{\mathrm{ex,\ pyscf}}$	$E_{\mathrm{elec,\ M}} - E_{\mathrm{elec,\ pyscf}}$
0.1	2.01×10^{-5}	1.34×10^{-4}	2.77×10^{-6}	1.76×10^{-5}
0.2	4.56×10^{-5}	2.07×10^{-3}	3.20×10^{-6}	3.20×10^{-4}
0.3	1.57×10^{-4}	5.95×10^{-3}	8.17×10^{-6}	1.01×10^{-3}
0.4	2.86×10^{-4}	9.35×10^{-3}	8.42×10^{-5}	1.73×10^{-3}
0.5	4.09×10^{-4}	1.19×10^{-2}	1.79×10^{-4}	2.30×10^{-3}
0.6	5.27×10^{-4}	1.41×10^{-2}	2.69×10^{-4}	2.70×10^{-3}
0.7	6.44×10^{-4}	1.61×10^{-2}	3.17×10^{-4}	3.02×10^{-3}
0.8	7.58×10^{-4}	1.82×10^{-2}	3.35×10^{-4}	3.36×10^{-3}
0.9	8.68×10^{-4}	2.02×10^{-2}	3.44×10^{-4}	3.73×10^{-3}
1.0	9.68×10^{-4}	2.24×10^{-2}	3.58×10^{-4}	4.13×10^{-3}
1.1	1.06×10^{-3}	2.45×10^{-2}	3.79×10^{-4}	4.54×10^{-3}
1.2	1.15×10^{-3}	2.66×10^{-2}	4.11×10^{-4}	4.88×10^{-3}
1.3	1.24×10^{-3}	2.86×10^{-2}	4.40×10^{-4}	5.14×10^{-3}
1.4	1.34×10^{-3}	3.05×10^{-2}	5.02×10^{-4}	5.49×10^{-3}
1.5	1.44×10^{-3}	3.29×10^{-2}	5.52×10^{-4}	5.76×10^{-3}
1.6	1.55×10^{-3}	3.55×10^{-2}	6.03×10^{-4}	5.97×10^{-3}
1.7	1.67×10^{-3}	3.82×10^{-2}	6.59×10^{-4}	6.49×10^{-3}
1.8	1.80×10^{-3}	4.10×10^{-2}	7.03×10^{-4}	6.87×10^{-3}
1.9	1.95×10^{-3}	4.39×10^{-2}	7.33×10^{-4}	7.30×10^{-3}
2.0	2.08×10^{-3}	4.69×10^{-2}	7.79×10^{-4}	7.75×10^{-3}

Table B.3: Comparative Analysis of Error Metrics Between JK-Engine Algorithm with the metric matrix $M_{\alpha\beta}$ and Density Fitting Object from PySCF for Ethane C_2H_6 under various α values in cc-pVTZ basis set and cc-pvtz-jkfit auxiliary basis set: This table presents the differences in calculated values for exchange integrals ($K_{\mathrm{M}} - K_{\mathrm{pyscf}}$), Coulomb integrals ($J_{\mathrm{M}} - J_{\mathrm{pyscf}}$), exchange energies ($E_{\mathrm{ex,\ M}} - E_{\mathrm{ex,\ pyscf}}$), and electronic energies ($E_{\mathrm{elec,\ M}} - E_{\mathrm{elec,\ pyscf}}$) utilizing different α parameters.

α	$K_{\mathrm{M}} - K_{\mathrm{pyscf}}$	$J_{\mathrm{M}} - J_{\mathrm{pyscf}}$	$E_{\mathrm{ex,\ M}} - E_{\mathrm{ex,\ pyscf}}$	$E_{\mathrm{elec,\ M}} - E_{\mathrm{elec,\ pyscf}}$
0.1	1.88×10^{-5}	1.33×10^{-4}	7.65×10^{-7}	1.93×10^{-5}
0.2	1.15×10^{-4}	2.42×10^{-3}	5.40×10^{-6}	3.39×10^{-4}
0.3	4.01×10^{-4}	8.34×10^{-3}	3.23×10^{-5}	1.13×10^{-3}
0.4	6.90×10^{-4}	1.44×10^{-2}	1.03×10^{-4}	2.01×10^{-3}
0.5	8.86×10^{-4}	1.88×10^{-2}	2.18×10^{-4}	2.78×10^{-3}
0.6	1.01×10^{-3}	2.21×10^{-2}	3.25×10^{-4}	3.40×10^{-3}
0.7	1.12×10^{-3}	2.52×10^{-2}	3.96×10^{-4}	3.90×10^{-3}
0.8	1.22×10^{-3}	2.80×10^{-2}	4.38×10^{-4}	4.31×10^{-3}
0.9	1.31×10^{-3}	3.05×10^{-2}	4.61×10^{-4}	4.63×10^{-3}
1.0	1.41×10^{-3}	3.28×10^{-2}	4.86×10^{-4}	4.94×10^{-3}
1.1	1.50×10^{-3}	3.49×10^{-2}	5.07×10^{-4}	5.17×10^{-3}
1.2	1.58×10^{-3}	3.67×10^{-2}	5.22×10^{-4}	5.36×10^{-3}
1.3	1.66×10^{-3}	3.83×10^{-2}	5.35×10^{-4}	5.52×10^{-3}
1.4	1.74×10^{-3}	3.97×10^{-2}	5.45×10^{-4}	5.65×10^{-3}
1.5	1.81×10^{-3}	4.09×10^{-2}	5.54×10^{-4}	5.75×10^{-3}
1.6	1.88×10^{-3}	4.20×10^{-2}	5.61×10^{-4}	5.83×10^{-3}
1.7	1.95×10^{-3}	4.29×10^{-2}	5.66×10^{-4}	5.89×10^{-3}
1.8	2.01×10^{-3}	4.36×10^{-2}	5.71×10^{-4}	5.94×10^{-3}
1.9	2.07×10^{-3}	4.42×10^{-2}	5.74×10^{-4}	5.98×10^{-3}
2.0	2.13×10^{-3}	4.47×10^{-2}	5.77×10^{-4}	6.01×10^{-3}

Table B.4: Comparative Analysis of Error Metrics Between JK-Engine Algorithm with the metric matrix $M_{\alpha\beta}$ and Density Fitting Object from PySCF for Ethylene C_2H_4 under various α values in cc-pVTZ basis set and cc-pvtz-jkfit auxiliary basis set: This table presents the differences in calculated values for exchange integrals ($K_{\mathrm{M}} - K_{\mathrm{pyscf}}$), Coulomb integrals ($J_{\mathrm{M}} - J_{\mathrm{pyscf}}$), exchange energies ($E_{\mathrm{ex,\ M}} - E_{\mathrm{ex,\ pyscf}}$), and electronic energies ($E_{\mathrm{elec,\ M}} - E_{\mathrm{elec,\ pyscf}}$) utilizing different α parameters.

B.2 Variations on threshP

threshP	$(R_1)_{max}$	$(R_2)_{max}$	$K_{\mathrm{Q}} - K_{\mathrm{pyscf}}$	$J_{\mathrm{Q}} - J_{\mathrm{pyscf}}$	$E_{\mathrm{ex,\ Q}} - E_{\mathrm{ex}}$	$E_{\mathrm{elec,\ Q}} - E_{\mathrm{elec}}$
10^{-1}	43.8	1.20×10^{-1}	6.35×10^{-4}	1.01×10^{-2}	4.16×10^{-5}	7.93×10^{-4}
10^{-2}	43.8	7.41×10^{-2}	3.89×10^{-4}	8.42×10^{-3}	1.43×10^{-4}	8.25×10^{-4}
10^{-3}	43.8	1.42×10^{-2}	4.01×10^{-4}	6.25×10^{-3}	1.32×10^{-4}	9.77×10^{-4}
10^{-4}	43.8	4.34×10^{-3}	6.95×10^{-4}	5.66×10^{-3}	5.24×10^{-4}	4.57×10^{-3}
10^{-5}	43.8	4.43×10^{-3}	4.01×10^{-4}	6.19×10^{-3}	9.67×10^{-5}	7.56×10^{-4}
10^{-6}	43.8	1.22×10^{-2}	3.22×10^{-4}	6.57×10^{-3}	1.82×10^{-4}	1.30×10^{-3}
10^{-7}	43.8	8.78×10^{-2}	2.98×10^{-4}	6.63×10^{-3}	1.53×10^{-04}	1.40×10^{-3}
10^{-8}	43.8	6.71×10^{-2}	2.99×10^{-4}	6.71×10^{-3}	2.49×10^{-5}	8.35×10^{-4}

Table B.5: Error Analysis Between JK-Engine with the metric matrix Q_{xy} and Density Fitting Object in PySCF for Various Threshold in the inversion of P_{xy} Matrix, $threshP$ for Water Monomer (H_2O): This table indicates the differences in computed values for exchange and Coulomb integrals, exchange energies, and electronic energies between the JK-Engine algorithm, and the Density Fitting approach in PySCF. These calculations were performed under the setting of $\alpha = 0.6$ with the cc-pVTZ basis set and the cc-pvtz-jkfit auxiliary basis set.

threshP	$(R_1)_{max}$	$(R_2)_{max}$	$K_{\mathrm{Q}} - K_{\mathrm{pyscf}}$	$J_{\mathrm{Q}} - J_{\mathrm{pyscf}}$	$E_{\mathrm{ex,\ Q}} - E_{\mathrm{ex}}$	$E_{\mathrm{elec,\ Q}} - E_{\mathrm{elec}}$
10^{-1}	43.8	3.02	7.56×10^{-04}	1.21×10^{-02}	9.25×10^{-6}	9.57×10^{-03}
10^{-2}	43.8	3.02	8.85×10^{-04}	1.21×10^{-02}	3.93×10^{-4}	1.06×10^{-02}
10^{-3}	43.8	3.02	7.66×10^{-04}	1.18×10^{-02}	1.63×10^{-4}	5.69×10^{-03}
10^{-4}	43.8	3.02	1.07×10^{-03}	1.11×10^{-02}	9.58×10^{-04}	3.81×10^{-03}
10^{-5}	43.8	3.02	9.14×10^{-04}	9.50×10^{-03}	8.56×10^{-05}	6.34×10^{-03}
10^{-6}	43.8	3.07	1.11×10^{-03}	1.02×10^{-02}	5.41×10^{-04}	1.90×10^{-02}
10^{-7}	43.8	3.08	1.12×10^{-03}	1.02×10^{-02}	4.75×10^{-04}	1.92×10^{-02}
10^{-8}	43.8	3.03	1.11×10^{-03}	1.03×10^{-02}	2.57×10^{-04}	1.78×10^{-02}

Table B.6: Error Analysis Between JK-Engine with the metric matrix Q_{xy} and Density Fitting Object in PySCF for Various Threshold in the inversion of P_{xy} Matrix $threshP$ for Water Dimer $(H_2O)_2$: This table indicates the differences in computed values for exchange and Coulomb integrals, exchange energies, and electronic energies between the JK-Engine algorithm, and the Density Fitting approach in PySCF. These calculations were performed under the setting of $\alpha = 0.6$ with the cc-pVTZ basis set and the cc-pvtz-jkfit auxiliary basis set.

threshP	$(R_1)_{max}$	$(R_2)_{max}$	$K_{\mathrm{Q}} - K_{\mathrm{pyscf}}$	$J_{\mathrm{Q}} - J_{\mathrm{pyscf}}$	$E_{\mathrm{ex,\ Q}} - E_{\mathrm{ex}}$	$E_{\mathrm{elec,\ Q}} - E_{\mathrm{elec}}$
10^{-1}	65.4	3.53×10^{-1}	4.53×10^{-4}	1.85×10^{-2}	4.08×10^{-4}	5.15×10^{-3}
10^{-2}	65.4	5.98×10^{-2}	4.60×10^{-4}	1.59×10^{-2}	6.58×10^{-4}	1.05×10^{-2}
10^{-3}	65.4	2.99×10^{-2}	4.26×10^{-4}	1.44×10^{-2}	5.28×10^{-5}	1.42×10^{-3}
10^{-4}	65.4	2.16×10^{-2}	3.96×10^{-4}	1.55×10^{-2}	5.70×10^{-4}	7.79×10^{-3}
10^{-5}	65.4	2.73×10^{-2}	3.73×10^{-4}	2.12×10^{-2}	1.09×10^{-4}	1.72×10^{-3}
10^{-6}	65.4	4.12×10^{-1}	8.13×10^{-4}	2.08×10^{-2}	5.42×10^{-4}	7.37×10^{-3}
10^{-7}	65.4	2.89×10^{0}	7.75×10^{-4}	2.08×10^{-2}	5.04×10^{-4}	7.13×10^{-3}
10^{-8}	65.4	7.68×10^{0}	3.97×10^{-4}	1.95×10^{-2}	9.59×10^{-6}	2.46×10^{-3}

Table B.7: Error Analysis Between JK-Engine with the metric matrix Q_{xy} and Density Fitting Object in PySCF for Various Threshold in the inversion of P_{xy} Matrix $threshP$ for Ethylene (C_2H_4): This table indicates the differences in computed values for exchange and Coulomb integrals, exchange energies, and electronic energies between the JK-Engine algorithm, and the Density Fitting approach in PySCF. These calculations were performed under the setting of $\alpha = 0.6$ with the cc-pVTZ basis set and the cc-pvtz-jkfit auxiliary basis set.

threshP	$(R_1)_{max}$	$(R_2)_{max}$	$K_{\mathrm{Q}} - K_{\mathrm{pyscf}}$	$J_{\mathrm{Q}} - J_{\mathrm{pyscf}}$	$E_{\mathrm{ex,\ Q}} - E_{\mathrm{ex}}$	$E_{\mathrm{elec,\ Q}} - E_{\mathrm{elec}}$
10^{-1}	147	3.63×10^{-1}	4.73×10^{-4}	1.60×10^{-2}	5.63×10^{-4}	6.76×10^{-3}
10^{-2}	147	6.38×10^{-2}	2.58×10^{-4}	1.22×10^{-2}	2.42×10^{-4}	2.76×10^{-3}
10^{-3}	147	1.97×10^{-2}	2.39×10^{-4}	1.30×10^{-2}	9.93×10^{-5}	2.70×10^{-3}
10^{-4}	147	1.19×10^{-2}	3.73×10^{-4}	1.33×10^{-2}	4.56×10^{-4}	6.00×10^{-3}
10^{-5}	147	1.65×10^{-2}	5.76×10^{-4}	1.57×10^{-2}	5.48×10^{-4}	7.45×10^{-3}
10^{-6}	147	2.91×10^{-1}	6.50×10^{-4}	1.77×10^{-2}	5.21×10^{-4}	3.30×10^{-3}
10^{-7}	147	6.73×10^{0}	1.10×10^{-3}	1.58×10^{-2}	7.42×10^{-4}	1.10×10^{-2}
10^{-8}	147	2.64×10^{1}	1.09×10^{-3}	1.47×10^{-2}	7.62×10^{-4}	1.02×10^{-2}

Table B.8: Error Analysis Between JK-Engine with the metric matrix Q_{xy} and Density Fitting Object in PySCF for Various Threshold in the inversion of P_{xy} Matrix $threshP$ for Ethane (C_2H_6): This table indicates the differences in computed values for exchange and Coulomb integrals, exchange energies, and electronic energies between the JK-Engine algorithm, and the Density Fitting approach in PySCF. These calculations were performed under the setting of $\alpha = 0.6$ with the cc-pVTZ basis set and the cc-pvtz-jkfit auxiliary basis set.