An Exploration of Molecular Mechanics and Quantum Chemical Methods

Masood Malekghassemi

June 9, 2009

Abstract

Computational chemistry is no new subject however, the idea of taking the process from including a human being in the loop between ab initio methods and general rule making for molecular mechanics calculations to excluding that same being via AI seems to be unimplemented. This project revolves around the idea that molecular mechanics is an incredibly fast and viable way of calculating the geometry of molecules and that as such, given the proper parameterizations and generalities, can be as accurate as or more accurate than ab initio methods.

However, due to a large number of setbacks including but not limited to: lack of any knowledge of ab initio methods at the start of the project, five restarts/refactors on the molecular mechanics portion of the project due to code smell, inaccessible resources (SpringerLink esp.), and resources I never exploited (teachers); I never finished the project this year. I will continue to work on it and hopefully you will find a rewritten version of this paper coming from elsewhere.

Chapter 1

Introduction

When one thinks of chemistry one normally thinks of people in lab coats with beakers of liquids being mixed together in exact amounts: 0.4 molar of this, 2 molar of that... However, a relatively 'recent' form of chemistry has come up called computational chemistry. Instead of putting potentially expensive chemicals together physically to examine reactions and gather energies and understand geometries of molecules, one can take a desired arrangement of atoms, plug it into a computer, and see what happens. Such programs exist, namely 'GAMESS', 'Gaussian', 'NWChem', 'AMBRE', etc. However, these programs are essentially black boxes to the chemist. I wanted to delve deeper into their inner workings to try to understand what exactly it was that they were doing, how it was that they were understanding the basis sets given to them, how they manipulated wavefunctions, how they found energies, how they determined geometries, and how they performed the subroutines required for that which was said above. The final product for this project would then be a conglomerate of different quantum mechanical (ab initio) methods and molecular mechanical methods put together to automate parameterization of molecular mechanical simulations (which, considering the great lengths chemists can go through to match their parameterizations to certain forms of experiment, is a difficult task indeed).

Chapter 2

Background

2.1 Shallow Background

Computational chemistry is a form of chemistry that takes physical rules and applies them to virtual models in a virtual space. The chemist enters a geometry and queries the computer with regards to its various conformational energies and electron densities to try to understand how it is that the molecule behaves. While experiment may be the final decision on what happens or doesn't happen when you mix chemical A with chemical B, it sometimes leaves the chemist in the dark with regards to what exactly happens during the reaction. Example questions may be: How is it that chemical A reacts with chemical B? Where do the two have to hit for the reaction to occur? How can I use this information to better engineer the reaction? Etc. Etc.

Using computational simulations, a chemist can slow down the reactions to nanosecond long time steps (even shorter if he/she wants to). The chemist can determine the theoretical existence of a chemical that may be helpful for performing a certain task. The chemist can do preliminary steps to see whether or not performing actual experiment may be beneficial, irregardless of how expensive the requisite chemicals may be.

Now, there are three different categories of computational chemistry separated based on how far removed they are from the fundamentals. The 'abinitio' methods start from first principles and calculate the inner workings of a system. The 'semiempirical' methods still work from first principles, but take in extra information from empirical studies and let go of certain calculations to make computation easier (such as scrapping all electrons but those in the valence shell from consideration). The 'empirical' methods work directly from empirical data, working off of a so called 'ball and spring' model (each bond is a spring parameterized by experiment).

2.2 Deeper Background

The following is a crash course with many holes regarding what I'm doing. The point is not to teach you but to show you just how weird this stuff is.

The Schrodinger equation is the basis for how computational chemistry is performed without regards to relativity (for which the mechanics become much more complicated and nigh impossible [at least for me] for one to understand in the space of a few months of on and off self-teaching). In its pseudo-eigenvalue time-independent form, it is:

$$H\psi = E\psi \quad (1)$$

Harmless looking enough, it is impossible to solve this equation for ψ (the function that represents the entirety of the system) when there are greater than two interacting bodies involved, namely more than two protons, electrons, or combinations of both. The H is the 'Hamiltonian' operator which when applied to a wavefunction ψ scales ψ by its energy under the effect of the given Hamiltonian. In the case of molecules, the generally used Hamiltonian pulls out the nuclear repulsion energy, the electronic repulson energy, the

electron exchange energy (a difficult item to explain), the electron-nuclear attraction energy, and adds it all together to give the total energy of the system.

To solve this equation, we can use computers to, instead of analytically solving it exactly (which again, is impossible for greater than two bodies, ie. anything more complicated than mono-electronic Hydrogen), analytically and numerically solve for it approximately. This can be done in a few methods, but all (seem to) derive themselves from the variational principle and perturbation theory. Also, they all share the 'Born-Oppenheimer approximation', that is to say that they all 'freeze' the nuclei in place in a molecule to simplify the calculations - allowing the protons to interact with the electrons causes major mathematical headaches and pains.

Starting with the variational theory... We take the equation above and pre-multiply both sides by $\psi^\dagger$, and integrate (the multiplication is to avoid nasty infinities; again, this is a crash course - really it's just meant for you to have an idea of just how much I had to dig to get to this stuff).

$$\langle\psi|H|\psi\rangle = \langle\psi|E|\psi\rangle \quad (2)$$

$$E = \frac{\langle\psi|H|\psi\rangle}{\langle\psi|\psi\rangle} \quad (3)$$

From here, if we can assume that, using molecular orbital theory, the whole wavefunction is a linear combination of atomic mono-electronic wavefunctions (spatially, 'orbitals' [which can also refer to pairs of electrons]), we can then take ψ and turn it into a summation $\Sigma c_i \phi_i$. Given that we already know the atomic wavefunctions (solved for by previous generations of computational chemists), we can solve for molecular wavefunctions by differentiating the left hand side of (3) with respect to every c_i, find where there are minimums, and then use those as our weighted value for that atomic wavefunction contributing to the whole molecular wavefunction. This allows us to solve for the expected lowest energy state (the most likely electronic conformation of the molecule) given a certain nuclear arrangement. And that's the variational theorem in a highly condensed and slightly cracked nutshell.

Perturbation theory is slightly easier to comprehend. The idea is that, given a system that you know how to solve, and given a system that is impossible to solve but is *similar* to that which you can solve, you can add a 'perturbation' term to the mathematics involved in the solvable problem to solve approximately yet accurately the impossible to solve system. The simplest example of this I can think of that is applicable is the Hamiltonian for a Hydrogen atom versus the Hamiltonian for a Helium atom.

Hydrogen Hamiltonian: $H = \bigtriangledown^2 + \frac{1}{r_1}$ (4) Helium Hamiltonian: $H = \bigtriangledown^2 + \frac{1}{r_1} + \frac{1}{r_2} + \frac{1}{r_{1,2}}$ (5)

In the case above, the Helium atom has a hamiltonian that is the equivalent of the Hydrogen's Hamiltonian when considering each electron individually. The added electron repulsion term, the $\frac{1}{r_{1,2}}$, is the 'perturbation' term, the term that represents our movement from a simple system to a slightly more complex system.

2.3 Methods used

The methods mentioned above can then be used to derive, essentially, the remainder of the computational chemistry methods based in quantum mechanics. The 'simplest' of fully ab-initio methods is the Hartree-Fock method, which basically takes the wavefunction and makes incremental improvements to the individual weightings of atomic wavefunctions to determine the ground state (or excited states) of a molecule until a 'self consistent field' is generated. A crude analogy to computer science would be to say that it's like setting a hill climbing algorithm loose nearby a local point of interest (say, a minimum in energy). The 'simplest' of semi-empirical methods, the Extended Huckel method, doesn't require a self-consistent field because of the way it's set up. No incremental approach towards the proper wavefunction can be made.

There are plenty of ab-initio methods, such as density-functional theory, Hartree-Fock, Moller-

Plesset Perturbation Theory, Configuration Interaction (various methods are a subset of this), etc. There are yet more semi-empirical methods: Fenske-Hall, Simple Huckel Method, Extended Huckel Method, Pariser-Parr-Pople, Austin-Model 1, CNDO, INDO, NDDO, etc. The ball and spring model, Molecular Mechanics, is so heavily parameterized by experiment that one could consider each separate force-field (a parameterization of the model, and there are a lot of them) to be its own method. In short, there is a large variety of methods, and each of them is complex in its own right with regards to implementation.

2.4 The Project

The project will definitely not attempt to implement every single one of the methods mentioned above. It will, however, at least attempt to implement that which is common to the vast majority of the quantum mechanical methods and a generalized form of the molecular mechanical methods as well. The requirements for solving for the approximate wavefunction in most ab-initio and semi-empirical methods are shared. With regards to an equation, the Roothan equation (used in the Restricted-Hartree-Fock method) sets up the requirements visibly:

$$FC = SC\epsilon \quad (6)$$

Where F is the Fock matrix (essentially the set of integrals representing the energetic interactions between electrons), C is the 'coefficient' matrix (representing the weights for our linear combinations of atomic orbitals), and S is the matrix of overlaps between orbitals. Obviously (for me, probably not for you - this is a crash course, remember?) I need to be able to take the integrals required by the mono-electronic Hamiltonian across all space (negative infinity to infinity) and the overlap integrals across all space (again from negative infinity to infinity). This requires a lot of research and understanding of the derivations (both of which have been occupying me for the longest time).

Chapter 3

Methodology

3.1 Integration

To perform the integrations in the most general way possible, I needed a large number of basic and advanced mathematics structures capable of handling manipulation of polynomials and integration of cartesian gaussian functions. In order to take an overlap integral over all space, for example, one needs to be able to perform what is essentially voodoo magic on functions:

$$\int \cdots \int_{-\infty}^{\infty} [e^{-\alpha r^2} \Pi_i^n x_i dx_i] * [e^{-\alpha r'^2} \Pi_i^{n'} x'_i dx'_i] \quad (7)$$

Where all of the 'x's with the apostrophe are translated (as in, mathematically shifted) versions of those without the apostrophe. To do this requires a decent amount of manipulation of the monomial terms $\Pi_i^n x_i$ and $\Pi_i^{n'} x'_i$. First, one has to know where the center of the product of the gaussian functions, the $e^{-\alpha r^2}$ and $e^{-\alpha r'^2}$, are going to end up (by using the Gaussian product rule), then one has to translate all of the polynomials to coordinates at that point, and then one has to multiply out the polynomials, and then convert them back all into a summation of individual monomial terms with a totally new Gaussian function tacked on to the end, and *then* take the necessitated integral using the Gamma function identity:

$$\int_0^{\infty} x^{z-1} e^{-x} dx = \Gamma(z) \quad (8)$$

In summary, it's a total pain. Regardless, the code properly performs the integrals.

3.2 Matrix Diagonalization

This project also requires the diagonalization of matrices to find the pseudo-eigenvalues of the Hamiltonian once it is converted to matrix form. Because the Fock matrix is symmetric in form, the diagonalization process used may be made as a set of orthogonal transformations made on the symmetric matrix. This is called Jacobi diagonalization.

3.3 Programming Methodology

While this may not be of much interest, I have used multiple kinds of programming techniques specific to C++ to aid in making my code as extensible as possible. While this led to numerous restarts of the project's code base due to code smell (as in, the code started to look too complicated to seriously have been the correct and proper solution, thus I scrapped it), I eventually came to an understanding that no matter what I do, I'm going to be unhappy with what I've done because of trade offs. Thus, in the recent (and final) days of this project, I've made the most progress I've made in the past year. Anyway, continuing...

All classes I've created are templated for their scalar types and natural types and the vast majority of them have access to type traits classes (templated classes that utilize the curiously recurring template pattern in C++ to define static type attributes related to the class they are being templated for) to determine at compile time what it is

that I want them to do with whatever types it is I've given to them. As an example, I've templated my Factorial class to behave differently when given a type that has a floating point as specified by its type traits than when given a type that is of the unsigned integer type as specified again by its type traits. Thus, code of the form:

```
Factorial<unsigned>::evaluate(3)
```

will evaluate to a more numerically exact value than

```
Factorial<double>::evaluate(3.0)
```

because it can take advantage of the constraints set by the argument's type without any time consuming dynamic run time checking.

There is also, then, my extensive use of what I call variadic argument iterators. This is a kind of iterator that I have not seen anywhere else in anyone else's code, which is surprising to me because of just how useful it is, despite the shortcomings of stability and the requirement of great care when passing it as an argument. The basic idea is that in vanilla C, there're constructs called 'variadic functions' which can take varying numbers of arguments. The way one grabs the arguments is by using standard library provided macros, which are unfriendly when it comes to passing variables around. Some example code:

```
int fcn(int num, ...)
{
    unsigned tot=0;
    va_list ap;
    va_start(ap, num);
    for(unsigned j=0; j<num; j++)
        tot+=va_arg(ap, unsigned);
    va_end(ap);
    return tot;
}
```

This particular example shows a va_list object being created. However, considering that the vast majority of functions that can take varying numbers of arguments are generalized to use iterators, the use of variadic arguments have taken a back seat, despite their aesthetic appeal in code (and I'm all about making my code pretty, else I restart).

A solution to change the variadic macro arguments into an iterator suitable for passing to iterator based functions is not too difficult to construct. An example follows (note: it is missing the copy functionality necessary to maintain stability of the iterator across reassignments):

```
template<typename scalar_t>
struct VariableArgsIterator
{
    VariableArgsIterator(scalar_t arg0, va_list* args)
        : ap(args), arg(arg0) {}
    va_list* ap;
    scalar_t arg;
    scalar_t operator*()
    { return arg; }
    scalar_t operator++()
    { arg = va_arg(*ap, scalar_t); }
    scalar_t operator++(int)
    { arg = va_arg(*ap, scalar_t); }
};
int fcn(int num, ...)
{
    va_list ap;
    va_start(ap, x0);
    VariableArgsIterator<scalar_t>
        variable_args_iter(x0, &ap);
    scalar_t args[dimension];
    copy_n(variable_args_iter,
        dimension, args);
    scalar_t ret = evaluate(args);
    va_end(ap);
    return ret;
}
```

Where 'copy_n' is a function that takes an iterator argument. This particular transformation is suitable for generalizing a class that can be templated to take varying numbers of arguments (such as classes representing monomials; one function argument per variable involved in the monomial) in a particular function without having to respecialize whatever functions that class uses to accept variadic structure pointers.

Chapter 4

Testing

I have been testing the individual components of my program as they come around. The polynomials in particular were tested thoroughly to ensure that there was nothing fishy going on with them, especially for expansion of a variable into another polynomial and polynomial formation from shifting the variables of a base polynomial (necessary for calculating overlap as stated in 'Methodology'). The various other sections of my math library with analogs in the Boost Library were tested against it using random intervals and random point choices (far more reliable than definitive point choices, but also nigh impossible to tabulate).

However, testing for the diagonalization of a matrix was a slightly different task requiring a test matrix. An example matrix and its diagonalization is shown below:

4.1 Expected Results

Due to the lack of an actual program, I cannot have any 'expected results'.

4.2 Visualization

Due to the lack of an actual program, I cannot have any 'visualizations', save for simplistic two-dimensional diagonstic images of orbitals, e.g.:

```
-- Pre-jacobi call --A:
        2.000   0.000   0.000   0.000   -1.000
        0.000   1.000   0.000   -2.000  0.000
        0.000   0.000   3.000   0.000   0.000
        0.000   -2.000  0.000   4.000   0.000
        -1.000  0.000   0.000   0.000   5.000
P:
        1.000   0.000   0.000   0.000   0.000
        0.000   1.000   0.000   0.000   0.000
        0.000   0.000   1.000   0.000   0.000
        0.000   0.000   0.000   1.000   0.000
        0.000   0.000   0.000   0.000   1.000
-- Post-jacobi call --
A:
        1.697   0.000   0.000   0.000   0.000
        0.000   0.000   0.000   0.000   0.000
        0.000   0.000   3.000   0.000   0.000
        0.000   0.000   0.000   5.000   0.000
        0.000   0.000   0.000   0.000   5.303
P:
        0.957   0.000   0.000   0.000   -0.290
        0.000   0.894   0.000   -0.447  0.000
        0.000   0.000   1.000   0.000   0.000
        0.000   0.447   0.000   0.894   0.000
        0.290   0.000   0.000   0.000   0.957
P_transposed:
        0.957   0.000   0.000   0.000   -0.290
        0.000   0.894   0.000   -0.447  0.000
        0.000   0.000   1.000   0.000   0.000
        0.000   0.447   0.000   0.894   0.000
        0.290   0.000   0.000   0.000   0.957
P * D * P^-1:
        2.000   0.000   0.000   0.000   -1.000
        0.000   1.000   0.000   -2.000  0.000
        0.000   0.000   3.000   0.000   0.000
        0.000   -2.000  0.000   4.000   0.000
        -1.000  0.000   0.000   0.000   5.000
Successfully tested JacobiEigenvalueAlgorithm
```

Figure 4.1: Test on symmetric matrix of Jacobi diagonalization procedure

Figure 4.2: Principle energy level equal to 1

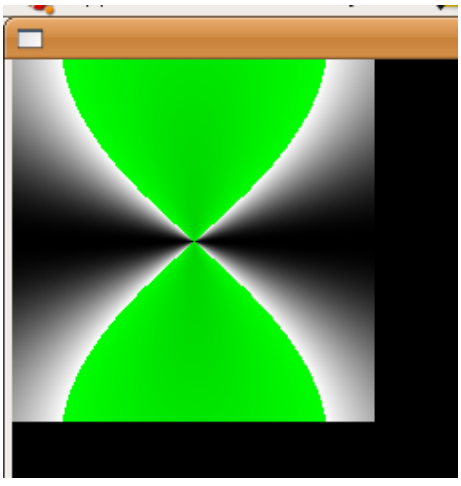

Figure 4.3: Principle energy level equal to 2

Chapter 5

Future Direction

Seeing as how this project is nowhere near completion and my pretentiousness precedes me on this project, I am obligated to continue this project to some form of finish. The list of items on my vast todo list include finding access to documentation to match mathematical form to what I can imply of the XML format used by the EMSL Basis Set Exchange, such that I may actually have useful radial wavefunction components. I also need to properly construct the methods used by the ab initio quantum chemical procedures and I should decide with finality on the actual structure of my molecular mechanical model. This act of *deciding* is crucial, simply because I essentially redid the molecular mechanics part over five times, wasting crucial months I could have spent on understanding quantum mechanics. But that's the past, continuing on about the future...

Beyond getting access to radial wavefunctions, I also need to understand the derivations used for the various integrals in the quantum mechanical methods. The greatest of my concerns is the coulomb integral (which is a four center integral - i.e. ridiculously complex), and at the current time, none of the derivations make much sense (expansion of $\frac{1}{r}$ in terms of the spherical harmonics? What?). Also, there is the issue of having some sort of visualization to validate my efforts on the project. That will have to come as I continue to work on this project past the end of the school year as well...

Bibliography

[1] Basis Set Exchange: A Community Database for Computational Sciences Schuchardt, K.L., Didier, B.T., Elsethagen, T., Sun, L., Gurumoorthi, V., Chase, J., Li, J., and Windus, T.L. J. Chem. Inf. Model., 47(3), 1045-1052, 2007, doi:10.1021/ci600510j.

[2] Bohm, D. (1951). Quantum Theory. New York: Dover.

[3] Cramer, Christopher J. (2004). Essentials of computational chemistry: theories and models. John Wiley and Sons.

[4] Guseinov, I. 1970. J. Phys. B: Atom. Molec. Phys. 3. 1399-412.

[5] Kuang, Jiyun and C D Lin. J. Phys. B: At. Mol. Opt. Phys. 30 (1997) 25292548.

[6] Lewars, E. (2003). Computational Chemistry: Introduction to the Theory and Applications of Molecular and Quantum Mechanics. Boston: Kluwer Academic Publishers.

[7] Nemes, Gergo. Ed.S.Sykora, Vol.II. Received September 24, 2007.

[8] Pilar, F. L. (2001). Elementary Quantum Chemistry (2nd ed.). Mineola, New York: Dover. (Original work published 1968)

[9] Rasch, J. and A. C. H. Yu. SIAM J. SCI. COMPUT. Vol. 25, No. 4, pp. 14161428

[10] Rico, J. F. and R. Lpez and G. Ramrez. Journal of Molecular Structure: THEOCHEM. Volume 537, Issues 1-3, 12 March 2001, Pages 27-40

[11] Schlegel, H. Bernard and Michael J. Frisch. Int. J. Q. Chem. Vol. 54, pp. 83-87.

[12] Stockis, Armel and Roald Hoffmann. J. Am. Chem. Soc., 1980, 102 (9), pp 29522962

[13] Yasuda, Koji. Wiley InterScience (2007). DOI 10.1002/jcc.20779.