Understanding and integrating quantum chemistry byte by byte

Annabelle Lolinco

Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/etd

Recommended Citation
Lolinco, Annabelle, "Understanding and integrating quantum chemistry byte by byte" (2020). Graduate Theses and Dissertations. 17842.
https://lib.dr.iastate.edu/etd/17842

This Thesis is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
Understanding and integrating quantum chemistry byte by byte

by

Annabelle Tam-Ha Lolinco

A thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Physical Chemistry

Program of Study Committee:
Theresa L. Windus, Major Professor
Mark S. Gordon
Thomas A. Holme

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this thesis. The Graduate College will ensure this thesis is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University
Ames, Iowa
2020

Copyright © Annabelle Tam-Ha Lolinco, 2020. All rights reserved.
# TABLE OF CONTENTS

| LIST OF FIGURES                                                   | ................................................................. | iv  |
| LIST OF TABLES                                                   | ................................................................. | vi  |
| ACKNOWLEDGEMENTS                                                 | ................................................................. | vii |
| ABSTRACT                                                        | ................................................................. | ix  |
| CHAPTER 1. GENERAL INTRODUCTION                                  | ......................................................................... | 1  |
| Background                                                       | ......................................................................... | 3  |
| Interoperability                                                 | ......................................................................... | 3  |
| Computational Chemistry Considerations                           | ......................................................................... | 7  |
| Quantum Chemistry Theory                                         | ......................................................................... | 8  |
| Basis sets                                                       | ......................................................................... | 10 |
| Effective core potential                                         | ......................................................................... | 11 |
| Density functional theory                                        | ......................................................................... | 12 |
| References                                                       | ......................................................................... | 13 |
| CHAPTER 2. QUANTUM CHEMISTRY COMMON DRIVER AND DATABASE (QCDB):   | ......................................................................... | 17 |
| AUTOMATION AND INTEROPERABILITY AMONG MULTIPLE QUANTUM CHEMISTRY PROGRAMS | ................................................................ | 17 |
| Abstract                                                        | ......................................................................... | 18 |
| Introduction                                                     | ......................................................................... | 18 |
| Features & Design Philosophy                                     | ......................................................................... | 22 |
| QCSchema and the Quantum Chemistry Software Ecosystem            | ......................................................................... | 23 |
| Technical Aspects to Interoperability                            | ......................................................................... | 28 |
| Standard QC input                                                | ......................................................................... | 28 |
| Routine considerations                                           | ......................................................................... | 32 |
| Modes                                                           | ......................................................................... | 32 |
| Keywords                                                         | ......................................................................... | 34 |
| QCVariables                                                      | ......................................................................... | 38 |
| Example Application                                              | ......................................................................... | 39 |
| Diatomic Spectroscopic Constant Fitting                          | ......................................................................... | 39 |
| Summary and Conclusions                                          | ......................................................................... | 45 |
| Acknowledgements                                                 | ......................................................................... | 47 |
| Supporting Information                                           | ......................................................................... | 47 |
| Maintenance                                                      | ......................................................................... | 47 |
| QC Program Capabilities                                          | ......................................................................... | 48 |
| Procedure Capabilities                                           | ......................................................................... | 50 |
| References                                                       | ......................................................................... | 55 |
LIST OF FIGURES

Figure 2-1 A breakdown of the QCArchive ecosystem modules and the QCDB (left and right, respectively) and a summary of their overarching features. .......................... 21

Figure 2-2 Layout and access pattern between community quantum chemistry codes, QCEngine and QCDB. ................................................................. 24

Figure 2-3 Degrees of unifying access to quantum chemical calculations. ........................................ 25

Figure 2-4 An example of bad implementation of keywords using CFOUR ......................... 37

Figure 2-5 Values are relative to spectroscopic constants calculated with the E_base PES (See Table 2-1). ................................................................. 44

Figure 3-1 The hydroamination reaction using dimethylamine with the bis-amide Zr-complex (left) to form the tris-amide Zr complex ..................................................... 59

Figure 3-2 A general reaction coordinate for an exothermic reaction that points out the reactants (A), the transition state (B), the products (C), the activation energy (D), and the overall energy of the system (E). ..................................................... 60

Figure 3-3 The geometry of the transition state where the marked atoms (encircled in green) indicate the N of the dimethylamine and its hydrogen transferring to the benzylic carbon. The smaller gray atoms are C, blue are N, white are H, red O, and the single, larger black atom is Zr. ..................................................... 64

Figure 3-4 The hydroamination reaction progress (left to right) with the addition of the dimethylamine, where the Zr binds the N of the amine first. As the dimethylamine bonds to Zr, the oxazolide structure moves away, and the hydrogen off of the incoming dimethylamine moves towards the benzylic carbon. Finally, the tris-amide-Zr-complex is formed. The upper righthand structure is a depiction of the TS ..................................................... 65
Figure C-1 A breakdown of the current NWChem implementation for the QCDB and QCEngine functionalities for (a,b) energy and gradients and (b) Hessian, properties and relativistic effects. The colors indicate if the modules are redesigned from an earlier version of the QCDB (purple), newly added (teal), testing needed (teal with orange outline), or not fully implemented in some fashion (blue with orange outline). The top level of the figure shows the block heading that NWChem’s input structure has. Those that are a mix of colors at the top level may be a more generalized option, i.e. the Tensor Contraction Engine (TCE).
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Description of calculations performed in this demo and their associated QC programs.</td>
<td>42</td>
</tr>
<tr>
<td>2-2</td>
<td>Comparison between theory and experiment for bond lengths (Å) and spectroscopic constants (cm$^{-1}$) of three diatomic molecules. All terms correspond to the difference between a value and the case CCSD(T)/cc-pCV[Q5]Z calculation.</td>
<td>43</td>
</tr>
<tr>
<td>2-3</td>
<td>Confirmed working energy methods in QCDB. Alternate implementations listed in rows within a QC Program, available analytic methods listed with “Y” and default implementation for a given program, method, reference, and all-electron/frozen-core (ae/fc) choice listed with “D”.</td>
<td>51</td>
</tr>
<tr>
<td>3-1</td>
<td>Summary of the important energetic quantities for the hydroamination reaction with the bis-amide Zr complex (kcal/mol).</td>
<td>65</td>
</tr>
<tr>
<td>C-1</td>
<td>Additional theory and module options capable of running in QCDB for NWChem. The majority of the TCE is now available for use.</td>
<td>80</td>
</tr>
</tbody>
</table>
ACKNOWLEDGMENTS

First and foremost, I am forever grateful to the larger scientific community, friends, and family that continues to uplift and encourages me to pursue my curiosity and love for science in all mediums. As much as I have taken, I hope to give back just as much to show my gratitude and love to those who have sacrificed for me to be where I am.

I especially thank my major professor Theresa Windus for her positivity, encouragement, and wholehearted support throughout my time in the Windus group. I am indebted to her kindness and understanding, even when personal challenges introduced themselves through my tenure in the group. I also appreciate the support of my committee members, Professors Mark Gordon and Tom Holme as well.

Thank you to Lori Burns for her mentorship throughout the Quantum Chemistry Common Driver & Database (QCDB) project. I could not have learned as much without her guidance. I am grateful for her patience and words of wisdom as I dived into programming.

To the former and current members of the Windus group, thank you all for your levity and camaraderie. Times both in and out of the office have been more enjoyable with you all around! I want to recognize Ellie Fought, in particular, for enduring the pains of writing with me.

Beyond the sciences, I want to recognize my mentors, role models, peers, and all-around wonderful human beings who have impacted my creativity, critical thinking, and the intersectionality of all the facets of my identity. I take pride in knowing that the village I have managed to cultivate over the years is so expansive. I have learned and grown so much in my years, but I know I still have more to experience and am excited to bring as many of you along with me on the journey.
The work done on the QCDB was supported by the National Science Foundation SI2 grant #OCI-1047772.

This work was supported by the U.S. Department of Energy (DOE), Office of Science, Office of Basic Energy Sciences, Division of Chemical Sciences, Geosciences & Biosciences at the Ames Laboratory under contract no. DE-AC02-07CH11358. The Ames Lab is operated for the DOE by Iowa State University. The document number assigned to this thesis is IS-T 3270.
ABSTRACT

With the wealth of quantum chemistry software available, the computational molecular sciences community recognized the need for an open and extensible ecosystem of quantum chemistry for the modern scientific era. The Quantum Chemistry Common Driver and Database (QCDB) is one such application programming interface that addresses this need. QCDB introduces interoperability across multiple quantum chemistry software packages and implements best practices options. Through the work in this thesis and in tandem with the Molecular Sciences Software Institute (MolSSI) and their Quantum Chemistry Archive ecosystem (QCArchive), the QCDB has been able to integrate NWChem, among other programs, and many of its quantum mechanics options.

Non-innocent ligands are an important, understudied component in catalytic reactions. With the interest in developing transition metal-catalyzed reactions due to their natural abundance, sustainability, and cost-effectiveness, studying reaction progress with a non-innocent ligand provides an avenue of catalytic reactions that are highly active and versatile. Computational calculations were made for the hydroamination of a bis-amide Zr-complex to produce the tris-amide Zr-complex, including the transition state and binding energy of the dimethylamine.
CHAPTER 1. GENERAL INTRODUCTION

The power of computers in chemistry has improved the way chemistry is understood and studied. For example, the field of chemistry is now able to use computers to study fundamental properties of chemical systems and to design and inform more efficient and productive chemical syntheses and reactions. Computational chemistry provides a window into information that is otherwise difficult to obtain, such as transition states and reaction barriers. New capabilities, such as exascale computing, become tangible possibilities and will create an abundance of computational results. In turn, contemporary programming languages and formats have enabled scientists to create more powerful ways to approach a calculation or problem in methods that reduce computational costs and allow chemists to preserve chemical information via databases. With the quantity of data more readily available through modern computations, data analysis and interpretation are fundamental skills for computational chemists. As the chemical enterprise studies more complex systems, computers continue to be integral in providing the resources to explore grand challenges within the molecular sciences, such as catalyst design, and guide experiments.

In the era of modern science, leveraging foundational tools and information to build more accessible tools is important to the scientific community. For computational quantum chemistry, this can come in the form of increased communications between programs. The Quantum Chemistry Common Driver and Database (QCDB) aims to leverage modern programming languages and notation to easily use the unique features of a variety of quantum chemistry programs without the barrier of learning the lexicon of each program. The QCDB is an Python-based interface that is interoperable with several quantum chemistry programs, one of which is NWChem\(^1\), the focus of my work and is expounded upon in Chapter 2. QCDB simplifies the
complexities of the quantum chemistry programs’ unique features for ease of access to a larger user base. The single input generated through QCDB allows users access to multiple quantum chemistry programs, creating interoperability between the programs by leveraging both the unique features and commonalities shared in quantum chemistry programs. This cannot be done without a larger community effort, as developers and community experts of the varied quantum chemistry programs must coordinate and establish best practices in creating user-friendly syntax. Working with the Molecular Sciences Software Institute (MolSSI) and their Quantum Chemistry Archive (QCArchive) ecosystem, we have been able to hone and focus QCDB’s power as a driver. QCArchive’s structure as an ecosystem allows each module under its umbrella to have well-defined functionality that serves the larger ecosystem. Developers are now able to integrate more options from the quantum chemistry programs, and thus, more features, into QCDB.

Chapter 3 deviates from the topic of programming and communications to applied computations in chemical systems. Ligands that play an active role in catalytic reactions (aka, redox-active ligands or non-innocent ligands) can act as electron sinks, which have been of interest in recent literature.\textsuperscript{3-6} Complexes with these non-innocent ligands, created with readily available transition metals, can be characterized as highly active and versatile. Graduate student Yang Yun Chu and Professor Aaron Sadow of Iowa State University work experimentally on the reaction of CpOxZr, a mono-substituted cyclopentadiene with benzylic oxazolide structure, known as a bis-amide Zr-complex with dimethylamine (HNMe\textsubscript{2}) to produce a tris-amide Zr-product. This hydroamination reaction allows the release of the oxazolide ligand from the Zr center, formation of a double bond in the oxazoline between the N and C and protonates the benzylic carbon. A plausible reaction mechanism is determined via a search for a transition state calculated using NWChem alongside additional calculations to identify the dimethylamine
binding energy. Calculations suggest a potential transition state occurs when all the changes happen in a tandem fashion, through interchange substitution, though other possible mechanisms cannot be completely ruled out. Result are detailed in Chapter 3.

The conclusion discusses community efforts since the QCArchive has been made public\textsuperscript{7} for the broad computational molecular scientific community to submit additions or updates to the code. Concluding remarks are also made about how additional calculations with different metals and substituents for the hydroamination reaction that can improve our understanding about the reaction as the properties of transitional metal complexes can be influenced by surrounding ligands. Appendices A-C supplement Chapter 2 by providing step-by-step instructions to becoming a contributor and user of the interoperable quantum chemistry ecosystem, examples of the input structures for basic and advanced calculations, and details of the redesigned and new capabilities of QCDB. The rest of the introduction includes foundational information in interoperability and quantum chemistry that are used in this work.

**Background**

**Interoperability**

Interoperability, the capability of integrating multiple technologies to work together, is important for efficiently and easily interconnecting multiple systems.\textsuperscript{8} A system is defined here as a mechanism or product that functions with other mechanisms or products to complete a function. By providing more expansive capabilities, interoperability is a feature of systems with related or similar features that enhances the overall functionality available to a user. This builds a bridge for the components to work together in a larger context. One can think of several examples in everyday life such as purchasing a snack from the vending machine. If the purchase was made with a credit card, there needs to be an interface between the machine recognizing the credit card information with the bank to authorize the necessary amount. If the credit card is
stolen and the bank is informed, then the transaction is denied. There are also interoperable components to ensure that the purchase made is what was intended. If the selected option is empty, the machine will recognize and acknowledge that another option should be chosen. A more related context for this thesis is the ways we use applications (apps). Developers must account for the app to be used in a variety of different systems, i.e., smartphone users having an Android, Apple, Google, or other such devices. If one such application was a mobile-based game, said application must send and receive data between phone systems, display correctly on the various screen sizes, et cetera.

Computational molecular sciences research groups have a plethora of programs to use, given the long history of the field. Due to independent development, the myriad of software available contains features that are unique to each specific program. Unfortunately, the diversity of programs also introduces difficulty in the form of custom input and output. If a user were to switch between programs, it would require time to learn the input language specifications of the program(s). Thus, a common language, or schema of how information is presented, becomes the central goal. Sharing the same structures would allow users to set up an input that would be accessible by many programs. This community-developed schema would also give users an output file that collects key information from the programs. By having a common system for the input and output, the interoperable interface can leverage multiple programs’ unique features and provide a more complete picture of a molecular system.

There have been other efforts to use interoperability for better communications between a handful of high-performance quantum chemistry software. The Common Component Architecture (CCA) framework took a component-based approach to interoperability for NWChem, MPQC, and GAMESS codes. For example, by picking and choosing which
components of different scientific software would be beneficial for the overall goal, quantum chemistry researchers were able to put together a package that was a collection of high performance computational modules interfacing two central methodologies: quantum mechanics and molecular mechanics (QM/MM) across several different software packages.\textsuperscript{9,12} The interoperable function of a hybrid QM/MM method allows large systems to be divided into two components. Molecular mechanics can be embedded to treat large chemical systems, particularly the environmental effects, while the quantum mechanics calculations focus on the targeted site of the reaction.\textsuperscript{13} Once the structural framework design was decided on, scientists were able to develop interfaces, typically wrappers, that would then connect to ports with a specific purpose such as feeding or receiving information. The CCA framework was also able to put together unique portions of the codes to perform new calculations that would not be possible in a single quantum chemistry program, e.g., multiple relativistic effects for high accuracy computations.\textsuperscript{13} If different components can compute the same quantity, users are able to pick which code to use based on user preferences, such as completion speed for a given architecture, the choice of what architecture to run the calculation on, or a more accurate calculation.

A drawback to the system of interoperability employed by CCA is the limitation of its adaptability. The ‘plug-and-play’ architecture increased the knowledge base necessary for a user to work with the CCA framework, at least in setting up the different components. It also highlights the challenge of implementing the componentization of high-performance computing packages due to its reliance on a multitude of tools that are constantly being updated. Ensuring that a stable version of the architecture is in place requires recompiling the packages on a regular basis.
While the CCA was limited to the three high performance computing programs, the computational molecular science community was looking to expand the interoperability premise across any quantum chemistry software. With the modernization of one quantum chemistry program, PSI4, David Sherill and Lori Burns reached out to several other computational quantum chemistry developers to leverage the power of each of these programs.\textsuperscript{14} When branching out, they first began with the CFOUR\textsuperscript{15} program. In the next step, together with developers of GAMESS and NWChem, the Quantum Chemistry Database and Driver (QCDB) project was established. Earlier iterations of the QCDB involved mixing program-specific syntax to invoke interoperability across programs.\textsuperscript{16} It was also embedded within one specific program, PSI4, which limited the interoperable capabilities, restricting input syntax to be dependent on one specific program rather than be agnostic. The community of computational molecular scientists realized that much of the programming of the nuts and bolts (the basics) can benefit from standard best practices in presenting data, i.e., using formats like JavaScript Object Notation (JSON). This paved the way for the Molecular Sciences Software Institute to establish QCArchive\textsuperscript{17}, an ecosystem of several packages that allows quantum chemistry software to work together and creates a database of basic jobs, making computation time much faster as well as providing open and extensible data to the community at large. From its early inception, the QCDB has transformed into a wrapper that works with the larger QCArchive ecosystem, which will be discussed thoroughly in Chapter 2. The data accessibility available from these efforts are in part due to the capability of modern programming language and data forms. For the former, Python was the programming language of choice given its implementation in PSI4 and its lightweight and user-friendly capabilities. Core developers from a wide variety of programs now
provide insight on expanding this interoperable effort and aim to integrate syntax without asking the native quantum chemistry programs to relinquish their identity.

**Computational Chemistry Considerations**

In preparing for any computational chemistry calculation, there are several input considerations that must be made for the molecule or chemical reaction of interest. Three fundamental calculations computational chemists use in their arsenal to understand molecules are energy, gradient, and Hessian calculations. Energy calculations can provide insight on the kinetics and thermal conditions of a chemical system. Gradients, the first derivative of the energy with respect to the molecular coordinates, can be used to provide optimized geometries of molecular systems. Hessians are second derivative matrices that can be used to calculate frequencies describing the vibrations of a molecule. Chemists can use the Hessian to visualize infrared spectra and can indicate whether or not stationary points are local minima or maxima. The latter, associated with a negative (imaginary) frequency, suggests that the geometry may be a transition state structure.

As transition states cannot generally be experimentally observed or captured, computational chemistry is able to provide important information by determining potential structures via a saddle point search.\(^{18}\) An initial guess of the transition state must be a close estimate to the true transition state in order to successfully run a transition state search calculation. What constitutes as “close” can be determined by the Hessian matrix produced for the transition state, which must have a negative eigenvalue that corresponds with the reaction coordinate that the saddle point search climbs to find a maximum value corresponding to the transition state’s higher energy and its structure.

Other methods, such as those collectively known as chain-of-state methods\(^{19}\), can determine the approximate geometry of a transition state, given geometries of the reactant and
product. This allows the calculation to project and follow a potential reaction coordinate that can be developed between the minima to form the minimum energy path. From the estimated geometry through a chain-of-state method, one can run a saddle point search that is better directed. Chapter 3 will expand on the particular use of the nudged elastic band method for the work herein on a hydroamination reaction for catalysis.

**Quantum Chemistry Theory**

Quantum mechanics is key in understanding the molecular interactions and properties of chemical systems. Due to several underlying principles of quantum mechanics, chemists are able to relate the total energy of a chemical system to one equation (equation 1). In this equation, the state of a physical system, absent of a time-dependent potential, is defined by the wavefunction $\Psi$; however, the state is still dependent on its electronic ($r$) and nuclear ($R$) coordinates and thus is written as $\Psi(r, R)$.

Quantum mechanics also dictates that all physically measurable and discrete quantities, observables like momentum, are described by an operator. Thus, the total energy, $E$, of a chemical system can be stated simply as:

$$\hat{H}\Psi(r, R) = E\Psi(r, R)$$

This equation is known as the time-independent Schrödinger equation. The operator $\hat{H}$ (Hamiltonian) contains five terms,

$$\hat{H} = \hat{T}_e + \hat{T}_n + \hat{V}_{en} + \hat{V}_{ee} + \hat{V}_{nn}$$

where $\hat{T}_e$ and $\hat{T}_n$ correspond to the kinetic energy of the electrons and nuclei, respectively; $\hat{V}_{en}$, $\hat{V}_{ee}$, and $\hat{V}_{nn}$ are the potential energy operators describing the electron-nucleus attraction, electron-electron repulsion, and nuclear-nuclear repulsion interactions, respectively. The Born-Oppenheimer approximation assumes that nuclei are stationary with respect to the electron and
its relatively significant speed. This allows for $\hat{T}_n$, to be considered null and $\hat{V}_{nn}$ to be constant for a given nuclear configuration, thus, making the electronic Hamiltonian,

$$\hat{H} = \hat{H}_{\text{elec}} = \hat{T}_e + \hat{V}_{en} + \hat{V}_{ee}$$

(3)

and resulting in a Schrödinger equation focused on the electronic energy,

$$\hat{H}_{\text{elec}} \Psi = E_{\text{elec}} \Psi$$

(4)

The total energy is the sum of $E_{\text{elec}}$ and $E_{\text{nuc}}$; the latter being the energy for the nuclear-nuclear repulsions, which can be determined from

$$E_{\text{nuc}} = \hat{V}_{nn} = \sum_{A < B} \frac{Z_A Z_B}{r_{AB}}$$

(5)

where $Z_A$ and $Z_B$ refer to the nuclear charge on atom A and atom B, respectively, and $r_{AB}$ is the distance between nucleus A and nucleus B. The Schrödinger equation can be used to solve for the H atom exactly, but for many-body systems, an approximation method is needed. The Hartree method uses the product of one-electron functions as an approximate wavefunction over the N-body system,

$$\Psi = \phi_1(r_1)\phi_2(r_2)\phi_3(r_3)...\phi_N(r_N)$$

(6)

where each $\phi_i(r)$ contains a spatial and spin component. However, this product does not follow the Pauli Exclusion Principle as the wavefunction is not antisymmetric when electrons are exchanged. To invoke the antisymmetric property, the Hartree—Fock (HF) method builds up the wavefunction with the incorporation of a Slater determinant.

The HF method is a powerful approximation due to its ability to never calculate an energy lower than the exact energy. The variational principle creates an upper bound that restricts the approximation. In the HF method, each of the one-electron orbitals minimizes its energy in the average potential of the collective of electrons. The resulting operator for treating
the potential over the field of electrons is known as the Fock operator \( \hat{F} \) which accounts for the Coulombic and exchange interaction between electrons. The HF equation is

\[
\hat{F}\phi_i = \epsilon_i \phi_i
\]  

(7)

where eigenvalue \( \epsilon_i \) is the energy for the \( i \)th orbital. This process is self-consistent as information about the orbitals depends on \( \hat{F} \), which is what is being solved for through equation 7. Thus, the solution of \( \hat{F} \) using initial orbitals creates new orbitals that feed into creating a new \( \hat{F} \), and so on until there is no significant difference in the orbital energies and density.

**Basis sets**

Basis sets are comprised of atomic basis functions \( \chi_r \) that are mathematical expressions of molecular orbitals,\(^{18} \phi_i = \sum_l c_{r_i} \chi_r \). In practice, the more that is known about a basis set, the more informed a user’s decision can be in creating their computational calculation. A balance must be struck between an extensive, more descriptive basis set that would be more accurate and the computational costs of running a calculation. Thus, the goal of any user is to reduce the size of the basis set without compromising the accuracy of the calculation. This section details information around the types of basis sets.

Two types of atomic orbitals are foundational for basis sets commonly used in chemistry calculations\(^{21} \). The first, Slater-type orbitals (STOs), are known in the form of equation 8

\[
X_{nlm}(\zeta, r, \theta, \varphi) = N Y_{lm}^m(\theta, \varphi) r^{n-1} e^{-\zeta r}
\]  

(8)

where \( n, l, \) and \( m \) stand for quantum numbers, \( \zeta \) the orbital exponent, \( r \) the distance between the nucleus and an electron, \( N \) is the normalization constant, and \( Y_{lm}^m \) the spherical harmonic functions. These orbitals are a description of H-atom-like systems. Using STOs can solve computational difficulties. Boys\(^{22} \) addressed this issue by using Gaussian-type functions (GTFs). The GTFs, aka, primitive basis functions, (equation 9) then informed the second type of atomic
orbitals, Gaussian-type orbitals (GTOs), which are linear combinations of GTFs that are then used to form molecular orbitals.

\[ g_{n \ell m}(\zeta, r, \theta, \varphi) = N Y_{\ell m}^m(\theta, \varphi) r^{2n-2} e^{-\zeta r^2} \]  

(9)

The GTOs are easier to compute; however, the GTOs do not behave as well near or far from the nucleus of chemical systems.

Basis sets are then built from a combination of these atomic orbitals in basis functions. The minimal basis set, as its name suggests, allots one basis function for every atomic orbital within the system of interest. The number of basis functions per atomic orbital leads to basis sets, such as double-zeta, triple-zeta, quadruple-zeta, and so forth. Basis sets may also include polarization and diffuse functions that can better describe the chemical system. Polarization basis functions, typically denoted by a “p” or *, in a Dunning and Pople basis set, respectively, consider the effects of atomic orbitals with a particular directionality. Diffuse basis functions, aug or + (i.e., aug-cc-pVDZ in Dunning basis sets or 6-311++G in Pople basis sets), account for the chemical system requiring description further away from the nucleus, such as anions and molecules with lone pair electrons. Mathematically, there is an addition of exponents and coefficients to the wavefunction to account for this.

**Effective core potential**

The chemical system included in this thesis has zinc, a transition metal and a heavy atom. The need for an effective core potential (ECP)\textsuperscript{23} is two-fold. There are many electrons with heavy-atom systems. ECPs exchange the core orbitals and electrons with a potential, making the calculation easier and faster. In addition, the relativistic effects that occur on the geometry and energies of heavy atom molecules cannot be ignored. ECPs, in theory, do not sacrifice the accuracy of the system of interest since the potential that replaces the core electrons accounts for
at least some of the relativistic effects. LANL2DZ\textsuperscript{24} is a double-zeta valence basis set that is large enough to provide a general picture of the molecular system without requesting too much computational expense.

**Density functional theory**

Ab initio methods give a more complete picture of the system of interest; however, the number of variables to control leads to complex mathematics and more expensive computations. To address this issue, theoretical chemists and physicists use functionals, a function of a function, in computational methods. Density Functional Theory (DFT) relies on finding the electronic energy of a molecule by using the functional of the electron density, \( \rho \). To capture the ground-state molecular energy (\( E_0 \)), Hohenberg and Kohn’s theorem\textsuperscript{25} describes the Hamiltonian as a sum of the chemical system’s kinetic and potential energies:

\[
E_0 = T[\rho] + V_{en}[\rho] + V_{ee}[\rho]
\]  

where \( T[\rho] \) is the kinetic energy, \( V_{en}[\rho] \) is the electron-nuclear attraction, and \( V_{ee}[\rho] \) is the electron-electron repulsion. A key facet of equation 10 is its dependence on the density, a function in itself, thus creating a functional as described earlier. However, the connection between the energy and the electron density requires the exact solution of the functional to be known. Kohn and Sham\textsuperscript{26} further developed DFT, approximating the solution to the density by using a similar approach to the system that HF takes with the addition of the exchange-correlation contribution to the total energy.

\[
\rho = \rho_s = \sum_{i=1}^{n} |\theta_i^{KS}|^2
\]

Equation 11 is a Slater determinant of the Kohn-Sham spin orbitals \( \theta_i^{KS} \), which can be used to determine the exchange-correlation functional, \( E_{xc} \). By modeling the HF method, Kohn-Sham orbitals can use the self-consistent field procedure to optimize its orbitals for \( \rho \).
DFT has become a popular computational chemistry method due to its use of the electron density to approximate the energy of the system while being computationally cost-effective. To this day, there are a plethora of DFT functionals available for computations as there is no methodical system for improving upon functionals. Generally, though, the DFT functionals can be grouped into the following categories, depending on the exchange-correlation approximations used, in order of increasing sophistication: local-density approximation (LDA), local spin density approximation (LSDA), generalized-gradient approximation (GGA), meta-GGA, and hybrid.27

References


17. Lori Burns and Daniel Smith detail the broader picture of the QCArchive, QCDB, and the ability to compose and decompose quantum chemistry software at SciPy 2019 which can be viewed on YouTube: [https://www.youtube.com/watch?v=HBkY_qnYSQw&t=118s](https://www.youtube.com/watch?v=HBkY_qnYSQw&t=118s)


CHAPTER 2. QUANTUM CHEMISTRY COMMON DRIVER AND DATABASE (QCDB): AUTOMATION AND INTEROPERABILITY AMONG MULTIPLE QUANTUM CHEMISTRY PROGRAMS

Modified from a manuscript to be submitted to the Journal of Chemical Physics and modified with permission from collaborators.

The NWChem work for this thesis is the contribution of Annabelle Lolinco, which includes the redesign of the interface components once the QCDB became a program-independent application program interface. Additional quantum mechanics modules were added to bring the full Tensor Contraction Engine (TCE) suite, Multiconfiguration Self-Consistent Field, spin-orbit density functional theory, relativistic effects, and much of the property block from NWChem for interoperable access.

Daniel G. A. Smith¹, Zachary L. Glick², Annabelle T. Lolinco³, Jiyoung Lee³,a, Devin A. Matthews⁴,b, Nuwan De Silva³,c, Levi N. Naden¹, Matt Welborn¹, Doaa Altarawy¹, Sebastian Lee⁵, Fang Liu⁶, John F. Stanton⁷,d, Mark S. Gordon³, Theresa L. Windus¹,³, C. David Sherrill², and Lori A. Burns²

¹: Molecular Sciences Software Institute, Blacksburg, Virginia 241061 USA

²: Center for Computational Molecular Science and Technology, School of Chemistry and Biochemistry, and School of Computation Science and Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332-0400, USA

³: Department of Chemistry, Iowa State University, Ames, Iowa 50011, USA

⁴: The Institute for Computational Engineering and Sciences, The University of Texas at Austin, Austin, Texas 78712, USA

⁵: California Institute of Technology, Pasadena, CA 91125, USA

⁶: Massachusetts Institute of Technology, Cambridge, MA 02139, USA
Abstract

Community efforts in the computational molecular sciences are shifting to more modular, open, and interoperable interfaces that work with current programs, providing more functionalities between programs. The Quantum Chemistry Common Driver and Database is one such application programming interface that introduces interoperability across multiple quantum chemistry software packages and implements best practices options. In tandem with the Molecular Sciences Software Institute and their Quantum Chemistry Archive ecosystem, the QCDB has been able to integrate unique functionalities of several quantum chemistry programs, e.g., CFOUR, GAMESS, Psi4, and NWChem, as well as run common computational functions, i.e., energy, gradient, Hessian, properties. Power and native users benefit from adopting these application programming interfaces as they lower the language barrier of input styles, enable an easy layout of variables and data, provide end to end interoperable programming for complex calculations, and use best-practice options.

Introduction

The number of quantum chemistry programs is continuously increasing to build a rich spectrum of programs where any level of accuracy, performance, distributed computing, GPU enabled, or appropriately licensed program can be obtained. While this spectrum is generally beneficial from an end user point of view, the diversity of custom input and output makes it
difficult to switch between programs at will without needing to learn the vagaries of each - costing time and increasing the potential for error. In addition, single computations are often not as useful in the modern era as the ability to run complex workflows or analysis routines.

There are layers of expertise that users must master to use one or several quantum chemistry (QC) programs as research tools for the simplest/least divisible task. The user interface, like a traditional machine interface, has elements, or knobs, that users interact with. To that end, (I-a) they must know what model chemistry will treat the molecular system of interest with adequate physics in tractable time, a question of scientific expertise. (I-b) They must know any knobs to turn to enact modifications of that model chemistry, e.g. density-fitting, convergence, active space, also a question of scientific expertise. (I-c) They must know the names given by a QC program to the knobs that dial up that model chemistry and those modifications, a question of domain-specific language (DSL) expertise (here, “domain” is the software silo). (I-d) They would benefit from knowing the insider best-practice knobs that select the most efficient algorithms and approximations specialized to the model chemistry, a question of program expertise. (I-e) They must know the structure of the input deck by which the QC program receives instruction, a question of formatting and DSL expertise. Lastly on the input side, they must know the dance of files, environment variables, and commands to launch the job, a question of (I-f) program operation and (I-g) high-performance computing (HPC) expertise.

On the output/analysis side, further skills are required. The preceding sequence will generally provide users with an ASCII output file, some ancillary array output, and perhaps a program-specific structured output file. From there, (O-a) users must know what strings from the output deck mark the desired result, a matter of DSL expertise. If the targeted quantity is not explicitly printed but is derivable, (O-b) they must know what arithmetic can obtain it, a question
of QC expertise. If individual energies or derivatives are to be combined (O-c) for a more sophisticated model chemistry (e.g., basis set extrapolation\textsuperscript{1}, HEAT procedure\textsuperscript{2}, empirical correction\textsuperscript{3}), (O-d), for system decomposition or perturbation (e.g., counterpoise procedure, geometry optimization, finite difference derivatives), or (O-e) for general scripting, the users may be able to use routines inbuilt into QC programs in their vertical integration efforts thereby needing again DSL expertise. More commonly, they may want to combine the results with other programs.

Thus, the development of the Quantum Chemistry Common Driver and Database (QCDB) began within Psi4\textsuperscript{4} and expanded to include CFOUR\textsuperscript{5}, GAMESS\textsuperscript{6}, and NWChem\textsuperscript{7}. Atop of the QC programs, there were also specific procedures embedded within QC programs and available outside that could enhance interoperability. Our collaborative efforts aim to ease many user and scaling barriers and catalyze QC program interoperability while preserving workflow through multiple entry points, preparing for future extensibility and pressing for maximal code reuse. The Molecular Sciences Software Institute\textsuperscript{8} (MolSSI) also began developing a Quantum Chemistry Archive (QCArchive) ecosystem\textsuperscript{9} with the intent that the design would be open, extensible and modular, all a boon for the computational molecular sciences community. Together, as shown in Figure 2-1, the QCDB and QCArchive are able to enhance the computational quantum chemistry user experience. An interoperable workflow requires the QCDB and QCEngine\textsuperscript{10} modules, both of which rely on QCElemental\textsuperscript{11} for physical constant and periodic table information and follow QCSchema\textsuperscript{12}, the community adopted JSON structure for output variables.
In addition to the array of software platforms, the substantial variability of user capabilities is another attribute of the QCDB and QCEngine APIs that ease the interoperability barrier among the program and procedures available. Consider a new QC practitioner figuring out which program’s B3LYP suits their hard hardware, a force field builder collecting thousands to millions of SAPT results, and a spectroscopist modeling a molecule with a composite method. Each have different needs as to input, operational, and results uniformity. The first would benefit from being able to try several installed programs without absorbing the manuals of each to create comparable inputs. The second would benefit from high-throughput input generation,
submission, and summary and detailed results querying. The third would benefit from simple syntax for composite methods with the power to call multiple compute engines without scripting. Neither QCEngine nor QCDB ever has the ambition to choose one QC program over another for a given method.

Features & Design Philosophy

The design philosophy of QCDB, and by extension, QCEngine, was meant for both native QC program and new users and provide additional functionalities beyond any one single QC program using the resources available. Python, as a modern programming language, has its advantages in providing adaptable structures in translating QC codes that would be interoperable across various QC programs. Figure 2-2 shows the various QC codes that are integrated into QCDB and QCEngine. QCDB has two types of client engines, programs and procedures. Programs are executables or libraries of any ilk, language, or interface that can produce total or partial energy, gradient, Hessian, and properties (E/G/H/P) calculations. These are always external and usually called through QCEngine. Procedures involve many, mixed, or multistage E/G/H/P pieces. These are things like optimizers, complete basis set (CBS) procedures, diatomic analysis, and may be internal or external to a program. The modules create ProgramHarnesses and ProcedureHarnesses that wrap around the available QC code and provide interoperability between programs through QCDB. Figure 2-3 showcases the differences between the APIs as QCEngine provides unified molecule, method, and execution process, whereas QCDB can leverage best practices options, interoperability among QC programs and procedures, and unifying keywords. Thus, having both are key to truly providing end to end interoperable programming.
Whenever a quantum chemistry mechanism takes in QC-program-agnostic analytic E/G/H/P (i.e., AtomicResults) but requires multiple ones (e.g., a finite difference derivative), needs additional software (e.g., effective fragment potentials (EFP) or SALC coordinates), needs to take action in multiple stages (e.g., a geometry optimizer), or could combine AtomicResults from different programs (e.g., a composite method), it is classified in QC Engine as a procedure (see upper left edge of Figure 2-2) and is implemented in a Procedure Harness to promote modularity and broaden its input scope. Because procedures act upon generalized quantities, any code interfaced with QCDB gets all of the procedures below “for free”. The following information details the QCArchive ecosystem development and the technical aspects of developing QCDB’s interoperability. Further details about the program and procedure capabilities can be found in the supporting information.

QCSchema and the Quantum Chemistry Software Ecosystem

To develop a standard information exchange format for QC programs, support just one interface for the driver, and encourage all QC packages to adopt is a difficult series of goals for a single research group, or even a handful of research groups, to successfully advocate to a broad developer community. However, MolSSI, funded by the U.S. National Science Foundation, provides a unique opportunity to sponsor community discussion and to advocate for standards. Members of our collaborative team and the codes represented have worked closely with MolSSI on their development of a QCSchema for quantum chemistry information exchange, which has been adopted for QCDB. Indeed, some of the code originally developed as a part of the QCDB interfacing project has been donated to MolSSI’s QCElemental and QC Engine modules from the MolSSI QCArchive infrastructure, the software stack supporting the QCArchive database project.9
Community codes (i) in a variety of languages are wrapped in QCschema input/output by a QCEngine harnesses (ii, iv), which may be light (if the code has an API or structured output) or heavy (if only text output available). The QCDB harnesses (iii, v) add unifying and ease-of-use layers atop the QCEngine calls. Whereas analytic energies and derivatives are classified as “programs” (ii, iii) and call QC codes directly, multi-stage and post-processing jobs are written as “procedures” (iv, v) for compositibility and distributability and call programs in turn. The QCDB driver provides API access to both sets. The (a), (b), (c, d) labels correspond with the stages of unified input in Figure 2-3.
Figure 2-3 Degrees of unifying access to quantum chemical calculations. Black text or grey shading are aspects not requiring DSL knowledge by user. The “execution” column uses the one-call Python API, as this is closest between QCEngine and QCDB, but other modes are available for each.

QCEelemental provides data and utilities usable by all QC packages. For data, it exposes NIST physical constants and periodic table information through a thin API and provides internally consistent unit conversion aided by the external module pint. QCEelemental is able to select
different dataset versions and property versions such as covalent and van der Waals radii. Additionally, QCEElemental encodes the MolSSI QCSchema for a QC or EFP Molecule, a QC job input morsel using the QCSchema AtomicInput or a QC job output record, AtomicResult. In addition to the specification of key/value data layout inherent to a schema, QCEElemental encodes validation and serialization routines for these objects through the external module Pydantic\(^1\), which collocates data layout and physics validation and schema definitions. Molecule parsing, manipulation, alignment, and output formatting routines are also attached to QCEElemental. Historically, many QCEElemental capabilities were developed for QCDB in Psi4\(^4\), then refactored into QCEElemental for broader community accessibility free from Psi4 and compiled-language dependence. QCDB uses all the QCEElemental capabilities mentioned, particularly for uniform treatment of fragmented, ghosted, and mixed-basis molecules across differing QC program features.

**QCEngine** provides a uniform execution interface for QC packages by consuming QCSchema AtomicInputs and emitting AtomicResults. Depending on the degree of programmatic access a QC package makes available, an interface of ProgramHarness may be simple as for a package that already speaks QCSchema like Psi4; moderate as for a package that supports a Python API like RDKit\(^1\) or Open MM\(^5\); or involved, as for an executable with ASCII I/O. Historically, QCArchive has focused on programs with native QCSchema or serialized output, be it binary, XML, JSON, or Python API, while QCDB has tackled raw ASCII output parsing. A QCDB ProgramHarness consists of taking an AtomicInput, translating the AtomicInput into input file(s) and execution conditions, executing it, collecting any useful output, parsing that in an AtomicResult and returning it. QCEngine additionally collects runtime data such as elapsed time, architecture of host, memory consumption of job, and
software environment details. By combining efforts, QCArchive and QCDB each gain access to more QC packages and share the maintenance and development burden.

The greater task of creating a common driver that can execute multiple QC programs from a single API or schema while presenting a uniform interface for model chemistry (e.g., CCSD/qz2p), HPC configuration (e.g., scratch, OpenMP), output collection and digitalization, and post-processing routines have been separated into different layers of concern. One principle is to harvest from existing QC programs their analytic QC capabilities and specialized drivable routines, i.e., vpt2 or makefp. QCDB is one of a few featured “endpoints” or “entry points” in the QCArchive ecosystem that users interact with directly. In that role, it has a Python API, JSON and YAML QCSchema modes, and an input parser that has migrated from Psi4. Structurally, QCDB contains 1) keyword machinery to handle defaults and choices by the user, the engine programmers (best-practice options), and the driver; 2) driver definitions for E/G/H/P, plus routing to finite difference if analytic implementations are not available; 3) post-processing procedures that comprise of many, mixed, or multistage energy, gradient, Hessian pieces; 4) QCVariable machinery to assign the harvesting and multiply the definitions; 5) QCEngine wrappers.

Some general principles are that the molecule must always be in a QCElemental-readable format that includes Cartesian, Z-matrix, and EFP formats, with the former two supporting variable and deferred coordinates. Basis set specification by default uses the Psi4 basis set library which is amply stocked with Pople, Dunning, Karlsruhe, and other orbital and fitting basis sets. Future work will move this to the MolSSI Basis Set Exchange\textsuperscript{16} (BSE). These are formatted for the engine and fed as user-specified basis sets. Alternately, the user can specify the program’s native basis set library. The general course of a job is to turn all components of a
predictable grammar into keywords, which can be resolved by the options resolver that juggles competing suggestions and requirements by the user, driver, QCSchema, and best-practice options. Once resolved, options take on a possibly nested key-value intermediate representation that can be formatted according to the program’s grammar into input file(s). QCDB takes some care toward extracting the most precise values from files, taking more decimal places over fewer and preferring binary values over strings. QCDB returns quantities in input orientation rather than QC program internal representation. In scope, it aims to ease the 99% while allow the 1% while neglecting the 0.01%.

Thus, the motivation behind QCDB, a Python module that presents a uniform interface to multiple QC codes. To accomplish the uniform interface, as a first step, we lifted the driver out of Psi4 and made it program-agnostic. Then, hooked up codes that can compute energies, gradients, etc. Each goes through a well-defined interface that keeps operation sandboxed and ready for distributed computing.

**Technical Aspects to Interoperability**

**Standard QC input**

Molecule specification is the most important aspect that QCEngine and QCDB control to the exclusion of a program’s DSL. The QCSchema Molecule can store mass, isotope, charge/multiplicity, fragmentation, ghostedness, connectivity information, and more along with the basic element and Cartesian geometry data. All quantities are stored in atomic mass units (amu) or Bohr to avoid imprecision from multiple unit conversions through different physical datasets.

Initializing a molecule can occur through a variety of string formats (of Cartesians) or directly by arrays. These formats are then extensively validated and converted to schema. In
QCDB, molecules can additionally be specified via Z-matrix or mixed Cartesian/Z-matrix.

QCSchema Molecules holds most data relevant to molecular system specification in QC, including EFP fragments, which are parsed without additional software through QCDB and are stored in a secondary object. Items which appear in some programs’ molecule sections but do not fit in QCSchema Molecule, such as the stars signaling optimizable internal coordinates in CFOUR.

A requirement to combining vector data from multiple jobs is that it be in a common frame of reference. Though each QC program has a standard internal orientation, these can be different between programs or between input geometries, and not all programs can return quantities in an arbitrary input frame and atom ordering. To smooth over inconsistent capabilities, the input and output geometries are collected from output data and an aligner computes the displacement, rotation matrix, and atom mapping needed to transform between them. Then all vector results have the appropriate transformations applied so that all results in AtomicResult are in the input orientation. A user can control this reorientation. When it is turned off, QCEngine returns the vector in program native orientation and QCDB returns the vector in Psi4 native frame.

Notwithstanding the curation efforts of the BSE, each QC program maintains an internal library of basis sets with basis set developer updates applied, uneven program owner specializations applied, and different spellings for accessing a given basis, not to mention different data formats. Also, syntax to specify custom or per-atom basis sets varies greatly between programs. In QCEngine, only programs’ internal libraries are used, accessed from the QCSchema.model.basis field. Thus, the same string value directed toward different programs can lead to different results, and different strings can lead to the same results, due to DSL. To
allow consistency between programs and to reduce user DSL demands, QCDB pulls basis sets from a single library (Psi4’s in .gbs format) and performs the translation into each program’s native specification and format, including setting spherical or Cartesian for d-shells and higher according to basis set design. In this way, a standard case-insensitive label and a consistent interface to custom and mixed basis sets is available. Alternatively, QCDB can act like QCEngine to access a program’s internal basis set library through program-specific keywords (e.g., set gamess basis gbasis accd vs. set basis aug-cc-pvdz). Once the user requests the QCDB library through set basis, overriding the default spherical/Cartesian setting must be done through set puream True (as opposed to local keywords like set cfour spherical). While Psi4’s basis set library is used at present, future work will switch to interfacing with the new MolSSI BSE project.

Perhaps the most powerful/compelling element of QCSchema is the ability to request methods by a single string rather than piecemeal (e.g., energy("bp86-d2") in place of QChem’s 17,18). As far as possible, all method information and no extraneous information is concentrated into the QCSchema.model.method field. This is the primary translation effort of each QCEngine ProgramHarness, as shown by the uniformity of the method field in Fig. 3(b). In calling QCEngine, the user supplies the canonical method name, including any program specifics. If two programs have made a different choice of VWN3 vs. VWN5 for B3LYP, then the same model field submitted to each will return different answers. This is consistent with the principle that users can translate an input file directly into QCSchema.

A complication to this principle is when programs conflate non-method information like algorithm (rimp2) or alternate code paths (task tce energy) into the primary method call. To maintain QCSchema integrity, the project invents top-level keywords like "qc module" ="tce"
to allow deliberate choice of the Tensor Contraction Engine\textsuperscript{19} (TCE) over the hand-coded coupled-cluster in NWChem or \texttt{"mp2 type" = "df"} to instruct density-fitting in Q-Chem. Note that unless disabled, TCE will be accessed by default for NWChem when hand-coded modules are not available, for example when running ROHF CCSD. The \texttt{qc module} keyword can also control the choice of VCC/ECC/NCC in CFOUR and DFMP2/DFOCC/CC/DETCI in Psi4, though these also have local knobs through the cfour cc module and psi4 qc module, respectively. In the case of NWChem, there is an additional complication that the method to be run (e.g., CI, MBPT and CC) is declared in a TCE input block. The \texttt{QCSchema} method \texttt{model.method="ccsd" or energy("ccsd")} is set using the TCE block information.

Method specification in QCDB is similar to QCEngine except that a compound program-method argument like \texttt{optimize("nwc-ccsd(t)")} is used. This difference from QCEngine is historical and endures because of the simplification it lends to composite model chemistries, e.g., \texttt{gradient("p4-mp2/cc-pv[56]Z + d: nwc-ccsd/cc-pv[tq]Z + d: c4-ccsd[tq]/cc-pvdz")}. The previous command requests a Dunning 5\(\zeta\) to 6\(\zeta\) Helgaker-formula extrapolation of the MP2 correlation gradient performed by Psi4 with a coupled-cluster singles-doubles excitations correction (CCSD MP2) at Dunning triple-\(\zeta\) quadruple-\(\zeta\) Helgaker-formula extrapolation gradient performed by NWChem with a coupled-cluster up to quadruples excitations at cc-pVDZ performed by CFOUR, all atop an implicit 6-\(\zeta\) Hartree--Fock.

Additionally, QCDB tests the major methods to ensure the same string yields the same result between QC programs. User specification of method information in keywords instead of through model is overwritten without warning in QCEngine, while in QCDB, contradictory information yields an error.
Routine considerations

Developing the architecture of the QCArchive ecosystem and QCDB also meant taking into consideration the ways QC programs handle memory, disk, and parallelism functions. User specification of memory resources, working directory and execution environment are managed by QCEngine and is outside the QCSchema. By default, the entirety of the target compute node’s memory is given to the job. If user-specified, input units are flexible, e.g.,

qcdb.compute(...) or qcmg.compute(..., local config = {“memory”: “10gb”}). In either case, the memory quantity is translated into DSL keyword names like MEMORY SIZE and MEM UNIT for CFOUR. Memory keywords placed directly in a QCSchema are ignored and overwritten in QC Engine or raises an error if conflicting in QCDB. An exception is cases like NWChem\textsuperscript{6} where aggregated memory is managed by QC Engine but distribution between heap, stack, and global are editable through keywords (e.g., MEMORY TOTAL or MEMORY STACK). The execution flags or environment variable that superintend QC program parallelism, as well as their single- or multi-node capabilities, are built into their respective QC Engine ProgramHarnesses. By default, a job gets the full compute resources (nodes and/or cores) assigned it. Each job is run in a quarantined scratch directory created for it and populated by input and any auxiliary files. Execution occurs through Python subprocess (or less often through Python API). Output files such as stdout, stderr, and any program-specific files in text or binary format are collected and returned in QCSchema fields before scratch directory deletion.

Modes

Providing a few distinct modes of operation are an effort to tailor QCDB’s capabilities for differing levels of expertise with individual QC programs and for differing needs for driver capabilities (interface to single-program vs. integration of multiple programs), while not
imposing a workflow. The most controlling is the unified mode which endeavors to elicit identical results out of identical input conditions, being roughly the combination of method, basis, driver, reference, and active space. This mode is required for multi-program procedure runs (e.g., `energy("p4-mp2/cc-pv[tq]z + d:c4-ccsd/cc-pvtz")`) and is recommended in general. In this mode, QCDB-level defaults are imposed by the driver, such as non-density-fitting algorithms, non-frozen-core spaces, tighter convergence criteria for gradients vs energies, or for finite difference vs analytic derivatives. Best-practice defaults are also present.

Another mode, denoted “sandwich” is for users focusing on a single QC program who want the driver routines and method mapping (e.g., `gradient("gms-ccsd")` or `energy("gms-b3lyp", bsse type="vmfc")`) and I/O wrapping advantages of QCDB but do not want surprise resets of their accustomed defaults. QCDB- and driver-level, and best-practices defaults are all turned off. This mode is effectively how QCEngine runs.

Finally, “expert” mode is for when users only want I/O wrapping and molecule specification from the QCDB infrastructure. In this case, the set of native input keywords, including those that set method and derivative level, are reformatted directly to the QCDB spelling, and the call is made to, for example, `energy("cfour")`. This mode is discouraged, but it is useful for shortcutting processing between the user and the QC program.

To illustrate the modes, in the absence of any additional user keywords, some background facts are needed.

1. Psi4’s default MP2 algorithm is density-fitted, while CFOUR, GAMESS, NWChem, and QCDB’s is converged canonical HF.

2. CFOUR, GAMESS, NWChem, Psi4, and QCDB’s default HF energy convergence is $10^{-7}$, $10^{-5}$, $10^{-4}$, $10^{-6}$, and $10^{-6}$, respectively. It should be noted that the NWChem
tolerance is for the norm of the orbital gradient and not on the energy differences.

The energy converges to approximately the square of the norm.

3. CFOUR default CC mode is VCC for CCSD energy, while QCDB best-practices for CCSD energy in CFOUR is ECC.

4. NWChem default task ccsd energy does not run for open-shell, while QCDB uses the CCSD module for RHF and the TCE module for ROHF.

5. GAMESS freezes core by default while CFOUR, NWChem, Psi4, and QCDB run all-electron.

In unified mode, `energy(“gms-mp2”)` and `energy(“p4-mp2”)` will both run all-electron MP2 without density-fitting and with convergence to $10^{-6}$. After setting ROHF, `energy(“c4-ccsd”)` will run through ECC, while `energy(“nwc-ccsd”)` will run through TCE, again both all-electron, with HF to $10^{-6}$. In sandwich mode, `energy(“gms-mp2”)` produces a conventional frozen-core MP2 energy converged to $10^{-5}$, while `energy(“p4-mp2”)` produces a density-fitted all-electron value converged to $10^{-6}$. In the ROHF CCSD case, the CFOUR job will run as all-electron through VCC with HF converged to $10^{-7}$, while an NWChem submission will decline to run.

**Keywords**

QC programs have hundreds of keywords controlling their operation. The variety in spelling and text arrangement by which the same ideas are communicated to different QC programs is staggering and a considerable barrier to trying new codes. The necessity to represent any single program input file as a QCSchema `AtomicInput` requires mapping rules to the key/value representation of the keywords field such that a user familiar with the DSL can mindlessly perform the translation. The primary guideline is that the right-hand-side value must
be a simple data quantity (though not necessarily a fixed type) in natural Python syntax (e.g., CFOUR’s 3-1-1-0/2-0-1-0 becomes \( \{ [3, 1, 1, 0], [2, 0, 1, 0] \} \)) and the left-hand-side key is a string that encodes any level of nesting with double underscore (e.g., GAMESS’s contrl__scftyp or Orca’s casscf__rel__gtensor), this nesting structure accounts for options that may inherently have a single underscore already (i.e., NWChem’s CCSD(2)_T option in the Tensor Contraction Engine) as QCDB or QCEngine processes the output. A present/absent keyword (as opposed to a key/value pair) becomes a boolean, like NWChem’s scf__rohf. Any unnecessary case-sensitivity on the part of a QC program is handled by the ProgramHarness (e.g., q22p becomes lowercase for CFOUR, while a filename option passes unchanged). The greatest challenge to this mapping is some programs’ unstructured parsing that blurs module nesting vs. keyword name vs. keyword value. An example is freeze in NWChem’s modules, like mp2. Here, freeze 1, freeze core 1, freeze atomic, and freeze atomic O 1 have the same result for an oxygen atom. But freeze=1, freeze_core=1, freeze=atomic, freeze_atomic=”O”: 1 are not independent options.

There are three guiding principles as developers we are following:

- A user familiar with native QC program’s input deck and the schematizing principles should be able to write out the QCSchema keywords section (accordingly set_options in QCDB).

- The keywords section shall be Dict[str, Union[bool, str, int, float, List, Tuple, Dict[str, Any]]]. Any module hierarchy shall be represented by double underscore and any QC program specification is by prefixing the program with an underscore. For example, calling for an ROHF run in NWChem would be “nwchem_scf__rohf” : True.
• Options should be independent and granular such that they’re 1:1 with other programs, not 5:1, that is no single grid option that covers lebedev number, pruning, and quadrature all wrapped together.

Thus, NWChem’s input file structure implementation into QCSchema gave developers a unique challenge of determining what would be intuitive and logical due to the tiered block structure, where a given module and keyword can have values of arrays, strings, and numerical data that span several topics and can be unwieldly to process. The usual module-level keyword pattern would hold: to tighten convergence for hand-coded CC, set “nwchem_ccsd__thresh”; for TCE CC set “nwchem_tce__thresh”; or agnostically set “e_convergence”.

The input below (Figure 2-4) is set up to fail in four ways: contradictory specification of memory, multiplicity, computation method, and derivative level. Note, though, that the CFOUR units angstrom setting is permissible, since it concurs with the value implied in the molecule block.

As all communication happens through QCSchema, any program input file must be expressible in it. Though a nested key/value structure may seem sufficiently flexible for QC program keywords, there are quirks and ambiguities. Foremost to resist is the temptation to
memory 300 mb
molecule{
H
H 1 0.7
}

set basis 6-31g
set cfour_multiplicity 3 #clash with implicit singlet in #molecule{}corn above
set cfour_units angstrom #no problem
set cfour_memory size 1000000 #clash with 300 mb above
set cfour_calc_level ccsd #clash with ‘c4-scf’ below
set cfour_deriv_level first #clash with energy() below
energy(‘c4-scf’)

Figure 2-4 An example of bad implementation of keywords using CFOUR

rationalize a program’s keywords syntax – the abiding principle is that longstanding users of the program should be able to translate without aides a DSL input file into QCSchema. Some complications are alternate paths through the code that were once expressed with the method. As discussed above, we invent a top-level keyword qc_module to allow deliberate choice of the Tensor Contraction Engine (tce) or hand-coded coupled-cluster (cc) NWChem. For QCDB, when you want to target an option toward a particular program, it can be prefixed by the program; hence, charge becomes nwchem_charge or cfour_charge.

Barriers to users using multiple QC backends or to QCSchema producing uniform output when fed to different programs are (1) heterogeneous control across QC programs as each as its own knobs and keywords and (2) incompatible results as different defaults mean slightly different answers across programs defeating mixing programs. These issues are answered by (1) keyword translations which allows users to focus on scientific choices not DSL, (2) best-practice keywords that allow shorter inputs, quicker calcs, and bridge the dev/user knobs gap, and (3) lower barriers to using a variety of codes, many with unique features.
**QCVariables**

The QC output stream, whether ASCII or binary, is read immediately after program execution in the parse output function of the ProgramHarness. Scalar and array result quantities, such as PBE TOTAL ENERGY, MP4 CORRELATION ENERGY, (T) CORRECTION ENERGY, PBE TOTAL GRADIENT, and CCSD DIPOLE are extracted and held as significant figure preserving floats or NumPy arrays, respectively, and known as QCVariables. Extraction uses the most precise available source, whether the standard output stream or one of the auxiliary files generated by the end of a computation. Results such as CFOUR’s GRID or NWChem’s movecs are available programmatically through the ProgramHarnesses and will be kept as a separate output file. The computation’s internal geometry is always collected, and any vector results are manipulated in concert with it as described in the Keywords section. In QCEngine, programs are taken at their word for appropriately labeling harvested quantities that are collated in QCSchema AtomicProperties lists.

A vexation for readers of QC output files is that they will contain different quantities like total vs. correlation energy or opposite-spin vs. triplet energy that are interconvertible but not directly comparable. QCVariables (qcvars) in QCDB enforce the definition and consistency of common QC definitions and encode common combining rules. They are applied in post-processing and additionally help identify parsing and parsing programming errors. More importantly, this ensure that a maximum of data gets harvested from each run, that exactly the same quantities are collected from each QC program, and that trivially defined methods such as SCS(N)-MP2 and B3LYP-D3(BJ) do not clutter either the QC code or its parsing. Extra tags are attached to many qcvars to track combinability.
Using binary representations of floats rather than truncated strings from output files is a powerful argument for API integration rather than parsing. This is particularly important when dealing with many numbers with slight differences, such as the case with finite difference or many-body expansion sums. This is already clean in Psi4 since results are transferred in full precision from the C++ layer to the Python API and hence to QCEngine. For CFOUR, QCDB stores a list of keys and their types from the JAINDX, and uses this to read the binary JOBARC file for certain QC results (e.g., energy) and organizational data (e.g., atom reordering index). Ordinary analytic method runs in CFOUR generally have their results collected by parsing the text output file. Future work may switch this to JOBARC reading instead for greater precision.

**Example Application**

**Diatomic Spectroscopic Constant Fitting**

With contemporary quantum chemistry (QC) software, it is entirely possible to approach the *ab initio* limit in the description of molecules with up to two heavy atoms. Such spectroscopically accurate calculations require extrapolating to the full configuration interaction and complete basis set limits under the relativistic Born-Oppenheimer (BO) approximation, followed by usually negligible corrections to account for both relativistic effects and the BO approximation itself. Not only does this type of calculation present a remarkable computational challenge, it can also be practically difficult to incorporate multiple corrections and extrapolations into a project’s workflow. While all of the necessary features are present across various QC software packages, no single package exclusively implements everything (let alone has the best implementation). Furthermore, enforcing consistent geometries, basis sets, convergence criteria, frozen orbitals, etc. between programs is a cumbersome, often error-prone task. The QCDB driver remedies this problem by providing an easy-to-use Python interface to multiple QC programs.
To showcase this capability of the QCDB driver, the ground states of a few diatomic molecules (BH, HF, and C₂) are optimized at essentially the ab initio limit, and spectroscopic constants are computed and compared to experiment. We include corrections for electron correlation beyond CCSD(T), basis set effects beyond an already high-quality core-valence quadruple/quintuple-zeta extrapolation, relativistic effects, and the Born-Oppenheimer diagonal correction, using four different QC programs through the unified QCDB interface. We examine the effect of each correction separately as well as the cumulative effect of all corrections. Understanding of the cost/usefulness of each correction is helpful for designing reasonable extrapolations for larger systems.

A spectroscopically accurate model chemistry energy (E_{Total}) is defined as a base energy (E_{Base}) with five separate corrections:

\[ E_{Total} = E_{Base} + \Delta E_{\text{Basis}} + \Delta E_{\text{DBOC}} + \Delta E_{\text{Relativistic}} + \Delta E_{\text{CCSDTQ}} + \Delta E_{\text{FCI}} \]  

Each energy and the QC program(s) used to obtain it is defined in Table 2.2. The rovibrational spectrum of a diatomic molecule is often expressed with Dunham’s expansion:

\[ E_{\nu J} = \hbar \sum_{kl} Y_{kl} \left( \nu + \frac{1}{2} \right)^k \left[ J(J + 1) \right]^l \]  

The first few Dunham coefficients correspond to well-studied spectroscopic constants:

\[ Y_{10} = \omega_e, \ Y_{20} = -\omega_e x_e, \ Y_{01} = B_e, \ Y_{02} = -\bar{D}_e, \ Y_{11} = -\alpha_e \]  

The following truncation of the expansion was used to describe a diatomic:

\[ E \approx U(r_e) + \hbar \omega_e \left( \nu + \frac{1}{2} \right) + \hbar B_e J(J + 1) - \hbar \omega_e x_e \left( \nu + \frac{1}{2} \right)^2 - \hbar \alpha_e \left( \nu + \frac{1}{2} \right) J(J + 1) - \bar{D}_e J^2 (J + 1)^2 \]  

The spectroscopic constants are then describable in terms of the electronic PES U(r_e) and its derivatives:
\[ I_e \equiv \mu r_e^2 \quad B_e \equiv \frac{\hbar}{8\pi^2 I_e} \quad \omega_e \equiv \frac{1}{2\pi} \left[ \frac{U''(r_e)}{\mu} \right]^{1/2} \]  
\[ (9) \]

\[ \omega_e x_e \equiv \frac{B_e^2 r_e^2}{4\hbar \omega_e^2} \left[ \frac{10B_e r_e^2 [U'''(r_e)]^2}{3\hbar \omega_e^2} - U''(r_e) \right] \]  
\[ (10) \]

\[ \alpha_e \equiv \frac{2B_e^2}{\omega_e} \left[ \frac{2B_e r_e^2 [U'''(r_e)]^2}{\hbar \omega_e^2} + 3 \right] \quad \overline{D}_e \equiv \frac{4B_e^3}{\omega_e^2} \]  
\[ (11) \]

Accessed through the QCDB interface, Psi4’s diatomic procedure fits a set of points \((r, E(r))\) to this truncation, solving for the spectroscopic constants via a least-squares optimization. This procedure was used in the following way for each diatomic:

1. Through the QCDB driver, \(E_{\text{Total}}\) was calculated at 7 values of \(r\), spaced 0.005 Å apart and centered approximately at the minimum of the PES. The spectroscopic constants were calculated with Psi4, including an approximate \(r_e\).

2. This 7-point calculation was repeated, using the approximate \(r_e\) from the first step as the central point. The spectroscopic constants calculated from these PES points are those tabulated here.

Basis sets with spherical harmonics were used in all calculations, and basis set coefficients were standardized across all programs via QCDB. Electrons in core orbitals were frozen for computations using the cc-pVXZ basis sets, which lack core correlation functions. Energies were converged to at least \(10^{-10}\) Hartrees in all programs. Even tighter convergence would be beneficial for the numerical differentiation performed in the fitting. Numerical tests suggest that this precision in energy can lead to uncertainties in \(\alpha_e\) (proportional to \(U'''(r_e)\)) and \(\omega_e x_e\) (proportional to \(U''''(r_e)\)) as large as 0.2 cm\(^{-1}\) and 0.0001 cm\(^{-1}\), respectively.

The calculations of all diatomics and spectroscopic constants are presented in Table 2-2, and the results for \(r_e\) and \(\omega_e\) are again shown in Figure 2-5 for easier analysis. Prior to discussing
the chemical and computational implications of these results, it is worthwhile to first note that corrections for BH closely match those of a previous study\textsuperscript{20} by Temelso et al. (which used a similar but less exact extrapolation). This validates these results from a software perspective: each program must be using correct geometries, basis sets, convergence criteria, etc. The finite difference nature of the fitting procedure makes close agreement between programs particularly important.

Table 2-1 Description of calculations performed in this demo and their associated QC programs.

<table>
<thead>
<tr>
<th>Name</th>
<th>Method</th>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_{\text{Base}}$</td>
<td>$\text{CCSD(T) / cc-pCV}[Q5]Z$</td>
<td>NWChem</td>
</tr>
<tr>
<td>$E_{\text{Basis}}$</td>
<td>$\text{MP2 / (aug-cc-pCV}[56]Z - cc-pCV}[Q5]Z$</td>
<td>Psi4</td>
</tr>
<tr>
<td>$E_{\text{DBOC}}$</td>
<td>$\text{CCSD / cc-pCVDZ}$</td>
<td>CFOUR</td>
</tr>
<tr>
<td>$E_{\text{Rel}}$</td>
<td>$\text{X2C-CCSD(T) / cc-pCVTZ}$</td>
<td>Psi4</td>
</tr>
<tr>
<td>$E_{\text{CCSDTQ}}$</td>
<td>$(\text{CCSDTQ \text{CCSD(T)}} / cc-pVTZ)$</td>
<td>CFOUR</td>
</tr>
<tr>
<td>$E_{\text{FCI}}$</td>
<td>$(\text{FCI \text{CCSDTQ)}} / cc-pVDZ$</td>
<td>GAMESS/CFOUR</td>
</tr>
</tbody>
</table>
Table 2-2 Comparison between theory and experiment for bond lengths (Å) and spectroscopic constants (cm\(^{-1}\)) of three diatomic molecules. All terms correspond to the difference between a value and the case CCSD(T)/cc-pCV[Q5]Z calculation.

<table>
<thead>
<tr>
<th>Molecule &amp; Method</th>
<th>(r_e)</th>
<th>(\omega_e)</th>
<th>(\omega_e\lambda_e)</th>
<th>(B_e)</th>
<th>(D_e)</th>
<th>(\alpha_e)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BH</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Base</td>
<td>1.22890</td>
<td>2371.24</td>
<td>49.4</td>
<td>12.088</td>
<td>0.001257</td>
<td>0.423</td>
</tr>
<tr>
<td>Basis</td>
<td>+0.00018</td>
<td>0.44</td>
<td>0.4</td>
<td>0.004</td>
<td>0.000001</td>
<td>0.001</td>
</tr>
<tr>
<td>DBOC</td>
<td>+0.00065</td>
<td>2.33</td>
<td>0.2</td>
<td>0.013</td>
<td>0.000002</td>
<td>+0.000</td>
</tr>
<tr>
<td>Rel</td>
<td>0.00001</td>
<td>0.57</td>
<td>+0.1</td>
<td>+0.000</td>
<td>+0.000001</td>
<td>+0.000</td>
</tr>
<tr>
<td>CCSDTQ</td>
<td>+0.00019</td>
<td>2.07</td>
<td>+0.1</td>
<td>0.004</td>
<td>+0.000001</td>
<td>+0.001</td>
</tr>
<tr>
<td>FCI</td>
<td>+0.00000</td>
<td>+0.00</td>
<td>0.2</td>
<td>+0.000</td>
<td>+0.000000</td>
<td>+0.000</td>
</tr>
<tr>
<td>Total</td>
<td>+0.00101</td>
<td>5.41</td>
<td>0.5</td>
<td>0.020</td>
<td>+0.000000</td>
<td>+0.000</td>
</tr>
<tr>
<td>Experiment</td>
<td>+0.00318</td>
<td>4.51</td>
<td>0.1</td>
<td>0.062</td>
<td>0.000022</td>
<td>0.001</td>
</tr>
<tr>
<td><strong>HF</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Base</td>
<td>0.91654</td>
<td>4147.01</td>
<td>90.5</td>
<td>20.968</td>
<td>0.002144</td>
<td>0.793</td>
</tr>
<tr>
<td>Basis</td>
<td>+0.00017</td>
<td>1.79</td>
<td>0.7</td>
<td>0.008</td>
<td>0.000001</td>
<td>0.002</td>
</tr>
<tr>
<td>DBOC</td>
<td>+0.00001</td>
<td>+0.32</td>
<td>0.2</td>
<td>0.001</td>
<td>0.000001</td>
<td>+0.000</td>
</tr>
<tr>
<td>Rel</td>
<td>+0.00006</td>
<td>3.54</td>
<td>1.3</td>
<td>0.003</td>
<td>+0.000003</td>
<td>+0.000</td>
</tr>
<tr>
<td>CCSDTQ</td>
<td>+0.00021</td>
<td>4.49</td>
<td>+0.1</td>
<td>0.009</td>
<td>+0.000002</td>
<td>+0.002</td>
</tr>
<tr>
<td>FCI</td>
<td>+0.00001</td>
<td>0.19</td>
<td>+0.0</td>
<td>+0.000</td>
<td>+0.000000</td>
<td>+0.000</td>
</tr>
<tr>
<td>Total</td>
<td>+0.00047</td>
<td>9.70</td>
<td>2.2</td>
<td>0.021</td>
<td>+0.000004</td>
<td>+0.000</td>
</tr>
<tr>
<td>Experiment</td>
<td>+0.00027</td>
<td>8.69</td>
<td>0.6</td>
<td>0.012</td>
<td>+0.000007</td>
<td>+0.005</td>
</tr>
<tr>
<td><strong>C(_2)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Base</td>
<td>1.24039</td>
<td>1873.63</td>
<td>12.6</td>
<td>1.826</td>
<td>0.000007</td>
<td>0.017</td>
</tr>
<tr>
<td>Basis</td>
<td>+0.00016</td>
<td>1.01</td>
<td>+0.0</td>
<td>+0.000</td>
<td>+0.000000</td>
<td>+0.000</td>
</tr>
<tr>
<td>DBOC</td>
<td>+0.00001</td>
<td>+0.09</td>
<td>+0.0</td>
<td>+0.000</td>
<td>+0.000000</td>
<td>+0.000</td>
</tr>
<tr>
<td>Rel</td>
<td>0.00016</td>
<td>0.41</td>
<td>+0.1</td>
<td>+0.000</td>
<td>+0.000000</td>
<td>+0.000</td>
</tr>
<tr>
<td>CCSDTQ</td>
<td>+0.00146</td>
<td>11.76</td>
<td>+0.8</td>
<td>0.004</td>
<td>+0.000000</td>
<td>+0.001</td>
</tr>
<tr>
<td>FCI</td>
<td>+0.00100</td>
<td>4.58</td>
<td>+0.0</td>
<td>0.003</td>
<td>+0.000000</td>
<td>+0.000</td>
</tr>
<tr>
<td>Total</td>
<td>+0.00248</td>
<td>17.81</td>
<td>+0.8</td>
<td>0.007</td>
<td>+0.000000</td>
<td>+0.001</td>
</tr>
<tr>
<td>Experiment</td>
<td>+0.00206</td>
<td>18.61</td>
<td>+1.0</td>
<td>0.006</td>
<td>+0.000000</td>
<td>+0.001</td>
</tr>
</tbody>
</table>
The total extrapolation procedure shows remarkable agreement with experiment for bond lengths $r_e$ (± 0.0005 Å) except for BH, off by 0.0022 Å. However, this extrapolation lacks nonadiabatic BO effects which were found by Martin\textsuperscript{21} to be unusually high for BH, approximately 0.0025 Å. This is rather close to the overall difference of 0.0022 Å between experiment and our best estimate. Theoretical harmonic frequencies $\omega_e$ are in excellent agreement with experiment, off by only 1 cm$^{-1}$. The rotational constant $B_e$ is also well predicted, within 0.01 cm$^{-1}$ for HF and C$\textsubscript{2}$ and off by a somewhat larger 0.04 cm$^{-1}$ for BH. The latter error may be largely due to the already noted non-BO effects, which cause a larger discrepancy in $r_e$ for BH. $\omega_e\times_e$ is in good agreement with experiment, matching within 0.2-0.4 cm$^{-1}$ for BH and C$\textsubscript{2}$.
but is off by a larger 1.6 cm\(^{-1}\) for HF. It is not clear that the corrections employed here actually improve this constant, and the remaining discrepancy could be due to the numerical precision limitations discussed earlier. Similarly, \(\alpha_e\) appears to not require corrections on top of the base method, which each change it by only \(\pm 0.002\) cm\(^{-1}\) or less. Final values are within 0.005 cm\(^{-1}\) of experiment.

Figure 2-5 shows that the sum of the small corrections matches experiment very well for \(r_e\) and \(\omega_e\) except for the bond length BH, where non-BO effects are important as noted above. All of the small corrections considered can be important for \(r_e\) and/or \(\omega_e\), although there is no consistency about their relative importance from one molecule to another. For example, the DBOC is rather important for BH (which has the lightest nuclei), but not for HF and even less so C\(_2\). Similarly, the FCI correction (beyond CCSDTQ) is negligible for BH and HF but is important for C\(_2\) (worth 0.001 angstrom and 4.6 cm\(^{-1}\)).\(^{22}\) In total, the corrections lower the value of \(\omega_e\) by a surprisingly large 17.81 cm\(^{-1}\) from the base CCSD(T) value, which is very close to the experimental \(\omega_e\) (18.61 cm\(^{-1}\) lower than the base). A large majority of this change is due to missing electron correlation: the CCSDTQ correction is responsible for about 12 cm\(^{-1}\) and the FCI correction by about another 5 cm\(^{-1}\). This is presumably due to the much larger degree of electron correlation in C\(_2\), arising from the close near-degeneracy of the \([\text{core}]2\sigma_g^22\sigma_u^21\pi_x^21\pi_y^2\) and \([\text{core}]2\sigma_g^21\pi_x^21\pi_y^23\sigma_u^2\).

**Summary and Conclusions**

The Quantum Chemistry Common Driver and Databases (QCDB) project provides a simple and powerful driver front-end to multiple quantum chemistry programs, allowing users automatic access to several features formerly requiring specialized scripts or laborious post-processing. These include built-in focal-point procedures, complete-basis-set extrapolation, basis
set superposition error corrections, and combinations thereof, for not only energies, but also gradients and Hessians. QCDB also allows the ability to mix and match capabilities of multiple quantum chemistry programs within a single computation. These features have been demonstrated with an application to computing spectroscopic constants of diatomic molecules at the \textit{ab initio} limit, including corrections for post-CCSD(T) electron correlation, beyond cc-pCV[Q5]Z basis set effects, relativistic effects, and the Born-Oppenheimer diagonal correction, combining total energies computed by Psi4, CFOUR, NWChem, and GAMESS. The PyOptKing and GeomeTRIC geometry optimizers have been interfaced to QCDB and work with gradients computed at any level of theory by any of the QCDB components, or with focal-point gradients computed as a combination of multiple components. QCDB should make it much easier for end users to automate the use of focal-point approaches, complete basis set extrapolation, etc., for large numbers of molecules and in the context of geometry optimization or vibrational frequency analysis, by mixing and matching components of the quantum chemistry packages of their choice.

QCDB has been designed to work well with emerging tools and standards developed by the Molecular Sciences Software Institute (MolSSI). In particular, QCDB and the Psi4 driver have been developed to utilize the MolSSI QCSchema JSON format for information passing, and were early test beds for the development of that standard. QCDB input writing and output parsing capabilities have also been developed in close collaboration with the development of the QCEngine component of the MolSSI QCArchive project. The interoperability framework is designed to make it easy to interface to additional quantum chemistry packages in the future.

For users, the advantages of adopting QCDB and the QCArchive Infrastructure are wide applicability of a single molecule representation, a uniform options input syntax of Python, and
the imposition of best-practice options without consulting the manual. For QC code developers, advantages are the easy encoding and tweaking of best-practice options in an easy Python layer rather than native knobs, a lower barrier to new users trying the code, and enhanced defaults validation. For power users, advantages are an easy interface where, at worst, the original output will still be available with a quick hook-up to post-processing without compiled languages.

The power of the combination of all these features will actually take a while to explore. We have recently submitted papers on the use of focal-point approaches as approximations to complete-basis-set CCSD(T) in geometry optimizations and for vibrational frequency computations. These explorations were made much easier through their automated implementation in the Psi4 (and now QCDB) drivers. The focal-point approach appears to give very good approximations to CCSD(T)/CBS at a substantially reduced computational cost. The need for programmatic access to QC results is certain. The interface, volume, and intricacy requirements of that access are widely varying across workflows. The QCArchive and QCDB projects provide the pieces/tools/adaptations to transition to programmatic access or to commence with it so that QC predictions can take their place in molecular modeling.

Acknowledgements

This work was supported in part by the U.S. National Science Foundation (Grants No. ACI-1449723, OCI-1047772, ACI-1547580, ACI-1450217).

Supporting Information

The supporting information contain detailed information about maintenance and capabilities of the QCDB and QCEngine APIs.

Maintenance

To support confidence in the code and freedom to improve it, QCEngine and QCDB undergo automatic testing, including dynamic, static, and coverage analysis. In particular, both
modules undergo unit testing of central Python code and integration testing by running full QC jobs using cloud testing infrastructure on private resources. QC Engine is developed by MolSSI software scientists and by the many contributors to the various harnesses. QCDB was once an internal upstream dependency of Psi4 and in future will become its required downstream dependency, ensuring QCDB’s future maintenance.

**QC Program Capabilities**

Details and special features from each interfaced QC program are described below. If results are available programmatically through QCVariables as opposed to text files, there may be a lag behind runnability where controlling keywords are available and QCDB will run and receive back results. Well-known QC methods whose running, output harvesting, and consistent output values have been established are collected in the supporting information. Table 2-3 has a list of working energy calls from each of the QC programs integrated into the QCDB.

Many CFOUR features are available, including most ground-state many-body perturbation theory and coupled-cluster energies, gradients, and Hessians. Excited states are available for running but not parsing. Special features include the ability to compute the diagonal Born—Oppenheimer correction using coupled-cluster theory and vibrational perturbation theory (VPT).

The GAMESS interface provides Hartree—Fock, DFT, MP2, and coupled-cluster methods. Special features include full configuration interaction. In addition, the GAMESS interface also provide effective fragment potential (EFP) capability through potential file generation and running pure EFP calculations on molecular clusters, energy(“gms-efp”). A particular complication for GAMESS is the controlled molecule and custom basis syntax, which led to QCDB feeding only symmetry-unique atoms and their full basis sets into GAMESS input.
deck. As QCEngine does not have symmetry capabilities, QCEngine restricts GAMESS calculations $C_1$.

The NWChem interface provides a large selection of the quantum mechanical methods available, including Hartree—Fock, DFT, MP2, and coupled-cluster method (both the code automatically derived and implemented with the TCE, and the hand-coded implementations where available). Additional calculations in the TCE are available which include the configuration interaction through singles, doubles, triples, and quadruples level of theory and MBPT methods through the fourth order. Special features include the CCSDTQ energies, excited states of motion (EOM) coupled-cluster energies, and relativistic approximations.

Essentially all Psi4 features are available, as QCDB began as the Psi4 driver. These include conventional and density-fitted Hartree—Fock, DFT, MP2, and coupled-cluster methods. Special features are symmetry-adapted perturbation theory, coupled-cluster response properties, density-fitted CCSD(T) gradients, and optimized orbital MP2, MP2.5, and MP3 energies and gradients.

A Python API to Grimme’s dftd3 executable for computing variants of -D2 and -D3 for arbitrary QCSchema Molecule with automatic or custom parameter sets has been available in Psi4 for several years.\textsuperscript{23-25} This has been adapted as a ProgramHarness for QCEngine and with the usual registrations of keywords and methods is available in QCDB.

A Python API to Kaliman’s PylibEFP library for computing interaction energy components between EFP2 fragments has been available in Psi4 since v1.3. Similarly, to Sec II C 5 this has been adapted as a ProgramHarness for QCEngine and QCDB and so, along with GAMESS, is available for running pure EFP computations. At a different level of integration
outside the scope of QCDB that Psi4 takes advantage of PyLibEFP can supply the EFP terms for QM self-consistent field computations.

MolSSI and community contributors have developed additional ProgramHarnesses for QCEngine that are not yet or are not suitable for integration with QCDB. These include Entos\textsuperscript{26}, MolPro\textsuperscript{27,28}, Mopac, MP2D\textsuperscript{29}, OpenMM, Q-Chem\textsuperscript{30}, RDKit, TeraChem\textsuperscript{31,32}, and TorchANI\textsuperscript{33}. QCEngine ProcedureHarnesses but not QCDB procedures can make use of these programs.

**Procedure Capabilities**

As shown in Figure 2-3, there are quite a few procedures that are available in either QCEngine or QCDB. Some procedures are pulled from the QC programs previously described, i.e., vpt2 from CFOUR or makefp from GAMESS. Other procedures have been developed with QCDB’s interoperability goal in mind, i.e., finite difference derivatives. Many of these procedures are already able to process through QCSchema and are parallelism-ready.

Presently available in QCEngine is the geomeTRIC geometry optimizer\textsuperscript{34}, which has been extensively used by the Open Force Field community. Presently available in QCDB are the Composite, FiniteDifference, ManyBody, diatomic, and vib routines inherited from the Psi4 Distributed Driver. The OptKing geometry optimizer\textsuperscript{35} has been translated to Python from C++ for the QCDB project while restrained electrostatic potential\textsuperscript{36,37} (resp) has been expanded from Psi4 to work with QCDB. Procedures makefp and vpt2 make use of specially extractable features from GAMESS and CFOUR, respectively, and have been guided by their primary developers.
Table 2-3 Confirmed working energy methods in QCDB. Alternate implementations listed in rows within a QC Program, available analytic methods listed with “Y” and default implementation for a given program, method, reference, and all-electron/frozen-core (ae/fc) choice listed with “D”.

<table>
<thead>
<tr>
<th>Method &amp; Harness</th>
<th>RHF energy()</th>
<th>UHF energy()</th>
<th>ROHF energy()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RHF</td>
<td>UHF</td>
<td>ROHF</td>
</tr>
<tr>
<td></td>
<td>CV</td>
<td>CV</td>
<td>CV</td>
</tr>
<tr>
<td></td>
<td>ae/fc</td>
<td>ae/fc</td>
<td>ae/fc</td>
</tr>
<tr>
<td>HF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CFOUR</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>GAMESS</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>NWChem</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>Psi4</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>MP2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CFOUR</td>
<td>D/D</td>
<td>D/D</td>
<td>D/D</td>
</tr>
<tr>
<td>GAMESS</td>
<td>D/D</td>
<td>D/D</td>
<td>D/D</td>
</tr>
<tr>
<td>NWChem</td>
<td>D/D</td>
<td>D/D</td>
<td>D/D</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tce</td>
<td>Y/Y</td>
<td>Y/Y</td>
<td>D*/D*</td>
</tr>
<tr>
<td>Psi4</td>
<td>D/D</td>
<td>D/D</td>
<td>D/D</td>
</tr>
<tr>
<td>CCSD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CFOUR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vcc</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ecc</td>
<td>D/D</td>
<td>D/D</td>
<td>D/D</td>
</tr>
<tr>
<td>ncc</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GAMESS</td>
<td>D/D</td>
<td></td>
<td>D/D</td>
</tr>
<tr>
<td>NWChem</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cc</td>
<td>D/D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tce</td>
<td>Y/Y</td>
<td>D/D</td>
<td>D/D</td>
</tr>
<tr>
<td>Psi4</td>
<td>D/D</td>
<td>D/D</td>
<td>D/D</td>
</tr>
<tr>
<td>fnocc</td>
<td>p/p</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCSD(T)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CFOUR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vcc</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ecc</td>
<td>D/D</td>
<td>D/D</td>
<td>D/D</td>
</tr>
<tr>
<td>ncc</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GAMESS</td>
<td>D/D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NWChem</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cc</td>
<td>D/D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tce</td>
<td>Y/Y</td>
<td>D/D</td>
<td>D*/D*</td>
</tr>
</tbody>
</table>
Table 2-3 Continued

<table>
<thead>
<tr>
<th>Method &amp; Harness</th>
<th>RHF</th>
<th>energy()</th>
<th>UHF</th>
<th>ROHF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CV</td>
<td></td>
<td>CV</td>
<td>CV</td>
</tr>
<tr>
<td></td>
<td>ae/fc</td>
<td></td>
<td>ae/fc</td>
<td>ae/fc</td>
</tr>
<tr>
<td>Psi4</td>
<td></td>
<td>D/D</td>
<td></td>
<td>D/D</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td>D/D</td>
<td></td>
<td>D/D</td>
</tr>
<tr>
<td>fnocc</td>
<td></td>
<td>p/p</td>
<td></td>
<td>D/D</td>
</tr>
</tbody>
</table>

The QCDB harmonic vibrational analysis routine is automatically run after any frequencies() computation. The ProcedureHarness takes in a Hessian matrix, the molecule, basis set information, and optional dipole derivatives through pure Python. There is still a lingering cdsalcs dependency wherein it performs the usual solution of whole or partial Hessians into normal modes and frequencies, reduced masses, turning points, infrared intensities, all returned in QCSchema. Other features include rotation translation space projection, quick reanalysis upon isotopic substitution, Molden\textsuperscript{17,18} output, and a full thermochemical report.

As QCDB’s emphasis is on interfacing QC programs’ quantum chemical methods, or unique features, rather than on their internal drivers such as optimizers or finite difference methods. User calls for non-analytic methods in a given program are by default routed through the QCDB finite difference procedure. This pure-Python implementation has been inherited from Psi4 and is capable of performing 3- and 5- point stencils for gradients, Hessians, and dipoles. Symmetry-adapted internal coordinates are employed, and partial Hessians may be computed. The procedure communicates through QCSchema and is parallelism-ready. The alternative of parsing output files that use a program’s glue code (such as multiple energies/gradients from internal optimization or finite different) has been implemented in some cases and can be expanded but is not preferred.
Whenever an additive model chemistry is prescribed involving differences of method (i.e., a focal point analysis or Δ correction), basis (i.e., a CBS extrapolation), keywords (e.g., all electron minus frozen core), or any combination thereof, the composite() procedure can encode it. Here, one can mix QC programs so as to do coupled cluster with CFOUR and MP2 with Psi4, for example. Implementing new basis extrapolation formulae is simple, and works on gradients and Hessians, as well as energies. If a lesser method energy can be obtained in the course of a needed greater method call, the procedure will avoid the unnecessary calculation (thus a TQ MP2 correlation energy extrapolation atop a DTQ HF energy will do 3, not 5, jobs). Input specification can be highly programmatic, through schema, or user-friendly.

All fragmentation and basis set superposition error (BSSE) treatments are collected into the 
\texttt{qcdb.manybody()} wrapper for many-body expansion (MBE) inherited from Psi4. The fragmentation pattern known from the \texttt{QCSchema Molecule} is applied to determine the degree of decomposition into monomers, dimers, etc. up to the full molecule, or the user can set the max n-body level. Total quantities (energy, gradient, or Hessian) and interaction quantities are accessible through uncounterpoise (noCP), counterpoise\textsuperscript{38} (CP), and Valiron-Mayer\textsuperscript{39} (VMFC) schemes, which may be run individually or presented separately from a comprehensive run. Geometry optimization with n-body-adapted quantities is also available. The wrapper can act on uniform single-method quantities or apply different model chemistries to each expansion level or interface with \texttt{qcdbcomposite()} or \texttt{qcdb.finitedifference()} results or both.

Anharmonic vibrational analysis has long been a feature of CFOUR. It requires a high-quality harmonic frequency procedure, then invocations to take geometry displacements along the normal coordinates and carry out further harmonic frequencies at each displacement, then invocations to combine the results into a third-order and partial fourth-order potential plus
vibrational analysis. Though many analytic Hessians are available in CFOUR, the capability to use analytic gradients in manual pleasantly parallel fashion is available and is perfectly suited to generalization to program-generic gradients. In this way, CFOUR is a helper program to produce anharmonic analyses of CCSD (internally, through QCDB calls), DFT (from another QC program), or CBS (that produces a generalized gradient). All qcdb.vpt2 communication is through schema, and the procedures is parallelism-ready.

A particular aspect is that the vpt2() procedure, which is essentially a series of invocations of the CFOUR subcommands like xcubic, expects the files with energies, dipoles, and gradients upon which it works to be in native JOBARC form. To accommodate this, QCDB uses Python modules to write imitations of the native files in string representations of binary form, which is lossless. Thus, Psi4 DFT gradients, for example, can be represented as a JOBARC and pass through the CFOUR mechanisms.

The two engines for computing EFP interactions, LibEFP and GAMESS, use the same parameter file for storing the EFP potential at a given basis set and molecular fragment. Only GAMESS can generate that file, and the routine has been wrapped by QCDB to access through qcdb.makefp. The resulting .efp file contents are returned in the JSON output and are available for writing to a personal library or to feed to subsequent "gms-efp", "lefp-efp", or "p4-efp" invocations of qcdb.energy() to determine non-covalent interactions between EFP fragments. This capability allows on-the-fly generation and use of EFP fragments.

The electrostatic potential analysis for diatomic molecules and RESP charge model exists in Psi4 as a post-processing procedure and plugin, respectively. The electrostatic potential analysis uses a list of electronic energies along the interatomic coordinate. The RESP charge model is obtained by an iterative fitting of the electrostatic potential emerging from QC
calculations on one or several conformers of a molecule to a classical point-charge potential.

This has been expanded to alternately draw from GAMESS using QCDB.

References


11. https://github.com/MolSSI/QCElemental


14. RDKit: https://www.rdkit.org/

15. Open MM: http://openmm.org/


22 J. S. Boschen et al., Theoretical Chemistry Accounts 133 (2013) 1425.


26 M. Frederick et al., entos: A Quantum Molecular Simulation Package 2019),


35 OptKing:  
http://www.psicode.org/psi4manual/master/autodir_options_c/module__optking.html


CHAPTER 3. COMPUTATIONAL CHEMICAL INSIGHT ON POTENTIAL ZIRCONIUM-BASED CATALYST

Introduction

Transition metal complexes have been of interest in the catalysis community given they are readily available and abundant in nature. Industry, in particular, is interested in using catalysts to reduce time constraints, costs, and scaling up the amount of product produced. Iron complexes, for example, have been attractive due to their non-toxic, sustainable, and cost-effective properties. Zirconium is another transition metal known to be abundant, sustainable, and cost-effective for catalytic chemistry with broad research applications. A notable example is the Ziegler-Natta catalysts, which are used in the synthesis of alpha-olefin polymers, and Zr complexes are a part of the homogenous class of Ziegler-Natta catalysts. However, achieving and sustaining catalytic activity with metal complexes has been an issue.

Concerns about catalytic activity rest not only on what metal can provide the chemistry but the ligands as well. Redox-active ligands, aka “non-innocent” ligands, can play a crucial role in catalytic reactions as new tools that can control the catalytic activity and selectivity of transitional metal complexes. As an active participant in the reaction, non-innocent ligands do not change the oxidation state of the metal, which have led to the discovery of highly active and versatile catalysts. The hydroamination reaction of a mono-substituted cyclopentadienyl-oxazolide, bis-amide zirconium complex (shortened to bis-amide Zr-complex) to form the mono-substituted cyclopentadienyl-oxazoline tris-amide zirconium complex (shortened to tris-amide Zr-complex), as shown in Figure 3-1, is the focus of this chapter.
Figure 3-1 The hydroamination reaction using dimethylamine with the bis-amide Zr-complex (left) to form the tris-amide Zr complex.

**Reaction**

It has been previously\textsuperscript{9-10} shown that the combination of cyclopentadienyl and oxazolinyl ligands are more catalytically activity due to the carbanionic character of oxazoline stabilizing the electrophilic nature of the cyclopentadienyl. Catalytic hydroamination reactions in combination with the selected ligands have not been explored. It is, however, known that catalytic hydroamination is an initial test of catalytic species reactivity\textsuperscript{10}. Thus, the bis-amide Zr-complex would undergo a ligand substitution, switching the oxazolide ligand with the dimethylamine, while the H transfers to the benzylic carbon, shifting the double bond to the oxazolide transforming it to an oxazoline. The reaction shown in Figure 3-1 was experimentally conducted by Yang-Yun Chu and Professor Aaron Sadow of Iowa State University. Possible mechanisms for the reaction include: 1) a dissociative substitution where the dimethylamine will sit over the oxazolide’s bond to the Zr-center to replace the oxazolide bond as well as approach the benzylic carbon for the H transfer in a tandem fashion for a one-step process, 2) an associative substitution where there is a stable intermediate between the bis-amide Zr-complex reactant and the tris-amide Zr-complex product, 3) an interchange substitution, which is an mix of the dissociative and associative substitution, and the reaction process is done in one step. The experimental context for the mechanism has proven that the reaction favors a dissociative
substitution as the mild reaction conditions were not conducive for stepwise mechanisms.

Obtaining a potential transition state structure and other computation calculations, such as the dimethylamine’s binding energy, provide insight into how the reaction progresses. The calculations elucidate whether or not the energy barrier between the bis-amide Zr-complex (the reactant) and the tris-amide Zr-complex (the product) is low to ease the catalytic cycle bottleneck, as a key component of catalysis is a reaction’s reversibility.

**Transition State Searching**

![Diagram of a reaction coordinate showing the reactants (A), transition state (B), products (C), activation energy (D), and overall energy of the system (E).](image)

Figure 3-2 A general reaction coordinate for an exothermic reaction that points out the reactants (A), the transition state (B), the products (C), the activation energy (D), and the overall energy of the system (E).

Chemical reactions are visually depicted by a reaction coordinate which shows the energetics of a system as a reactant transforms into a product (Figure 3-2). The transition state (TS) is a point on the reaction coordinate where there is a local maximum, known as the saddle point, which can be visualized by a potential energy surface (PES) that plots the molecular geometries against the energy. The saddle point is equivalent to the highest point of the reaction
coordinate. In computational chemistry, quantum chemistry software can determine a possible TS through several methods, including saddle point searching and chain-of-state methods\textsuperscript{11-13}. Saddle point searching requires a best guess at the potential geometry of a TS as the program will walk uphill from its local position, if possible. A user can feed in a Hessian to better guide the saddle point search on the PES. To ease the calculation of a potential TS, the development of chain-of-state methods\textsuperscript{11} allows a user to come closer to an approximation of a potential saddle point through creating images between the two endpoints of a reaction while simultaneously minimizing the system to produce a minimum energy path (MEP). The replica path (RPATH) method introduces spring forces to apply constraints on the adjacent images in the reaction pathway as well as maintain path smoothness, which has been useful in macromolecules\textsuperscript{11}. Nudged elastic band is one example of these groups of methods that was used to find the transition state of the system in this work.

The trial and error of both aforementioned methods can impede finding an optimized TS. While there are algorithms that can generate a guess for the TS, they can be complicated and typically non-generalizable for many reactions.\textsuperscript{14} There have also been attempts\textsuperscript{14} to automate the transition state search process since starting with a good initial guess for a TS can be challenging. These attempts\textsuperscript{14}, described in Jacobson et al., however, have been restricted to smaller basis sets and cost-effective quantum chemistry methods such as density functional theory (DFT). There have been other efforts in automated TS search methods\textsuperscript{12,15-16}, though whether or not these practices carry over to larger molecules and more complex reactions have yet to be explored.

**Nudged elastic band**

The nudged elastic band (NEB) method\textsuperscript{17-18} is a statistical approach in estimating the MEP. The initial and final states of the system are the boundary points that NEB uses to create
images, also known as beads, between the predetermined endpoints and to craft an MEP. The name stems from its distinct feature of having springs connect adjacent images together while ensuring the spring and atomic forces are adjusted without interfering with one another through orthogonality. The springs control the movement each image takes and is nudged, as an elastic band does, towards the MEP. Mathematically, an elastic band of $N + 1$ images can be represented as $[R_0, R_1, R_2, \ldots R_N]$ where $R_0$ and $R_N$ denote the fixed initial and final states and the energy minima of said states, respectively. $\hat{\mathbf{t}}_i$ is the normalized tangential force of the spring at image $i$ and can be used to determine the MEP closest to the initial guess.

$$\hat{\mathbf{t}}_i = \frac{R_{i+1} - R_{i-1}}{|R_{i+1} - R_{i-1}|}$$  \hspace{1cm} (1)

The computational cost of NEB can be more expensive than a saddle point search; however, in this case, NEB was beneficial due to difficulty in pinning down a better TS guess. The visualization of the MEP is an added bonus of NEB compared to a TS saddle point search as the information is readily available with the calculation.

**Computational Methods**

All calculations reported have been done using NWChem.$^{19}$ For the TS search, NEB was used for the reaction of dimethylamine addition to the bis-amide Zr-complex to provide a better estimate of a TS before using the saddle point search. The number of beads was increased to 12 to aid the calculation of the MEP; the algorithm was switched from the default quasi-Newton fixed point optimization$^{13}$ to the refining conjugate gradient$^{20}$ for reducing the maximum gradient. Additional parameters included the maximum iterations to be set to 20, step size 0.75, and convergence tolerance 0.00045 atomic units. After finding the TS guess, we are then able to run a saddle point search with a calculated Hessian to improve the search across the PES for a TS. The final level of theory used for optimizing the transition state were the DFT functional
B3LYP\textsuperscript{21-22} with a fine grid and a 6-311++G* basis set\textsuperscript{23} as well as the LANL2DZ\textsuperscript{24} for the effective core potential on Zr.

**Results and Conclusion**

The optimized binding energy of the dimethylamine to the bis-amide Zr-complex, as a replacement ligand, was calculated as -1.9 kcal/mol. From this dimethylamine bond structure, the overall reaction as represented in Figure 3-1 is slightly exothermic, $\Delta E_{\text{react}} = -3.9$ kcal/mol. The TS geometry (Figure 3-3) has the hydrogen on the nitrogen of the incoming dimethylamine stretching its bond as it moves towards the benzylic carbon, while the nitrogen of the oxazoline begins to separate from the Zr center as shown in the reaction progression in Figure 3-4. The activation energy from the reactants of bis-amide Zr-complex and dimethylamine to the TS structure (Figure 3-3) was determined to be 19.1 kcal/mol. All values can be found in Table 3-1.

As explained previously, there were multiple reaction mechanisms for the MEP. The basic outline of the reaction consisted of exchanging the oxazolide ligand with the dimethylamine via hydroamination and the H transfer to the benzylic carbon. The calculations studied the reaction as stepwise, with the dimethylamine binding before the H transfer, indicating an interchange substitution mechanism as there was no stable intermediate. Quantitatively, the amine binding energy (-3.9 kcal/mol) is small compared to the overall activation energy of the reaction, 19.1 kcal/mol. This suggests that there is very little difference among the different possible mechanisms that were considered since the binding of the dimethylamine is only slightly exoergic. Further calculations and studies at multiple dissociative pathways would need to be performed before ruling it out completely.

Additional work can be done with similar Zr-systems, as steric effects on the oxazolide, by changing the methyl substituents to bulkier ones, i.e., isopropyl, phenyl, tertbutyl, have been known to limit the catalytic activity of the chemical system. Another avenue of exploration in
understanding the catalytic reactivity and stability would be to exchange the zirconium center for either hafnium or titanium. Both metals are common analogs for zirconium, but given the size differences between the atoms, there is potential that the catalytic behavior may be different.

Figure 3-3 The geometry of the transition state where the marked atoms (encircled in green) indicate the N of the dimethylamine and its hydrogen transferring to the benzylic carbon. The smaller gray atoms are C, blue are N, white are H, red O, and the single, larger black atom is Zr.
Figure 3-4 The hydroamination reaction progress (left to right) with the addition of the dimethylamine, where the Zr binds the N of the amine first. As the dimethylamine bonds to Zr, the oxazolide structure moves away, and the hydrogen off of the incoming dimethylamine moves towards the benzylic carbon. Finally, the tris-amide-Zr-complex is formed. The upper righthand structure is a depiction of the TS.

Table 3-1 Summary of the important energetic quantities for the hydroamination reaction with the bis-amide Zr complex (kcal/mol).

<table>
<thead>
<tr>
<th>State</th>
<th>Energy (kcal/mol)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activation energy ($E_A$)</td>
<td>19.1</td>
</tr>
<tr>
<td>$\Delta E_{\text{react}}$</td>
<td>-3.9</td>
</tr>
<tr>
<td>HNMe$_2$ binding energy</td>
<td>-1.9</td>
</tr>
</tbody>
</table>

References


CHAPTER 4. GENERAL CONCLUSION

Since the QCArchive’s public beta release in fall 2019, the integration of additional functionalities has been buoyed by input from the community at large. QCDB’s role in providing best practices has included the integration of four quantum chemistry programs, four basic computational functions (energy, gradient, Hessian, and properties), a myriad of options from the aforementioned programs, alongside interoperable functionalities to work with multiple programs. NWChem’s integration with the interoperability project is detailed further in the Appendices of this work. Additional work to capture more of the quantum mechanics options and functions can still be done, including direct MP2, resolution of the identity (RI) MP2, optimization, etc. However, the current work shows the power of interoperability to enable more complex and integrated computations.

The computational insight of the hydroamination reaction of dimethylamine with the bis-amide Zr-complex has shown a potential reaction pathway. Zirconium is just one transition metal among many that has been fruitful for catalysis. Future work in computing the reaction with a different metal like titanium or hafnium could be performed to monitor the size effect of the metal. Potential steric effects on the non-innocent ligand oxazolide are another avenue of study that can affect the reaction, and future computational calculations can be performed by changing the substituents on the oxazolide.
APPENDIX A. WORKING ENVIRONMENT FOR QCDB / QCARCHIVE

For use of the Quantum Chemistry Common Driver and Databases (QCDB) and related modules from the QCArchive ecosystem, please use these instructions to build a Conda environment. This environment will be activated on your workstation when working on the QCDB; otherwise, testing and running calculations will not behave properly.

1. Build any quantum chemistry program that you’ll be working with, i.e., NWChem, GAMESS, PSI4, etc. Given the original four programs that were enlisted as part of the QCDB project, you can find information on their installs here:
   a. NWChem 6.8 can be forked and cloned from GitHub
      
      https://github.com/nwchemgit/nwchem/
   
      i. Older NWChem installations can be found on the website to compile the source: http://www.nwchem-sw.org/index.php/Download.
   b. GAMESS Installation can be requested at
      
      https://www.msg.chem.iastate.edu/gamess/download.html
   c. PSI4 can be forked and cloned from GitHub: https://github.com/psi4/psi4
   d. CFOUR can be requested following the directions at https://www.cfour.de

2. Ensure that you have Conda/Miniconda installed.

   (https://docs.conda.io/en/latest/miniconda.html)

3. On GitHub, you’ll want to fork and clone the following repos to be on the same directory level:
   a. qcdb: https://github.com/qcdb/qcdb,
   b. qcengine: https://github.com/MolSSI/QCEngine, and
   c. qceelemental: https://github.com/MolSSI/QCElemental
In addition, each of these repositories should have a remote origin that points to your forked repository, and a remote upstream that points back to the original/master repository as shown in the links above. Thus, when you do `git remote -v`, it should return:

```
origin https://github.com/youruserid/qcdb (fetch)
origin https://github.com/youruserid/qcdb (push)
upstream https://github.com/qcdb/qcdb.git (fetch)
upstream https://github.com/qcdb/qcdb.git (push)
```

If a primer is needed about remote branches and how to update from your local workstation to the forked repository, please refer to [this article from Atlassian](https://www.atlassian.com/git/tutorials/git-forks-and-upstreams).

4. Once all of the previous steps are done, open your terminal and carry out the following commands:

```
conda create -n envname python=3.7 psi4 pyyaml -c psi4'label/dev
conda activate envname
conda install bson -c conda-forge
conda remove qcelemental qceengine -force
cd /path/to/qcelemental; pip install -e
```

The last two are important as it reinstallss the live fork of QCElemental and QCQEngine rather than what is packaged in the conda environment, which may be an older, stable version, especially if you are contributing to the code.

5. Create a script with the following commands and information that can be sourced to activate the environment and check that all the components are in functioning order.

```
conda activate envname

QCDBLOC=/location/of/qcdb/top-level/directory

export PATH="/include/all/paths/to:/different/qc/programs/using:
as/well/as/miniconda/path/if/not/already/set"

export PYTHONPATH=$QCDBLOC

which qcprog1 qcprog2 ... python conda
```
python -c "import psi4; print(psi4.__version__); import qcelemental as qcel; print(qcel.__version__); import qcengine as qcng; print(qcng.__file__, qcng.__version__); import qcdb; print(qcdb.__file__, qcdb.__version__)"

You can expect to see something along the lines of this:

/Users/alolinco/miniconda3/envs/py37/bin/psi4
/Users/alolinco/miniconda3/envs/py37/bin/python
/Users/alolinco/nwchem-6.8/bin/MACX64/nwchem
/Users/alolinco/gamess/rungms
1.4a2.dev215
v0.12.0
v0.12.0+73.gd9a033f.dirty /Users/alolinco/qcengine/qcengine/__init__.py
/Users/alolinco/qcdb/qcdb/__init__.py 0.1

One common error that should be checked is the path to python, or any program that is not locally installed, directs to a local version rather than the conda environment.
APPENDIX B. RUNNING CALCULATIONS THROUGH QCDB

An extended selection of quality assurance tests can be found in the GitHub repository path `qcdb/qcdb/tests` (https://github.com/qcdb/qcdb/tree/master/qcdb/tests) and be used as templates for creating an example calculation. The input files are Python-based files (`file.py`). It is important that the `file imports addons and utils` which relies on where the `file.py` is placed within the repository.

Key Components

There are three functions that QCDB has developed when running tests:

- `qcdb.set_molecule()`

  This function is where the geometry of the molecule is set. Both Cartesian and z-matrix coordinates are accepted. However, the default assumption is that the units will be given in angstroms. For other unit types, the type needs to be explicitly set within this function, i.e., units au, units bohr, etc. If symmetry needs to be forced, that is also set within this function.

- `qcdb.set_options()`

  All agnostic and program-specific options can be set here in a key-value pair. Boolean, string, integer, float, list, tuple, and dictionary keywords are accepted as the values. There are some capabilities for nesting in the key or option, to accommodate structured data found in programs like NWChem with its block input structure. Once the program is called and the specific hierarchy within the program is determined by a couple of underscores (__) to separate the options. This allows native quantum chemistry program users to call `gameess_contrl__scftype="rhf"` or `nwchem_scf__rhf: True`. Special, non-intuitive options for NWChem are discussed in more detail in Appendix C.

- `qcdb.energy/gradient/Hessian/properties("program-level of theory")`
This function can run energy, gradient, Hessian, and property calculations for a single or multiple quantum chemistry programs.

**Basic Input**

This example is for a single program input. QCDB is calling a Hartree—Fock calculation in NWChem for a water molecule in a Dunning correlation-consistent double-zeta basis set with 600 mb of memory and printing the QCVariables that are harvested.

```python
import os
import qcdb
from .utils import *
from .addons import * #depending on where you’re writing file

h2o = qcdb.set_molecule('''
O
H 1 1.8
H 1 1.8 2 104.5'''
)

qcdb.set_options({
    'basis': 'cc-pvdz',
    'memory': '600 mb'
})

qcdb.energy('nwc-hf')
print(qcdb.print_variables())
```

Note that additional libraries and modules may be needed as necessary, e.g., numpy may be used for collecting gradients and other data that are formatted into arrays. It is also important to take note of where certain directories are with respect to your file, as shown in the example.
Advanced Input

This example is for a multi-program input.

```python
import qcdb

cpdb.set_molecule(""
O
H 1 1.0
H 1 1.0 2 90.0
"")

cpdb.set_options({
    "nwchem_ccsd__freeze": 1,
    "psi4_freeze_core": True,
})

en = qcdb.energy("p4-mp2/aug-cc-pv[Q5]Z + D:gms-ccsd/cc-pv[DT]Z + D:nwc-ccsd(t)/cc-pvDZ")

cpdb.print_variables()

cpdb.compare_values( -76.36750957, en, 6, "check")
```

Here, the `qcdb.energy()` is determining the energy of water using three different programs (Psi4, GAMESS, and NWChem) and comparing a composite value to a given value of water (-76.36750957 h). Psi4 is running a MP2 calculation with an augmented correlation-consistent quintuple-zeta basis set; GAMESS is using coupled-cluster singles and doubles with basis set extrapolation between correlation-consistent double and triple zetas; NWChem is calling a hand-coded coupled-cluster singles and doubles with parentheses-triples correction with a correlation-consistent double-zeta basis set.
Output

Variable Map:

```
<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;CURRENT CORRELATION ENERGY&quot;</td>
<td>0.000000000000</td>
</tr>
<tr>
<td>&quot;CURRENT ENERGY&quot;</td>
<td>-76.027308505989</td>
</tr>
<tr>
<td>&quot;CURRENT REFERENCE ENERGY&quot;</td>
<td>-76.027308505989</td>
</tr>
<tr>
<td>&quot;HF TOTAL ENERGY&quot;</td>
<td>-76.027308505989</td>
</tr>
<tr>
<td>&quot;NUCLEAR REPULSION ENERGY&quot;</td>
<td>9.240199677600</td>
</tr>
<tr>
<td>&quot;SCF TOTAL ENERGY&quot;</td>
<td>-76.027308505989</td>
</tr>
</tbody>
</table>
```

When a QCDB job is processed, a variable map like the one shown above is printed after the native quantum chemistry programs finished running calculations. Beyond the QCDB output file that is post-processed, the verbose setting in the runner.py can be changed to include printing out the native program’s output file as well. The runner.py can be found in each program’s directory, e.g. NWChem’s runner.py can be found in qcdb/qcdb/programs/nwchem (https://github.com/qcdb/qcdb/tree/master/qcdb/programs/nwchem).
APPENDIX C. QCDB CAPABILITIES WITH NWCHEM

An extensive look at the basic features of NWChem that were integrated into the Quantum Chemistry Common Driver and Databases (QCDB) can be found in Jiyoung Lee’s thesis\textsuperscript{1}. Since then, the QCDB has undergone structural changes that were worked on for the purpose of this thesis. The QCDB is no longer embedded within the Psi4 program and is an independent wrapper across the quantum chemistry programs. Below are additional capabilities and notable changes of NWChem that were integrated during the redesign of the QCDB. The NWChem Wiki documentation covers the native options at https://github.com/nwchemgit/nwchem/wiki that a native NWChem user can familiarize themselves with.

Redesigned Capabilities

Due to the block input system of NWChem, translation of options from NWChem for QCDB made for a challenge, as discussed in Chapter 2. The Tensor Contraction Engine (TCE) was one such QCDB module that had to be reconfigured during the rehaul of the QCDB code. Now to call any of the modules within the TCE, the option $\texttt{qc\_module}$ must be set to the string “TCE” (not case sensitive). Additionally, for multiple exchange correlation functionals within the DFT block, a string input is allowed for greater access to functional options within NWChem so that both alpha and numeric values are accepted.

Additional Capabilities

Additional options from NWChem that were added with the edits of the QCDB during this thesis including more quantum mechanics theories and TCE modules alongside options to include relativistic effects and property options as can be seen in Table C-1 and Figure C-1 respectively.
Figure C-1 A breakdown of the current NWChem implementation for the QCDB and QCEngine functionalities for (a,b) energy and gradients and (b) Hessian, properties and relativistic effects. The colors indicate if the modules are redesigned from an earlier version of the QCDB (purple), newly added (teal), testing needed (teal with orange outline), or not fully implemented in some fashion (blue with orange outline). The top level of the figure shows the block heading that NWChem’s input structure has. Those that are a mix of colors at the top level may be a more generalized option, i.e. the Tensor Contraction Engine (TCE).
Figure C-1 Continued
Table C-1 Additional theory and module options capable of running in QCDB for NWChem. The majority of the TCE is now available for use.

<table>
<thead>
<tr>
<th>Option Keywords</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nwc-mcscf</td>
<td>Multiconfiguration Self Consistent Field</td>
</tr>
<tr>
<td>nwc-sodft</td>
<td>Spin orbit density functional theory</td>
</tr>
<tr>
<td>nwc-cisd</td>
<td>Configuration interaction singles and doubles via Tensor contraction Engine (TCE)</td>
</tr>
<tr>
<td>nwc-cisdt</td>
<td>Configuration interaction singles, doubles, and triples via TCE</td>
</tr>
<tr>
<td>nwc-cisdtq</td>
<td>Configuration interaction singles, doubles, triples, and quadruples via TCE</td>
</tr>
<tr>
<td>nwc-qcisd</td>
<td>Quadratic configuration singles &amp; doubles via TCE</td>
</tr>
<tr>
<td>nwc-mp2</td>
<td>2\textsuperscript{nd}-order Moller-Plesset perturbation theory (non-density-fitting) via TCE</td>
</tr>
<tr>
<td>nwc-mp3</td>
<td>3\textsuperscript{rd}-order Moller-Plesset perturbation theory (non-density-fitting) via TCE</td>
</tr>
<tr>
<td>nwc-mp4</td>
<td>4\textsuperscript{th}-order Moller-Plesset perturbation theory (non-density-fitting) via TCE</td>
</tr>
<tr>
<td>nwc-ccsd[t]</td>
<td>Coupled-cluster singles and doubles (CCSD) and perturbative connected triples via TCE</td>
</tr>
<tr>
<td>nwc-ccsd(2)_t</td>
<td>CCSD and perturbative CCSD(T)\textsubscript{T} correction via TCE</td>
</tr>
<tr>
<td>nwc-ccsd(2)_tq</td>
<td>CCSD and perturbative CCSD(2) correction via TCE</td>
</tr>
<tr>
<td>nwc-ccsdt(2)_q</td>
<td>Coupled-cluster single, doubles and triples (CCSDT) and perturbative CCSDT(2)\textsubscript{Q} correction via TCE</td>
</tr>
<tr>
<td>nwc-cr-ccsd[t]</td>
<td>Completely renormalized CCSD[T] method via TCE</td>
</tr>
<tr>
<td>nwc-cr-ccsd(t)</td>
<td>Completely renormalized CCSD(T) method via TCE</td>
</tr>
<tr>
<td>nwc-lccsd</td>
<td>Linearized coupled-cluster singles and doubles via TCE</td>
</tr>
<tr>
<td>nwc-lccd</td>
<td>Linearized coupled-cluster doubles via TCE</td>
</tr>
</tbody>
</table>
### Table C-1 Continued

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nwc-lr-ccsd</td>
<td>Locally renormalized equation-of-motion coupled-cluster singles and doubles (EOMCCSD) via TCE</td>
</tr>
<tr>
<td>nwc-lr-ccsd(t)</td>
<td>CCSD and perturbative locally renormalized CCSD(T) correction via TCE</td>
</tr>
<tr>
<td>nwc-lr-ccsd(tq)-1</td>
<td>CCSD and perturbative locally renormalized CCSD(TQ) correction via TCE</td>
</tr>
</tbody>
</table>

Users of QCDB are also able to access the litany of property options in NWChem, both outside and inside of the TCE module. Relativistic effects are also accessible.

---

1: Lee, J. All-in-one driver: What do groups 11 and 16 have in common? Iowa State University, Graduate Theses and Dissertations, 2018.