FFDev: Progress Towards the Generation of *ab initio* Force Fields

by
Joshua Paul Radke
B.A., University of Minnesota, 1994

A thesis submitted to the Faculty of the
Graduate School of the
University of Colorado in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Chemistry
2002

This thesis entitled:
FFDev: Progress Towards the Generation of *ab initio* Force Fields
written by Joshua Paul Radke
has been approved for the Department of Chemistry

_____
David M. Walba

_____
Matthew A. Glaser

May 31st, 2002

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline

Radke, Joshua Paul (Ph.D. Chemistry)

FFDev: Progress Towards the Generation of *ab initio* Force Fields

Thesis directed by Professor David M. Walba

Classical interaction potentials, or force fields, are the fundamental input for any molecular simulation. Currently available force fields suffer from several limitations; namely availability, appropriateness, and quality. Low quality interaction potentials necessarily give low quality results when used in molecular simulations. The current state of force field development lies in the hands of a few specialists. Users of existing force fields are required to either purchase software, or implement their own software to use them. Also, if a user wants an improved force field, they are required to either start their own research program for that purpose, or wait for an update to an existing force field to be published. Furthermore, the procedures used to derive parameters for existing force fields (whether they are semi-empirical, or better yet, based on *ab initio* data) are often poorly documented. We have developed a suite of software that serves as a foundation for the on demand creation of strictly appropriate custom force fields from *ab initio* data. As the parameterization is automated, the element of human error/subjectivism in the many required transcription and decisions steps is eliminated. Further, every non-topologically/stereochemically equivalent atom has its own atom type and parameters. By employing software to do the force field creation from scratch, we have also created the opportunity for routine improvement of force fields by modifying the method of extraction of *ab initio* data, decreasing (and hopefully, eventually eliminating) reliance on experimental data. We believe that the best parameters for any classical interaction potential must come from *ab initio* data, and that our approach will eventually allow researchers to have access to free, fundamentally sound, appropriate, and highly accurate force fields.

Acknowledgements

This work was made possible by the generous support provided by NSF MRSEC Grant DMR 98-0555. More importantly, however, are the people who contributed directly to the completion of this thesis. Dr. David Walba, for his patience, and giving me the time to find a project that really tickled my fancy; Dr. Matthew Glaser, for the endless hours of brainstorming, direction, and instruction; but most of all for his patience with my (occasionally headstrong) demeanor. I would also like to thank Dr. Edgardo Garciá for his scientific and personal contributions.

For any (seemingly endless) journey, there are literally hundreds of people encountered along the way, and I would also like to thank them, especially any I miss in the rest of these acknowledgements.

Family is the most imporant support element one can have, and I would like to thank some of my family. First and foremost, I would like to thank my God for listening to my prayers over the years, which must have sounded like babbling to him. In close second comes my wife Stefani, who has been insanely patient with my continuously babbling about ideas no 'normal' people would care to hear about. I would also like to thank my father, Skip, for teaching me the curiosity to pursue a PhD, and my mother, Sherry, for who I am today. I'd also like to thank my brother, Adam, for all of the great Minnesota Vikings games Sundays. Thank you also to Leigh and Dana. And to the coolest in-laws on the planet, Calvin, Linda, Scott and Vickie, Rita, Bill, Patrick (Boogaman), Ricky, Scotty, and Cody.

Professionally, this project would not have been possible without the visionary movement started by Richard Stallman; open source software. Since the inception of the Free Software Foundation, there have been numerous pioneers, who have made open, useful, software available to all, not just the wealthy. In particular, Linus Torvalds, Eric Raymond, Larry Wall, Bob Young, Kirk McKusick, and Tim O'Reilly.

My family back in Minnesota has been there for me every year (except the present one) for opening of fishing season, a grand time of fellowship. I'd like to thank Valerie, Wayne, Tim (yes, I'm done), David, Joe, and John; Butch, Ron, Tina, Amanda, and their families; and of course, Rossie. They're all Herschbachs, except for the Radkes that come from afar, Bud, Donna, Kurt, Mark, and Julie. Thank you also Rusty and family.

I have a couple of things I do to help preserve my sanity. The most important of these has been teaching. I would like to thank the Minority Arts and Sciences program at the University of Colorado. Thank you also Alphonse, Angela, and Wendy. The real people who deserve my thanks are the students, who have given me more than I could give them in a lifetime. I can't name you all, but thank you.

All of my co-workers during my time here have been helpful in some capacity. I'd like to thank Uwe, Forrest, Dan, Craig and Loretta (the wovewy), Bruce and

Valerie, Eva and Bill, Ken and Tracy, Jen and Brian, Lei and Phong, Lixing, Matt and Darcie, Ethan and Jen, Alan, and Tim (for most of the illustrations in this thesis, as well as for his trivia savvy).

And finally, I'd like to thank all of my oldest friends, for believing I could do this, Patricia and Phillip (and family), Joe and James Waldo, Jim and Sue Hood, John and Brenda Fowler, Jon Ryan, Sean Flynn, Jim Miller, Brian Jarvis, Snuffleupagus (do I win, Will?), Brent, Steve, Peter Tielemann (for the fun time, and the work talk), Murray and Alison, Dennis, Iotis, Sarah, Scott, Erik, Katrina, Jeannie, Jennicam Jen (no socks?), Jen, Jen (thanks for the car!), Otis and Felix, Marsha, Mark, Summer, Siri, Jeff, Laura, Chris, and the rest of the Fargo crew. And of course, who can forget the value and support Aluminum siding provides for all of us.

Thank you, all of you who have been a part of my life, mentioned or unmentioned you have my deepest gratitude for sharing my life with me.

Table of Contents

Table of Figures

Chapter 1

Introduction

Every great endeavor begins with a story. In this case, it turns out that the eventual goal was much different than what was really done. At the inception of this project, I was asked to do a 'single molecule in a binding site' calculation. Upon studying the problem, it quickly became apparent that actually completing this goal would be a long process, and the expertise gained would be only applicable to the person who actually did all of the work. I wanted a 'permanent' solution to this

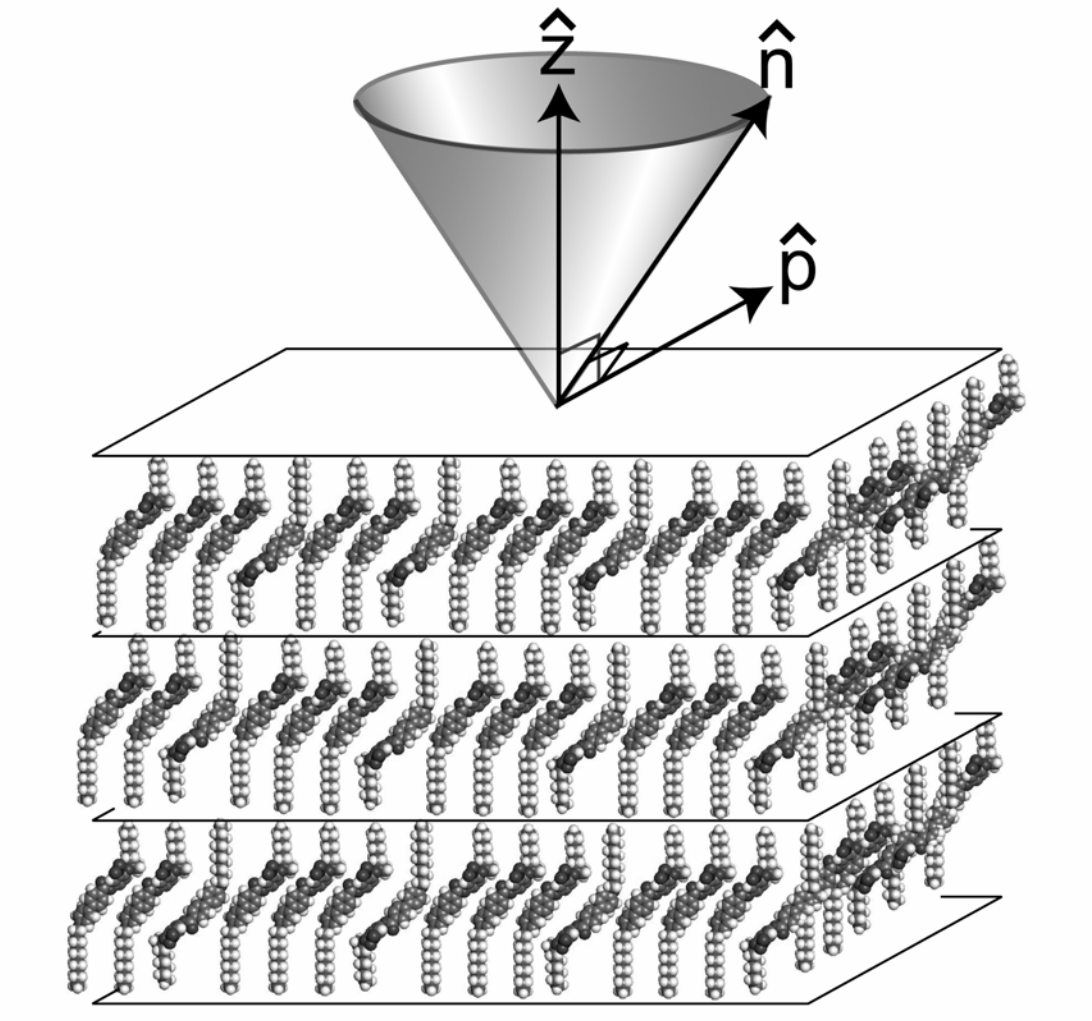


Figure 1: The structure of the Smectic C phase. The cone at the top of the figure demonstrates the important directors of the Smectic C phase. The vectors **n** and **z** define the tilt plane, which is perpendicular to the polar axis **p** (in the case of a Smectic C* phase).

problem, and so began FFDev.

The Simple Background

Liquid crystals are molecules that organize themselves in such a way that they have properties of both liquids, and crystals. They are truly liquids in that they flow, and take the shape of their container, but they also display some degree of long range positional or orientational order (but never enough to be identified as crystalline solids). This work focuses only on molecules within the smectic C phase (**Figure 1**).

The Boulder model [1] for polarization is really quite simple, and doesn't necessarily even require a computer to apply. Empirically, it was discovered very early [2] that for molecules within the smectic C phase, the flexible tails are always more tilted than the rigid cores. This simple fact implies that molecules within a smectic C phase prefer to be (or are at their lowest free energy) in conformations and orientations that fit well within the Boulder model binding site (**Figure 3**). The shape of the Boulder model binding site represents the effect of the Smectic C phase on a single molecule. One can easily apply this model by doing a bit of simple drawing, as shown in **Figure 2**

If one hopes to get quantitative, as opposed to qualitative, information from this model, they must convert this simple idea to an algorithmic basis. Maier-Saupe mean field theory [3] provides the perfect model for doing so. By docking the molecules into the Boulder model binding site, we are saying that there is some energy cost for deviating from conformations that fit well within that binding site. The true source of these energy costs need not be established, as we say they result from the sum of inter-molecular forces within the Smectic C phase.

The distribution function of a molecule is defined as the 'population', or probability, of every possible molecular configuration [4]. While the true distribution function of a molecule (at a given temperature and pressure) can never be found exactly, except for the simplest of cases, some methods exist to allow us to get very good approximations of this distribution function.

It is important to emphasize here that we are dealing with real liquids. A molecule in the Smectic C phase still has a great deal of conformational flexibility, and should never be thought of in the way that we typically think about crystalline solids.

In order to calculate this distribution function, we need to be able to evaluate the energy of all of the possible molecular configurations. Someday in the very distant future, we will be able to get 'arbitrarily exact' energies for all possible configurations, via *ab initio* calculations. Until that day comes, we instead must rely on a classical energy expression to evaluate these values. Our quantitative version of the Boulder model adds energy costs for deviations from the binding site geometry, as shown in **Figure 3** (specific details are presented in Chapter 2).

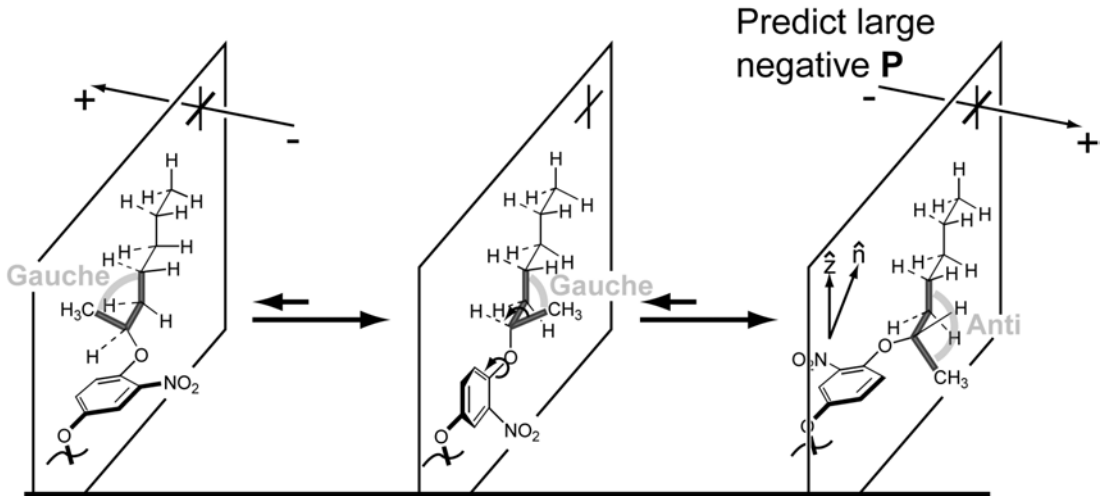


Figure 2: The "back of the envelope" Boulder model. Begin by placing the tails more tilted than the core (in an all trans configuration), and putting the molecule in a conformation that is intuitively low energy (center pane); then simply "crankshaft" around the indicated torsions to generate other geometries.

The Simple Motivation

The origins of the macroscopic polarization observed within smectic C* (the * simply means the phase is composed of chiral molecules) phases are one of the properties that we most understand. In 1974, Bob Meyer [5] gave a most elegant symmerty argument for why this macroscopic polarization should exist, but no work to date can accurately predict, *a priori*, what the experimental value of the macroscopic polarization should be for a given molecule. It's clear that the answer must exist, and that it involves the distribution function of a macroscopic sample, but once again that problem space is far too large to handle with both accuracy and precision (if we are challenged to find the distribution function of a single molecule, finding the distribution function for a macroscopic sample of liquid crystal molecules is completely out of our reach!)

Using the Boulder model to calculate the distribution function for a single molecule in this binding site allows us access to several numbers, including the overall (average) dipole moment of the molecule. The true macroscopic polarization is actually a polarization density (i.e. nanocoulombs/cm^2, or debye/cm^3), so we can take the dipole, and divide by the volume of the molecule, as derived from the bulk density of the liquid crystal. There is one further refinement necessary before we can report a polarization density, and that is to take the calculated dipole, and find the component of it along the 'true' polar axis, which is defined by the symmetry of the phase (**Figure 4**); we use this vector instead of the original dipole.

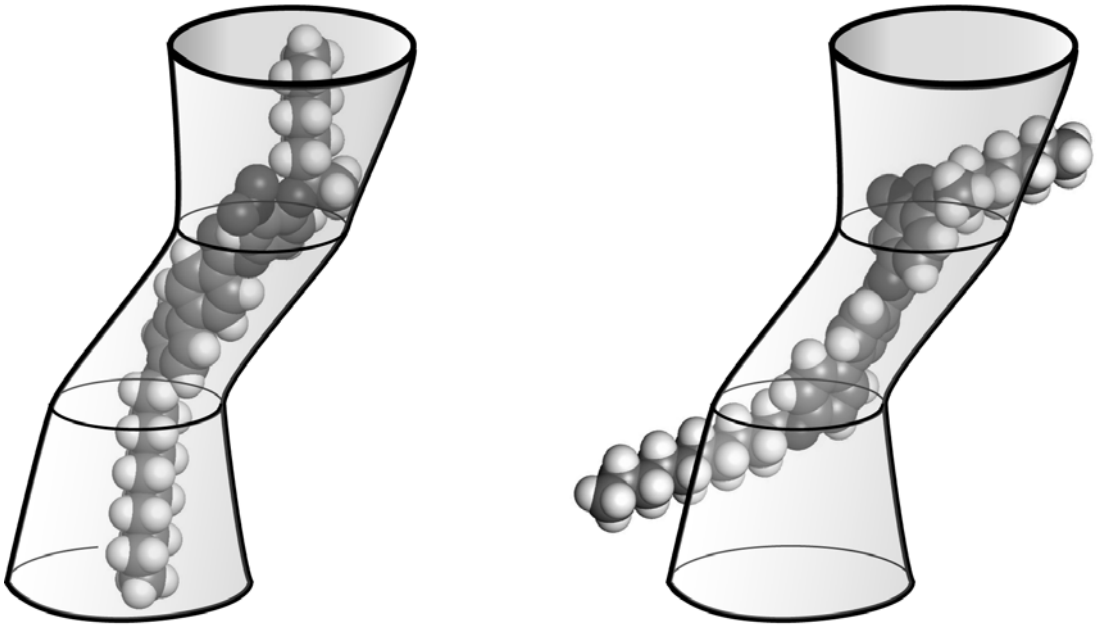


Figure 3: A typical molecule in two orientations in the Boulder model binding site. Two orientations are shown; in the left one, there would be no additional energy cost, in the right one, (with the same molecule simply rotated about its core axis), the molecule would suffer a significant energy penalty). The shape of the binding site simply comes from the influence of the rest of the phase upon a single molecule.

The real work

It turns out that the most difficult part of the previously outlined procedure is to get 'useful' force fields (interaction potentials) that correctly reproduce the shapes and energies of the various conformations in the distribution function. Many force fields already exist [6], but none of them diligently reproduce energies associated with the many dihedrals in any given liquid crystal. Not surprisingly, the Boulder model is very sensitive to molecular shapes, which are in turn very sensitive to these torsions.

For the aforementioned reasons, we are required to create our own force fields from *ab initio* data. Previous work at the University of Colorado had done exactly that [7], but again, the process was arduous and error-prone. We wanted to develop a software system to automate the majority of the process for many reasons. Firstly, it would greatly decrease the (human) time involved with creating one of our custom force fields. Secondly, it would remove the possibility of errors associated with transcription of values. Finally, it would algorithmically define procedures, and remove (or at the very least regularize) the many human decisions involved with the

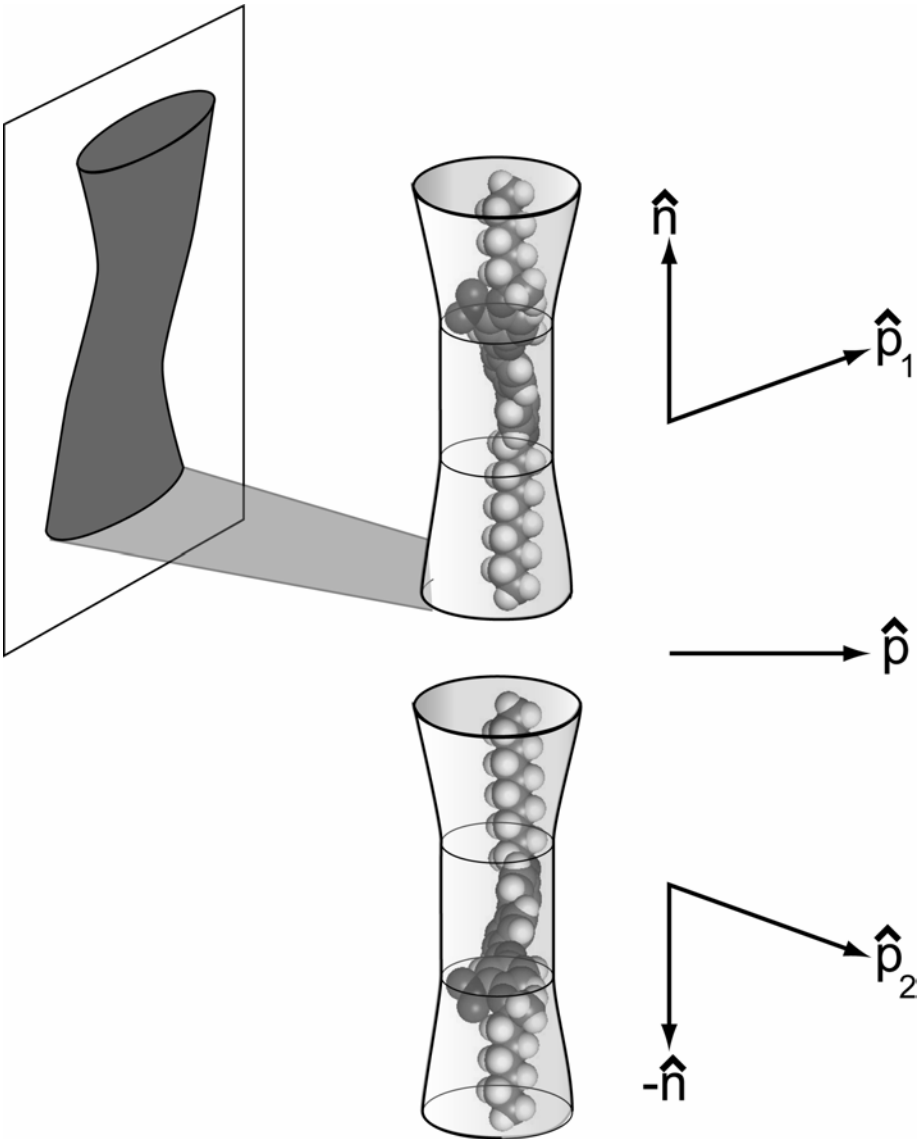


Figure 4: Illustrations of the tilt plane, which is perpendicular to the polar axis (top left), and the symmetry of the phase (right). In all cases of known calamitic liquid crystals, **n** goes to -**n**, meaning that there is no polar order along the long axis of liquid crystal molecules. In the case of molecules in the Smectic C phase, this rule manifests itself by enforcing a C_2 symmetry axis parallel to the polar axis. As a result, any component of polarization within the tilt plane in a binding site calculation will go to zero when the -**n** conformations are taken into account.

process. FFDev is the culmination of that effort, and the focus of the rest of this thesis.

The real background and motivation

The FFDev project endeavors to support and grow a relatively recent marriage in the world of physical chemistry. The fields of quantum chemistry and statistical mechanics have already begun to merge in the field of computational molecular mechanics, a marriage that promises profound affects in the very near future. Due to factors discussed shortly, the rate of progress in this field has been, and promises to continue to be, phenomenal.

Quantum chemistry

Quantum chemistry was first introduced by Heisenberg in 1925 [8, 9]. In the very same year, it was given a matrix-algebra formulation by Born and Jordan [10]. In 1926, Schrödinger independently introduced his wave mechanics formulation, proved the equivalence of the two methods, and established his name in history [11] (primarily due to the simpler mathematical formulation). While a firm footing to real solutions of atomic systems had been established, it became quickly apparent that systems with any real complexity were unsolvable. Dirac's famous quote set the stage for the future of quantum chemistry to date, he said [12]:

> "The underlying physical laws necessary for the mathematical theory of a large part of physics and the whole of chemistry are thus completely known, and the difficulty is only that the exact application of these laws leads to equations much too complicated to be soluble."

Dirac's realization marks the beginning of computational chemistry (many mark the beginning by Pople's early work [13], but the fundamental problem in my mind is reducing the complexity of problems to manageable sizes). In order to solve any problem of greater difficulty than a single hydrogen atom as the sole member of a universe, approximations need to be made. Computational quantum chemistry is the field of making those approximations, and applying the resulting methods to solve real world problems. Solutions to electronic and nuclear structure of molecules are called *ab initio* results, meaning they're derived from first principles, and rely on nothing more than the specification of the system in question, and certain universal constants. Two factors have been responsible for recent rapid advances in this field. Moore's law [14] states that "the number of transistors per square inch on integrated circuits doubles every 18 months". Transistor density quite closely translates to computational power/price (**Figure 5**, [15]).

I would have to say that the second major factor is the involvement of Industry and Academia in providing software to implement computational quantum chemical methods on modern computers. These two factors make is relatively easy to get high quality electronic and conformational structure information for moderate sized structures (on the order of 25 heavy atoms) in times ranging from several hours, to a maximum of a week, in most cases. The availability of solutions with such precision is one half of the fortunate timing that makes projects like FFDev possible.

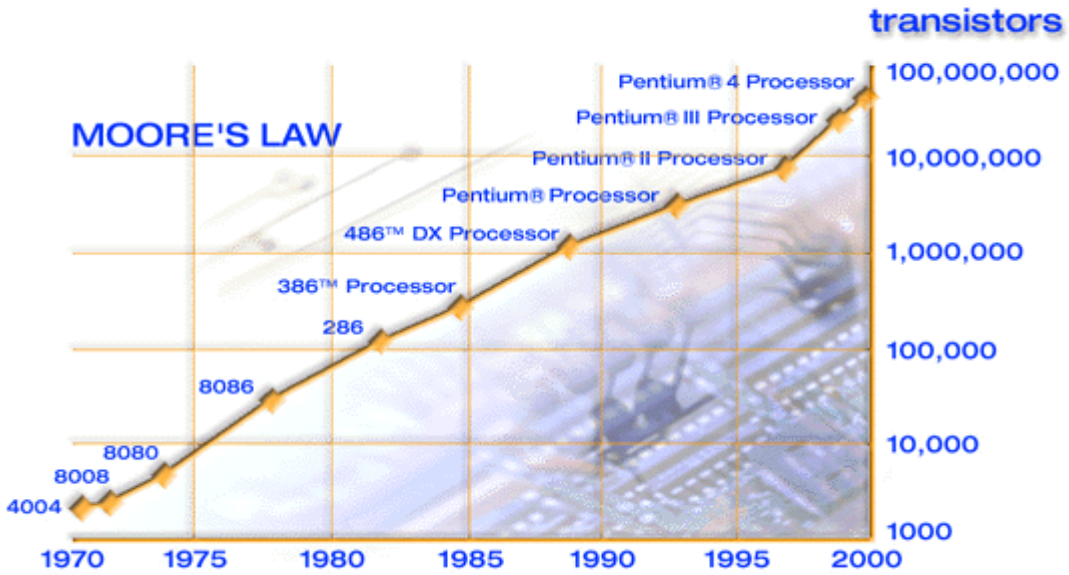


Figure 5: Moore's Law, as shown for the Intel x86 series of processors.

Statistical mechanics

If quantum chemistry embodies the 'genius of the twentieth century', the field of statistical mechanics embodies a philosophical journey though the ages. Much of the following account is taken from "Sketching the History of Statistical Mechanics and Thermodynamics" [16]. Democritus (470 to 360 BC) is frequently credited to be the 'father of atomism'. Atomism is the concept that at some level, the universe must be composed of indestructible, discrete units. While his philosophy was soon 'trampled' by the horde of Aristotelians to come, his fundamentally correct postulation of the nature of matter would serve as the foundation for statistical mechanics. Around 150 BC, Hero of Alexandria wrote "Pneumatics", a fascinating (and definitely recommended reading!) book on the behavior of fluids, including air [17]. It appears that a 1575 translation of "Pneumatics" to Latin may have been at least partially responsible for the explosion of understanding to happen within the next 100 years, which would eventually lead to various formulations of the ideal gas law by Boyle, Charles, Gay-Lussac, and Avogadro by the early 1800's. In 1843, Waterston [18] published a complete kinetic theory of gases, but was ignored, though he later tried to publish his work in journals as well. It wasn't until 1884 that Gibbs coined the term 'statistical mechanics' to refer to the study of thermodynamic properties of systems by the application of kinetic theory.

By this time, matter was treated as atoms, and classical physics was used to derive the properties of large systems, by assuming certain things about the behavior of atoms within these systems. The predictive value of the kinetic theory of gases and the ideal gas law (along with its associated variants, such as the van der Waals equation), are testament to the value of 'simple' classical models as powerful predictors of real phenomena. In the solution of all statistical mechanics problems, the single cohesive element is that the individual members of the system are given some behavior to govern their states, and the system is statistically analyzed, either in a time dependent, or time independent fashion. This analysis requires an integration or sum over all of the states. While some systems scale very well under this treatment (analytical expressions can be derived for any interesting property at arbitrary system sizes and/or timescales), the vast majority of conceivable problems do not, and so had been ignored until the advent of modern computers.

By the early 1950's, there was significant effort being put forth [19] in the academic community to use 'electronic computers' to solve statistical mechanics problems that had been completely out reach of statistical mechanics until the emergence of computers.

The marriage

There's no truly good way to draw the lines of when the marriage between quantum chemistry and statistical mechanics took place. As mentioned previously, computer simulations of systems of hard spheres were being done in the 50's. The earliest 'atomistic' simulation may mark this beginning just before the end of the decade, and was published in 1960 [20]. By the 1970's, computers were becoming powerful enough to treat systems of much greater complexity than simply collections of spheres. This marked the beginning of atomistic molecular mechanics as we know it today [21]. Despite the great variety of approaches to molecular modeling to date [6], a couple of key elements remain constant in all of the solutions.

Every force field must have some formula by which the energy of the system as a function of the positions of all of the members of the system can be evaluated. Ideally, that function will also have analytical derivatives of the energy with respect to the positions of each of the elements (this is necessary for expedience in time dependent simulations). A typical force field is simply a sum of terms, with each term providing the energy associated with a particular type of molecular feature. For every type of interaction in the entire system (i.e., for a bond comprised of two different types of atoms) we require parameters, or numbers that give information about that particular entity, such as bond length, and how 'strong' the bond is. The details of our implementation are beyond the scope of this thesis, though we'll provide an overview in Chapter 2; however, the input that goes into the force field is the prime focus.

For every force field known to the author, each atom in the system is assigned an 'atom type'. This is a descriptor whose purpose is to encapsulate all of the behavior of that atom in a variety of roles (i.e., atomic charge, as a member of a bond stretch or

angle, etc.). This is a very useful concept, and allows us to treat systems with large numbers of atoms with a reduced number of parameters. Despite its general usefulness, this particular approximation seemed inadequate for our purposes. As a partial solution to this problem, we have implemented a combination of our own descriptors (discussed in detail in Chapter 3) and our own stereochemical descriptors for tetrahedral stereogenic carbons as a way to assign unique identities to all atoms that are not topologically and stereochemically equivalent.

The process of providing all of the parameters for a given simulation is unsurprisingly called parameterization. Of the existing force fields, there are two sources of parameters. One type derives the parameters empirically, i.e., they seek a certain outcome of a molecular mechanics calculation by changing the parameters until the desired answer is achieved. The other source of parameters is from *ab initio* calculations. The majority of existing force fields use some combination of the two approaches, generating semi-empirical force fields. Increased reliance on *ab initio* data for parameterization of force fields has led to some very high quality force fields [22], and has improved quality altogether, yet the problem still remains that the product force field is either too specific to be generally useful, or too general to provide correct parameters for every term in the force field (the number of atom types is significantly less than the number of distinct atoms in the molecule for which the force field was generated).

So what is it that distinguishes our force fields from all of the others that are available? To our knowledge, we have the only system designed to generate force fields directly from *ab initio* data, as well as provide a unique identity (atom type) to each and every non-topologically/stereochemically equivalent atom. Our current progress does not allow us to extract all of the desired parameters from *ab initio* data, but it does allow us to get the ones we are particularly concerned with (energies about dihedrals), and it further serves as a "proof of concept" that such a direct mapping can be accomplished, and may indeed be done by future work on this project.

It seems obvious that the best possible classical interaction potentials must have a direct relationship with electronic structure, as provided by ab initio calculations. What the specifics of this relationship are, however, is not clear at all. In the absence of our approach, the only way to get 'better' force fields is to either change the form of your existing one (most frequently by adding coupling terms, but almost always one ends up adding more parameters), or try harder to change the parameters that go into it, in an effort to get more accurate results. We have added a third approach to rapid systematic improvement, and that is to refine one's method of generating the final force fields. This approach many advantages. Primarily, it allows for rapid prototyping and refinement of force fields (**Figure 6**). More specifically, it affords us the opportunity to do our refinement not only by changing the method of data abstraction, but also by changing either paramters or the form of the force field emprically, should we wish to. In short, we've opened the door for many more users to be involved with force field refinement. Finally, it promises to provide us with insight into the subtleties of why all classical interaction potentials fail, at some level.

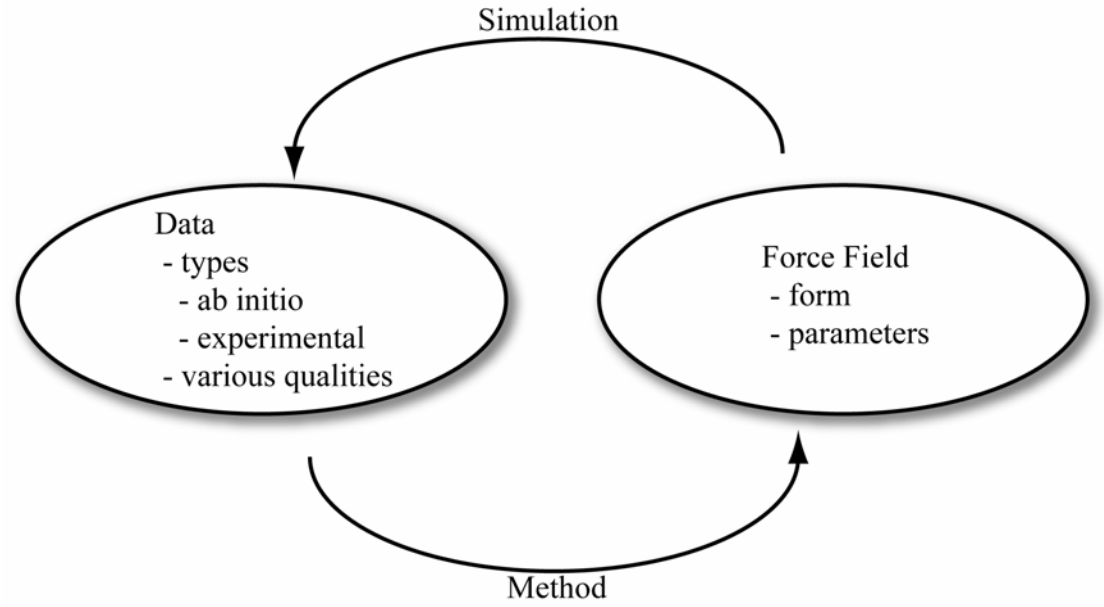


Figure 6: The real accomplisment. Creating and refining force fields is an iterative process. Current experts in the field have access to all of the tools to both run simulations with their force fields, and refine them using their own methods. Unfortunately, tools which automatically create force fields by well defined (and customizeable) methods have been largly unavailable until now.

A measure of success

No theory or model can be considered useful unless it is capable of reproducing (or better yet, predicting) experimental results. While the primary accomplishment is a "proof of concept", we still need to be able to verify its usefulness in a simulation. Since we are a liquid crystal group, and obviously supporters of the Boulder model, evaluation of our force fields within the context of that model seems the natural choice.

Van Gunsteren, et. al. [23] set out very clearly the elements of molecular mechanics simulations that must be considered, in order to validate the results. In that paper, he outlines five barriers to validation, and five basic requirements necessary to overcome those barriers. These are as follows:

1. A full description of the model and algorithms must be readily available.

2. A full description of the interaction function or force field must be readily available.

3. Simulation results must be shown as a function of simulation length.

4. The source code of the software must be able to be checked.

5. The set up of the simulations must be described in detail.

We have done our best to fulfill all of these criteria in presenting our results in Chapter 2. Classical interaction potentials were generated for a single test compound, and the aforementioned 'single molecule in a binding site' calculations were done. The results of these test cases clearly demonstrate that we have generated reasonably good agreement with both previous simulations, and experiment.

Chapter 2

The results

Three 'single molecule in a binding site' calculations were finished and the results of those calculations are presented here. Before presenting the results, however, we'll discuss a bit more background on the details of how the calculations were done. The single test compound will be named Compound 1 throughout the rest of the thesis. The structure of these molecules can be seen in **Figure 7**.

The exact procedure used to generate the force field for the test compound is the subject of Chapter 4, which is more or less a 'walkthrough' chapter. Here we summarize the details of the input to the force field, and how these *ab initio* values were mapped onto our classical interaction potential.

Since the input molecules for our simulations are too large to handle with reasonable detail with *ab intio* calculations (**Figure 7**), and we wanted to develop a system to be useful for any sized input molecule, we did calculations on smaller fragments. Appendix B shows which fragments were used for each of the test cases. The procecure for generating these fragments is called fragmentation, and will be used in several other places in this text.

All *ab initio* calculations were performed with Gaussian 98 [24]. For each of the fragments, first the geometry was optimized using Becke's three parameter Hybrid Functional Using the LYP correlation functional with closed shell restricted wave functions [25], with a 6-31g(d) basis set (RB3LYP/6-31g(d)). The energy of the optimized conformation was evaluated at the RB3LYP/6-311+g(2d,p) level. The terminology for the this procedure is to simply state that the energy was evaluated at the RB3LYP/6-311+g(2d,p)// RB3LYP/6-31g(d) level of theory.

For each of the dihedral vs. energy profiles we needed, at least 24 individual energies were calculated at the RB3LYP/6-31+g(d,p)//RB3LYP/6-31g(d) level of theory, which has been shown to give excellent results for its computational cost [26]. When optimizing the geometries of the fragments during the torsion scans, all dihedrals about certain bonds were frozen at their global energy minimum values. This is done to prevent dihedral vs. energy profiles which exhibit 'un-natural' asymmetry (based on the direction the profile is scanned), which frequently occur with unconstrained dihedrals. If a bond was between two sp^3 hybridized heavy (non-hydrogen) atoms, neither of the atoms was a terminal CH_3 group, and both of the atoms had no resonant (bond order 1.5) bonds, then all dihedrals about that bond were frozen.

For our purposes, we use fragments with hydrogens on sp^3 carbons absorbed into those carbons. The exact form of the force field that we use for our simulations is as follows [7]:

$U(\mathbf{r}^N) = U_{str} + U_{bend} + U_{tors} + U_{inv} + U_{vdw} + U_{coul}$, where the individual energy

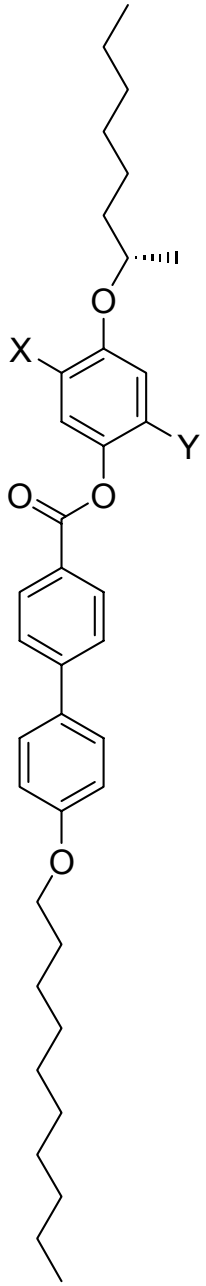


Figure 7: The compound for which force field was generated.

terms are defined as follows:

$$U_{str} = \sum_{\substack{bonds \\ ij}} \frac{1}{2} k_r \left(r_{ij} - r_{eq} \right)^2$$

$$U_{bend} = \sum_{\substack{angles \\ ijk}} \frac{1}{2} k_\theta \left(\theta_{ijk} - \theta_{eq} \right)^2$$

$$U_{tors} = \sum_{\substack{dihedrals \\ ijkl}} \sum_{n=0}^{6} c_{n\phi} \cos^n \phi_{ijkl}$$

$$U_{inv} = \sum_{\substack{umbrellas \\ ijkl}} k_\psi \left(\cos\psi_{eq} - \cos\psi_{ijkl} \right)^m$$

$$m = \begin{cases} 1, \psi_{eq} = 0 \\ 2, \psi_{eq} \neq 0 \end{cases}$$

$$U_{vdw} = \sum_{i<j} 4\varepsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r_{ij}} \right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}} \right)^{6} \right]$$

$$U_{coul} = \sum_{i>j} \frac{q_i q_j}{r_{ij}}$$

In the present work, all 1-2, 1-3, and 1-4 interactions are omitted in the evaluation of U_{vdw} and U_{coul}. The internal coordinates r_{ij}, θ_{ijk}, ϕ_{ijkl}, and ψ_{ijkl} are defined by:

$$r_{ij} = \left| \mathbf{r}_{ij} \right| = \left| \mathbf{r}_j - \mathbf{r}_i \right|$$

$$\theta_{ijk} = \cos^{-1} \left[- \frac{\mathbf{r}_{ij} \cdot \mathbf{r}_{jk}}{r_{ij} r_{jk}} \right]$$

$$\phi_{ijkl} = \cos^{-1} \left[\frac{\left(\mathbf{r}_{ij} \times \mathbf{r}_{jk} \right) \cdot \left(\mathbf{r}_{jk} \times \mathbf{r}_{kl} \right)}{\left| \left(\mathbf{r}_{ij} \times \mathbf{r}_{jk} \right) \right| \left| \left(\mathbf{r}_{jk} \times \mathbf{r}_{kl} \right) \right|} \right]$$

$$\psi_{ijkl} = \sin^{-1} \left[\frac{\mathbf{r}_{ij} \cdot \left(\mathbf{r}_{ik} \times \mathbf{r}_{il} \right)}{r_{ij} \left| \mathbf{r}_{ik} \times \mathbf{r}_{il} \right|} \right]$$

ψ_{ijkl} measures the angle between $\mathbf{r}_{ij}$ and the plane defined by $\mathbf{r}_{ik}$ and $\mathbf{r}_{il}$ for all three coordinate atoms i. The total inversion potential is taken to be the average of the umbrella torsion terms for the three possible choices of the special bond $\mathbf{r}_{ij}$. The definition of dihedral angle was as described by Kline and Prelog [27]. The total rotational potential about a bond is taken to be the average of all possible dihedrals about that bond. Both of these conventions were taken from the Dreiding II force field [28].

Parameters for bond stretching, and angle bending, were generic (Dreiding II [28]), though the equilibrium values for all bonds and angles were extracted from the global energy minimum conformation of the corresponding fragment. Generic inversion parameters were also used [28]. Point charges on the atoms were assigned based on the CHELPG scheme [29], and mapped onto the parent molecule from the relevant fragments. Carbons with absorbed hydrogens were assigned the sum of the charges of the carbon and all absorbed hydrogens. Van der Waals parameters were taken primarily from OPLS [30], though the values for the carbons containing absorbed hydrogens were taken from other sources [31].

This leaves only the parameters for the dihedrals to be determined. These parameters are determined in much the same way that they were in previous calculations of this type [7]. First, all torsional parameters are set such that no dihedral angle makes any energy contribution. Secondly, the torsion we are fitting is driven in exactly the same way as it was for the *ab initio* torsional potential (the same dihedral angles are driven, the same dihedrals are frozen), and the energy of the classical force field is recorded. Thirdly, for each dihedral angle, the difference between the classical energy and the *ab initio* energy is recorded. Finally, the $c_{n\phi}$ parameters are fitted to reproduce this energy profile.

In a separate verification step, the newly found parameters for the torsion are used (instead of all being set to 0), and the classical energy is evaluated by driving the system in the same was that it was in the generation step. Appendix A contains graphs comparing the *ab initio* vs. classical energy for every fitted torsion, as generated in this verification step, along with an illustration of which dihedral was driven. Appendix B shows the fragmentation of the test compound, as well as a graphical representation of which atoms and bonds were mapped from the child to the parent compound.

As mentioned in the introduction, the Boulder model binding site calculations require that we algorithmically implement the empirical fact that the tail is more tilted than the core. To do this in a molecular mechanics calculation, we require only three parameters. Firstly, we define which regions of the molecule are tail, and which regions are core. Secondly, we define an angle between the core director and the tail director (which can be parametrized from experimental data, if desired). Finally, we need to add an elogation potential. Since the single molecule simulation is done in a vacuum, and we only penalize the tail for not being parallel to the tail director,

omitting this elongation potential results in many conformations where the tail is folded, which is not in keeping with the Boulder Model.

To generate the molecular distribution function, we use a hybrid Monte Carlo scheme. Monte Carlo techniques generate the distribution function by evaluating the energy of trial states, and accepting or rejecting the state based on a Maxwell-Boltzmann criteria. To generate the trial states, we provide the indiviual atoms with random velocities, and evolve the system with molecular dynamics. Molecular dynamics simply integrates the equations of motion, based on the energy terms in the final force field, and evolves the system. The number of molecular dynamics steps between trial configurations is chosen such that the auto correlation with respect to polarization (the property of interest to us) decays at an 'acceptable' rate.

Compound 1

Development of the force field for compound 1 (**Figure 7**) was quite routine, with the exeption of two torsions that are strongly coupled, and required care in parametrization. By inspection, it's easy to see that the torsion about the carbon-nitrogen bond in the nitro group, and the carbon-oxygen bond in the nearby anisole group definitely interact. Since it would be 'unphysical' to lock the nitro group into some conformation while getting *ab initio* energies for various anisole dihedrals, it was allowed to 'float'. This required us to first fit the nitro torsion, and then allow it to float while fitting the anisole torsion. The results of these fits can be seen in Appendix A.

The calculated polarization for Compound 1 was -260.277 ± 17.586 nC/cm^2, which is in good qualitative agreement with the experimental value of -550 nC/cm^2. While the simulation generates an absolute magnitude of the polarization which is less than the experimental value, this is to be expected, as the Boulder Model does not account for orientation of the cores along the polar axes (due to π stacking, steric, or some other intermolecular effects), which we anticipate would raise the magnitude of the polarization significantly.

Figure 8 shows the auto correlation function for polarizations along the three primary axes. This function tells us how long (or how many monte carlo steps) it takes for a given value to be decoupled from a previous one. **Figure 9** shows the polarization as a function of trial configurations.

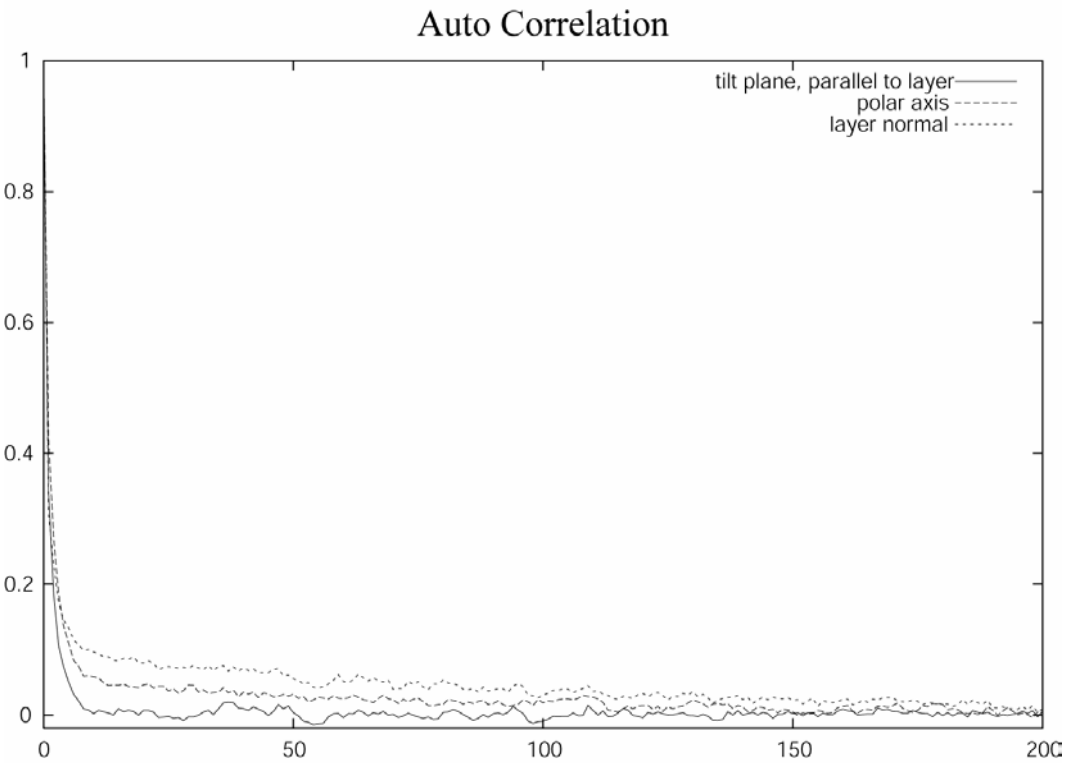


Figure 8: Auto correlation for the simulation of compound 1.

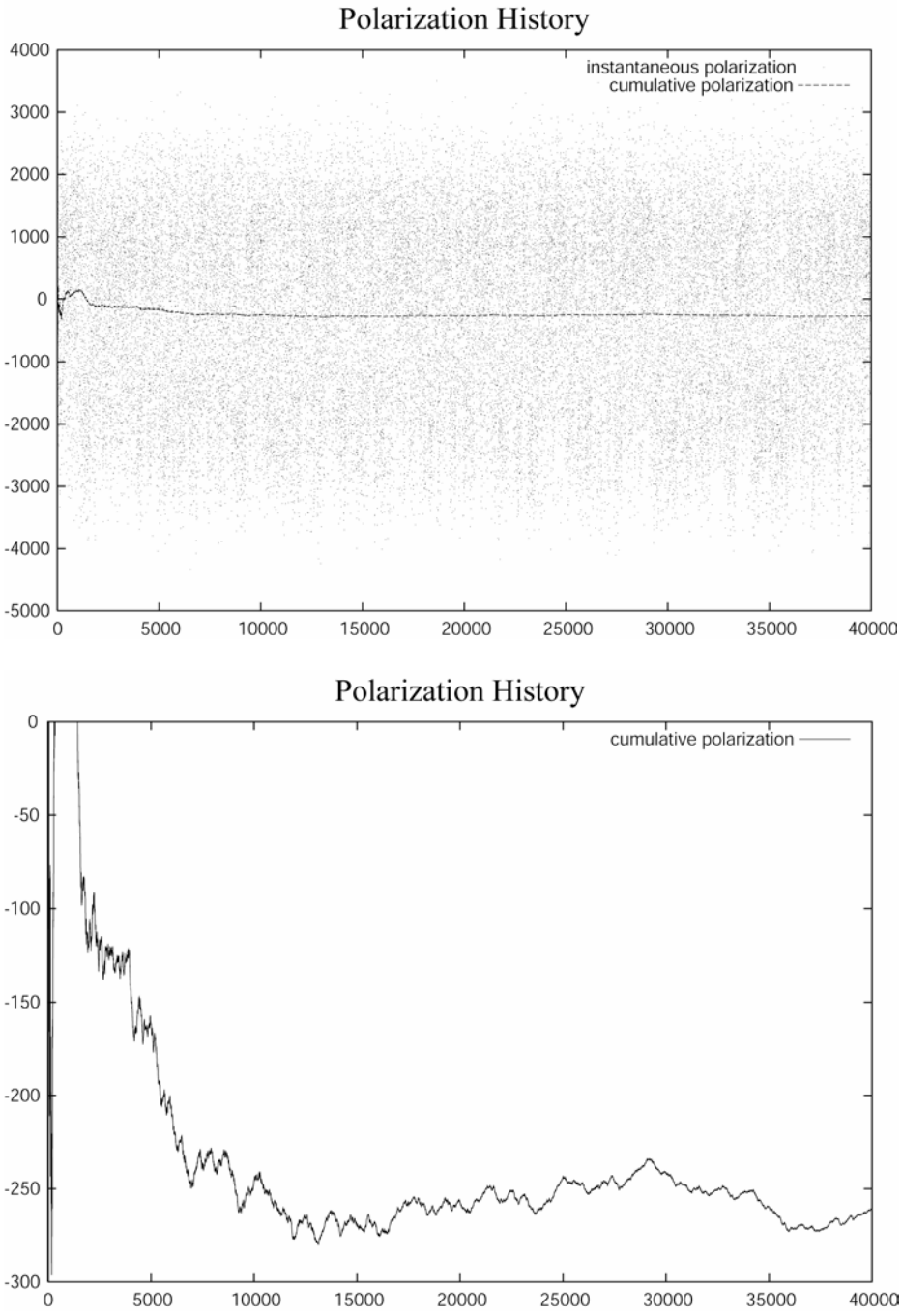


Figure 9: Polarization history for the simulation run on Compound 1, with instantaneous polarization (top pane), and without.

Chapter 3

Software, Algorithms, and Gory Details

This chapter covers the actual software that comprises FFDev. There are many programs in the entire suite, all with their individual functions. Here we do not try to give any kind of tutorial on using the software, as that is the subject of Chapter 4. Instead, we discuss the overall design, and the functionality of the individual components.

Overview

The rest of this chapter is definitely 'gory', so a brief summary of what happens when we generate a force field is presented here. The software is comprised of two agents (**Figure 11**). The qdb (quantum chemistry datbase) and related programs are resposible for holding, calculating, and returning various *ab initio* information to the process agent. The first step is to create a structure of the molecule for which you want a classical interaction potential. This can be done with any molecular drawing program, but for the best results, it should be in a conformation close to the global energy minimum. Running qdb_check on this file will create a partial force field file that contains all of the information presented in Appendix B (which atoms in the parent molecule should be represented by which fragment atoms, and the same information is available for all of the bonds). If the database did not have fragments for the initial parent molecule, another program will submit the *ab inito* calculations. Finally, when the database has the necessary information for all of the fragments, the final force field is generated with the final programs.

Design

All too often in academic software development, the first step of software creation is neglected to some degree. That step is software design. There is no *de facto* authority on the subject, as it is still very much evolving [32], but browsing various sources does reveal a pattern of topics that are very useful to guide a non-software engineer in this step. Among the many considerations, we paid special attention to (and will discuss in more detail) the following:

1) Portability
2) Scalability
3) Usability
4) Maintainability
5) Reusability

6) Performance

Portability

It was our goal to create a software system that would be functional on as many different computer platforms as possible. A platform includes both the type of processor, and the operating system. More specifically, we wanted to develop a system that would run on all *NIX variants, Intel x86/Microsoft Windows, and Macintosh (note that MacOS X was not available in the beginning of the project, but its release will greatly simplify our eventual goal). This restriction alone severely limits the choice of computer languages one can use. There are compilers for the C language available on just about every platform in existence. Additionally, Perl is also available on an incredibly large number of systems [33]. Unfortunately, neither of these languages supports any kind of graphical user interface, or image rendering directly. The original project design did not include plans for a user interface, or molecular rendering, but later development obviated the need for these tools. We settled on Tk for graphical user interface development, and OpenGL for graphics rendering, since both are available for both *NIX systems, and Microsoft Windows.

Scalability

While our own requirements for the developed software are quite moderate, we wanted to build a system that would eventually be useable on much larger problems. A typical liquid crystal molecule could weigh as much as 1000 atomic mass units or 160 atoms, but proteins can weigh much more, as many as 300,000 atomic mass units, or 50,000 atoms. Further, if one eventually used our libraries for simulation (a secondary consideration in development), one might need to have many large proteins resident in RAM at the same time. There are two specific areas of the program that are most effected by this issue.

Firstly, in the portions of the program written in C, we have created a ‘fundamental atom type’. This is the data structure used to represent an atom, for any task. In the current implementation, 50,000 atoms require only about 15 megabytes of RAM, allowing for very large systems to be held completely within RAM.

Secondly, part of the program suite involves interface with a database of quantum chemical calculations. The database currently has around twenty entries, and a typical entry on a Pentium III class system would take approximately 4 computer days to generate (note that these times are highly variable). When the database has grown to 2000 entries, the current program that serves information from it will have grown to 330 megabytes of RAM, once again, a somewhat moderate requirement for such a large amount of data.

Finally, the entire current code base is completely leak free (in terms of memory usage, and utility functions for freeing the more complex data structures are provided for ease of use by developers). Some libraries can be ‘abused’ in such a way as to

introduce memory leaks, but this is unavoidable in a procedural (non object oriented) language such as C.

Usability

Normally when software is developed, only the end user is considered. Since it was clear from the outset of the project that the work could never be completely finished within the timeframe of a single thesis, it was decided that both the developer and the end user be strongly considered in the overall design and implementation.

One of the most difficult tasks for a programmer working (for the first time) on somebody else's code is to understand both the problem the original developer was trying to solve, and how they actually solved the problem. We have attended to this difficulty in four ways. Firstly, all source code is copiously commented (approximately 25% of the lines are comments). Secondly, we have broken the overall problem down into small enough steps that it should be reasonably easy to understand what the problem is, and in turn, how the portion of code solves that problem. Thirdly, since one of the easiest ways to understand a problem is by watching the data flow through it, we have made all input and output be in text only format, and (hopefully) in plain English. Finally, by using procedural languages (C and Perl), we are forced to solve various problems in the same way that scientist generally do. There is much heated debate over what kind of language is better, but in our experience, scientists learn to solve problems by breaking them down, and taking steps, which is much more compatible with procedural languages than it is with object oriented languages.

Maintainability

This focus addresses not only maintenance, but extensibility as well. Extensibility is the process of adding onto existing work, without generating additional problems. We have addressed this issue in a variety of ways. As mentioned previously, all of the code is commented thoroughly, so it's easy for new developers to understand precisely what a given program or library does before they start work on it. We have also made every effort to separate the problems into 'specific' solutions, and 'general' solutions. This means that any code generated to deal with general solutions should be easily re-useable to solve other problems. It also makes the specific solutions more easily understood. Finally, consistently applied code formatting, long (descriptive) naming of variables and functions, and data abstraction that approximates chemists' notions all aid in the maintenance and extension of FFDev.

Reusability

This was largely addressed in the maintainability section. Reusability is the ability to take code that has already been generated, and use it elsewhere. The largest effort in this specific area was applied to the development of the atom_handling library, which was designed to do anything with the fundamental atom type that one

might want to do. Where functions needed to be used by both C and Perl programs, libraries were written so that the same function would be available from both languages. Towards the end of the project, after I got a bit more experience with Perl, several reusable libraries were developed, to aid future development using that portion of the code base.

Performance

Performance is listed last in the list of major concerns for a very simple reason. All too often, pure focus on performance issues compromises all of the other important issues, as addressed in the previous sections. Performance has not been utterly neglected, however. It is widely accepted that compiled C code is the fastest form of executable, save for assembly or machine programs (which are utterly non-portable). Contrary to popular belief, however, Perl is not nearly as slow as many believe [34]. Perl is a (run time) compiled language (not unlike C), and the development time is much faster, since the programmer need not spend their time with memory management, or character by character manipulation. For these reasons, portions of the code that have heavy performance requirements have been developed in C, and the rest was developed in Perl.

Conventions

For all of the following sections, program names will be given relative to the 'ff' directory. After the first mention of a program, the program extension and/or directory prefix may be omitted. When examples with syntax are presented, items in angle brackets (<, >) are mandatory, and must be supplied verbatim, and options in square brackets ([,]) are optional. Items separated by the pipe symbol (|) represent valid options, but only one of the options may be specified (exclusive or).

On the topic of descriptors

For the average organic chemist, it's trivial to look at two atoms in two different molecules, and decide whether or not they're 'similar'. Those involved in generating force fields do this regularly, but our goal was different. In order to automate this comparison, we needed some way to assign real values that could be compared to the atoms in any given molecule. These values (almost certainly) must be numeric, and they must also somehow capture the essence of the 'character' of the atom in question. Careful analysis of how an organic chemist makes this comparison reveals that they must rely very strongly on two factors. Most importantly, chemists' notice what 'kind' of atom they're looking at (e.g., carbon, hydrogen, nitrogen, etc.). Secondly, they notice the bonding in the nearby environment; for example; is this atom aromatic? aliphatic? What is the hybridization? There are many algorithms available for detailing the notion of 'similariy' in organic chemistry; the one that suited our purposes was the qcode..

The ubiquitous qcode

Before any real discussion about the software can continue, one needs to have a solid understanding of what a qcode is (a way to generate atom types), and how we use them. Any typical atomistic simulation software requires that each individual atom have a particular 'type', which identifies it as somehow 'chemically different' from atoms of other types. In *ab initio* quantum mechanical electronic structure calculations, however, the closest concept to atom type is the type of the nucleus, which is required to know how much positive charge it has, yet provides no differentiation between different carbons, for example. This difference introduces an important problem in mapping *ab initio* data onto classical interaction potentials.

The typical solution in other force fields is to look at the atom in question, and 'categorize' it as one of the available atom types (i.e., an aromatic carbon might be C_R). This approach can be automated [35], and often is, but was unsatisfactory for our purposes, as it results in a huge loss of information garnered from our *ab initio* calculations. The solution to this problem lies in Edgardo Garciá's qcode algorithm [36].

A qcode is a vector (or list) of numbers that uniquely identifies an atom, based on its topology and the electronegativity of topologically relevant atoms. For the current project, qcodes of depth 20 (QDEPTH) have been used throughout, but other QDEPTHs would be easy to implement.

The following section is (unfortunately) quite techincal in nature, as it gives the specific algorithm for derivation of the qcodes. The algorithm for determining the qcodes of all of the atoms in a molecule is as follows:

1) Assign a reduced electronegativity to all atoms in the molecule, which is given by: $\sqrt{\frac{\text{Pauling electronegativity}}{1+\text{the number of bonded atoms}}}$, this is the 0th element of the qcode vector

2) From n = 1 to (QDEPTH -1), and for all atoms in the molecule, do the following. The qcode at the nth position is given by: $\left(\frac{\sum \text{Neighbor's qcode}[n-1]}{\#\text{ of Neighbors}}+\text{This atoms qcode}[0]\right)\Big/2$

3) For each atom in the molecule record the current value of qcode[0]. It is used in the next step.

4) In the last step, we convert the intermediate qcodes to final qcodes. From n = 1 to (QDEPTH -1), and for all atoms in the

molecule, do the following. The qcode at the nth position is given by: $\frac{\text{qcode}[n] - \text{qcode}[0]}{\text{qcode}[0]}$

While the details of the algorithm are difficult to grasp, the results are absolutely ubiquitous to our work. Having a qcode available for each atom, however, is not enough for doing comparisons, we needed a way to be able to say whether two atoms are 'similar enough' (something very commonly done among chemists, but a bit difficult to implement algorithmically). We defined a 'qcode deviance', which compares two qcodes (many valued lists of numbers), and returns a simple scalar (one value, in this case, a number) that defines how 'similar' two qcodes (and thus, the underlying atoms) are. In practice, the deviance takes the form of a floating point number, such as 3.185. The integral part of that number (3) indicates to what range the topology of the two atoms in question are identical (**Figure 10**). The fractional part (.185) roughly corresponds to a percentile rank of 'how close' the neighbors beyond the exact match range are. A low value would indicate that beyond the exact match range, the molecules are radically different. A high value would indicate that beyond the exact match range, the molecule retain a fair degree of similarity. Once again, the following discussion is anything but easy to read. Unless you're interested in the exact implementation, it may be irrelevant. This algorithm is implemented as follows:

1) Define a floating point tolerance. If the absolute value of the difference between two numbers is less than this value, the two numbers are considered identical. This is necessary, since all floating point numbers on any machine are inaccurate in the last decimal place.

2) For each value in the two qcodes (denoted hereafter as qcode1[n] and qcode2[n]) from n = 0 until then end of the qcode, compare the two values. If they are identical, move on.

3) For the last pair of identical qcode elements, record the exact match, which is n + 1 (since the first element of the qcode is numbered 0).

4) Define a weighting factor (0.5), a sum accumulator (0), and the exact match (found in step 3). For n = one past the last match to n = the end of the qcode, add to the sum:

$$\left(e^{-weighting\_factor \cdot (n - exact\_match + 1) \cdot |qcode1[n] - qcode2[n]|}\right)^2$$

5) Since we want an average deviance, we set sum = sum / (qcode length – exact match). The sum now currently represents an

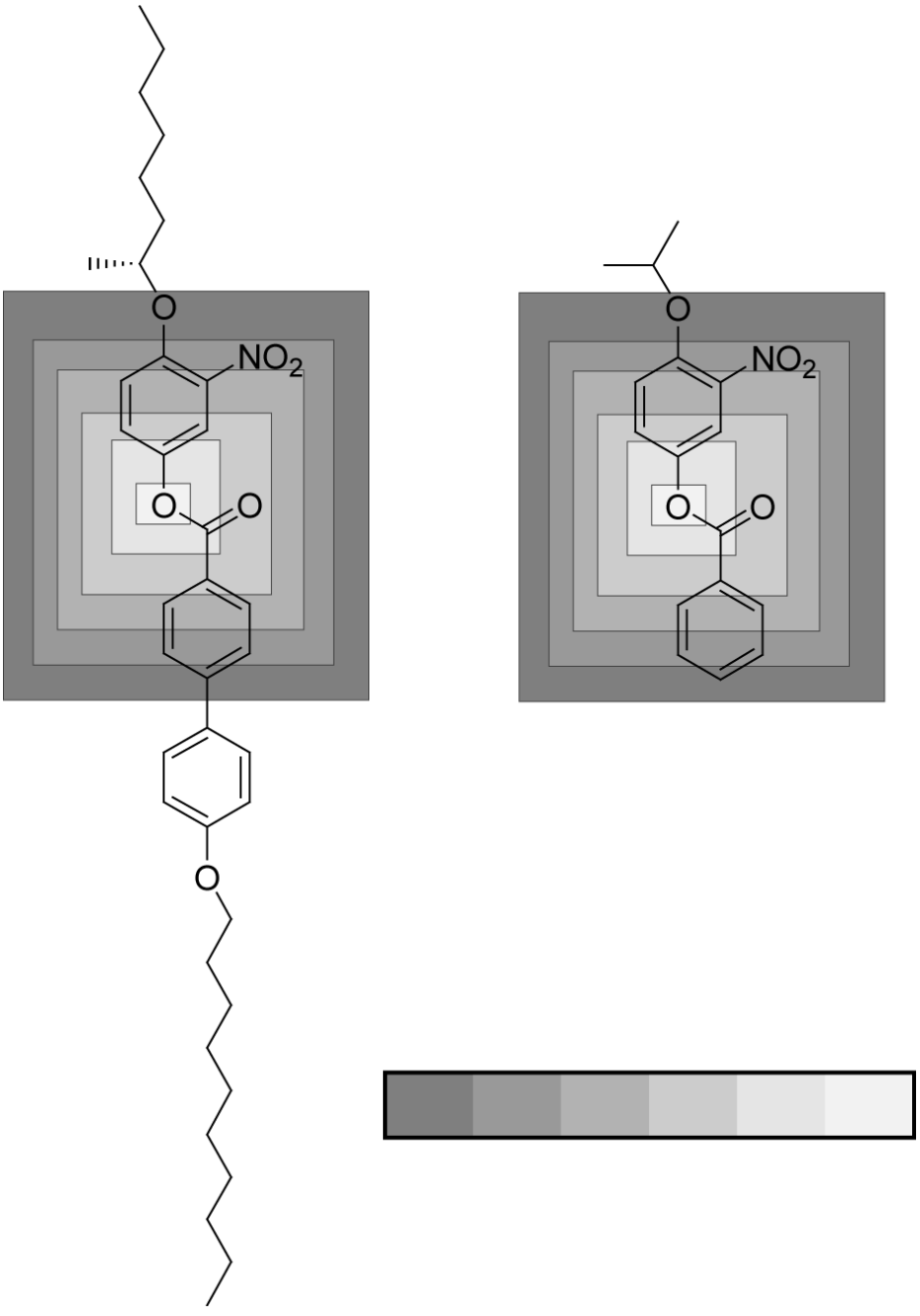


Figure 10: The "sphere of influence". For each of the atoms in the parent in all but the outermost shell, the corresponding fragment atom is connected to exactly the same atoms, and would give the appropriate "exact match".

'average error' in the non-exact matching portions of the qcode. In order for it to display the proper behavior (i.e., small error give a large value in the fractional returned part), we need to do further manipulations.

6) Set our fractional match to $-\log\left(\sqrt{\text{sum}}\right)$ (chemists may recognize this as a variant of the p function). If the sum is 0, or our fractional match is less than 0, return (exact match + 0.999), since this is more or less a perfect fit past the exact match part. If not, our fractional match is set to $fractional\_match/25$, which casts it into the (approximate) range of 0.01 to 0.70. Return (exact match + fractional match).

The qcode deviance is then available to all programs written in C and in Perl (via an XSUB [37]). Empirically, this comparison gave deviances which agreed with 'chemists' intuition' in all but a handful of cases, out of 80. Out of the dubious matches, all were 'close calls'. Within the project overall, we frequently compare atoms, in which case, we use the above described deviance by itself. We also frequently need to compare 'bonds', which are identified by the atoms on either end. In that case, we use the geometric mean $\left(\sqrt{\text{deviance1}\cdot\text{deviance2}}\right)$.

Since qcodes only contain topological information, any stereochemical information is lost. To alleviate this problem, we defined our own scheme for assigning absolute configurations to tetrahedral stereogenic carbons. The algorithm is very similar to the CIP scheme [38], but instead of using that scheme's prioritization, we relied on the qcodes to provide it. Using qcodes for this purpose makes the assignment more stable to small changes in connectivity, which was critical for our mapping fragment atoms and bonds to parent atoms and bonds.

Functional overview

This section will present a functional overview of how FFDev works. In very large software projects, it's impossible to describe the entire system in one view. Regardless, we will attempt to present the overall operation and functions of the individual components in a single pass. This presentation is very loosely based on ideas from the Universal Modeling Language [39].

Collaboration Summary

According to "The Unified Modeling Language User Guide", a collaboration diagram is an interaction diagram that emphasizes the structure organization of the objects that send and receive messages. Since our package is not written in and object oriented language, this view is not strictly applicable, but useful nonetheless.

Note that I have taken the liberty of using older symbols for presenting the behavior in this diagram, as they should be more familiar to readers.

Figure 11 shows the interactions of the major components of the software system. The quantum database portion (QDB) is an independent agent, and runs constantly. The generation system uses (and relies heavily upon) the QDB, and is designed to run through exactly once for any desired force field.

The QDB

The QDB consists primarily of four different programs (and several other very small programs). These are: qdb/qdb_query_server.pl, qdb/qdb_input_server.pl, qdb/qdb_local_submit.pl, qdb/torsion_driver.pl and qdb/qdb_maintenance_utilities/qdb_utilities.pl. An overview of each of their functional behavior and responsibilities follows.

The program qdb_query_server is responsible for providing all database output.

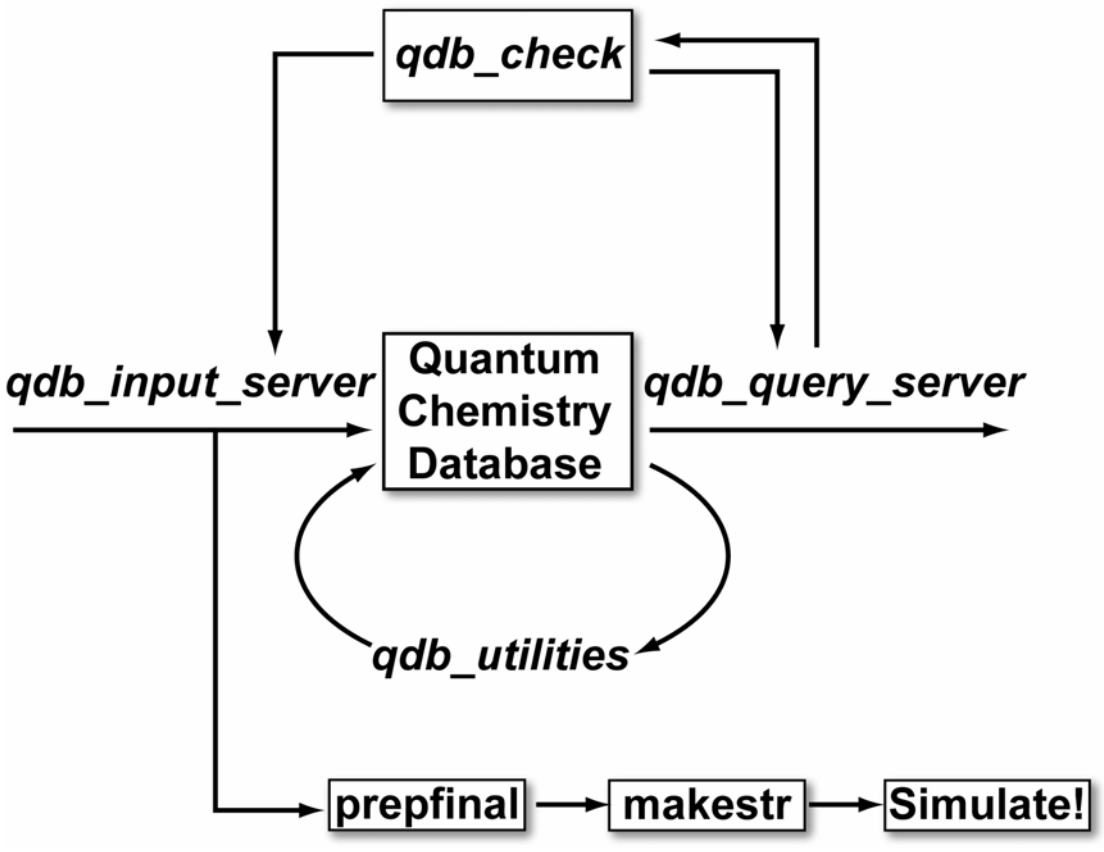


Figure 11: Collaboration summary diagram for the software. The central items represent the qdb agent, while qdb_check, prepfinal, and makestr represent the process agent.

It is a daemon (runs constantly) that listens to a TCP/IP port on the host machine, receives plain text queries (one can use a telnet client to connect to it if they like), does a search, and returns the requested information. The commands it understand are as follows:

<get> [# of matches] <atom|bond> <match> {qcode_1_list} [{qcode_2_list}]

<get> <charge> <charge_type> <directory> <atom_number>

When making a query to the database, the user has the option of asking for several matches, or omitting the number of matches, and getting the 'best' one. Note that there may be several 'equally good' matches, and these will all be returned if that is the case. When requesting an atom match, the user should provide one and only one qcode. When requesting a bond match, the user must provide two qcodes. Note that in this case, the curly brackets have no special meaning typographically, but are required by the server to parse the qcodes.

The charge query is also quite flexible. Note that in order to request a charge, the client must know which atom on a specific fragment they want a charge for. This query finds all topologically equivalent atoms, and returns the average charge to the client.

The program qdb_input_server is responsible for placing all new *ab initio* calculations in the database, and requesting that the calculations be done. It is also responsible for starting all relevant torsion_drivers (which will be discussed shortly). In its current state, it is not a daemon, but a run-once type of program. When input is put into its input directory, and it is run, it creates and submits any new fragments to the local *ab initio* program and queuing system. It also starts torsions (via torsion_driver) that the input may have requested. The user is personally responsible for looking within the database for completed fragments (in control/qis/in_progress), and placing them in the root database directory. It would also be wise to re-run the request, as any necessary torsions belonging to new fragments are not calculated until qdb_input_server sees the relevant fragment in the database.

The program qdb_local_submit is responsible for managing jobs within the local computing environment. It is likely that if the entire system is ported to another computer (or group of computers) that this program would need to be heavily modified. This program takes requests for jobs in the control/que file, and when it's running the jobs, places them in submitted_jobs. It also leaves messages for the requesting processes (via a message.<pid> file), so they can continue their work, if they were waiting for the calculation to finish before proceeding. Before it actually submits a job, qdb_local_submit does its best to be the most polite user of the DEC cluster. It first starts by counting the total number of jobs the user is running. If it is above some maximum value, it refuses to submit the job. It then looks at each individual machine. If the requesting user has a job on that machine, it eliminates that machine from the potential candidates. It then tries to allocate approximately 1.8 times as much memory as the job is likely to take on each of the candidate machines.

If any machine fails the memory allocation test, it is also removed from the potential candidates. Finally, it checks the load on each of the remaining machines three times, over the course of three minutes. The machine with the lowest load is selected to run the job.

The program torsion_driver.pl is responsible for running all of the torsions about any requested bond. It is started (automatically) by qdb_input_server.pl with information on which fragment, and which bond within that fragment we need angle vs. energy data for. If a directory already exists for the requested information (either because the previous torsion didn't finish, or perhaps because it is finished) it tries to restart any within that directory. In any case, what it does is run some number of torsions (provided from a configuration file) that are below some cutoff energy (again, from a configuration file, we've been using 20 kcal/mol as the cutoff). If there are no very high energy conformations (as would be the case for a torsion within an aromatic ring, for example), it will give some number of evenly spaced torsions. If there is a cutoff because of a high energy conformation, it will try to fill in as many angles as it needs to generate the requested number of data points. When it finishes, it offsets all of the energies so that the lowest energy conformation is at 0 kcal/mol, and records the information.

The maintenance of a (growing) database quickly became a concern. Each database entry has information about any stereochemistry the fragment may have, as well as other information. It became quickly apparent that we needed some way to check and repair the database as it grew. This is where qdb_utilities comes in. The program has three modes (specified on the command line), namely summarize, verify, and update (which may have been better named 'repair'). There is a subdirectory in its home directory called utilities, where small 'helper' programs reside. These include programs that re-generate qcodes, determine which bonds should be frozen in a torsion drive, etc. The program is designed so that it should be relatively easy to define a new task, and place the defining program in the utilities directory. It's then only a few small edits to include the new test in qdb_utilities. After placing a new entry into the database (from a completed request by qdb_input_server), it is critically important to update all portions of the database, as the input server does not properly 'condition' the fragment.

The generation system

The generation system is comprised of a surprisingly fewer number of components. The program qdb/qdb_check is responsible for taking an initial structure and creating the initial file it needs for specification of the pending force field. The program cmap/map_charges.pl is responsible for mapping (and normalizing) symmetrized charges onto the parent molecule. The program finstr/prepfinalff.pl is responsible for gathering all of the information necessary to construct an arbitrary force field for any modeling package, and saving it in an easy to 'reconstitute' way. Formally, this is where the work of FFDev ends, but there is one more component that we need in order to create input for Matthew Glaser's [7]

modeling software. That program is finstr/makestr.pl. All of these programs are described in some detail in the following sections.

The program qdb_check is the does the core work of the force field preparation. Firstly, it takes the molecular structure (as an XYZ file, with or without connectivity information), and verifies that it can fully represent the molecule in its own native format. The checks include checking bond orders, formal charges, valences, and other relevant properties of the atoms. It then generates two lists. The first list is a list of all of the atoms in the molecule. The second list is a list of all of the bonds in the molecule. Here is where it begins its search for fragments to use to generate the final force field from. It establishes an internet TCP/IP socket connection with qdb_query_server, and 'asks' the server if it has relevant matches for each of the atoms and bonds that it needs. It records this information, and gets to work on atoms and bonds that the database has no relevant fragments for. For each of these 'orphan' atoms or bonds, it begins to grow fragments that would satisfy the 'similarity' criterion. The new fragments are real substructures of the original molecule, with hydrogen's provided as need to fulfill valence. It was in this phase that we really got to compare how well the qcode matching criterion worked, and the data is presented in **Figure 12**. For the three test cases, the fragments that match, as well as all of the atom and bond mapping for compound 1, can be seen in appendix B. Finally, a request is output, which is destined for qdb_input_server. The request is in the form of a file which lists the parent molecule, an atom in parent to atom in fragment (from the database, or new fragment) list, a bond in parent to bond in fragment list, and finally, a trailing list of any new fragments it would like added to the database.

Unfortunately, the next step is to wait, since *ab initio* calculations can be quite time intensive. After all of the necessary calculations are done (or perhaps immediately, if 'good' fragments for everything in the parent molecule were already found), prep_final takes over. This program reads the (now mangled by qdb_input_server) file originally provided by qdb_check, and organizes all of the data into Perl data types. During this process, it runs map_charges, which simply queries qdb_query_server for symmetrized charges for all of the atoms, and then normalizes all of the charges so the sum of charges on the parent molecule is 0. After prep_final is done, it dumps its initialized data into a file that is trivial to reconstitute in another Perl program.

It's easy to create 'all the information needed for any force field', but it's a much more difficult task to translate the information into useful input for some simulation package. This is where makestr comes in. As is the case with all modeling software that we are aware of, it is required that all atoms have 'types' associated with them. In this case, we assign somewhat arbitrary types to the atoms, such that only atoms with identical qcodes end up with the same label. The rest of the program assigns parameters (as described in Chapter 2) to all but the torsions, and maps bond lengths and angles from the *ab initio* minimized fragments onto the parent molecule. It then provides the user with a number of options for fitting the remaining torsions. The end result is a directory structure full of the relevant parameters, and a final master force field and structure, ready for input into a binding site calculation.

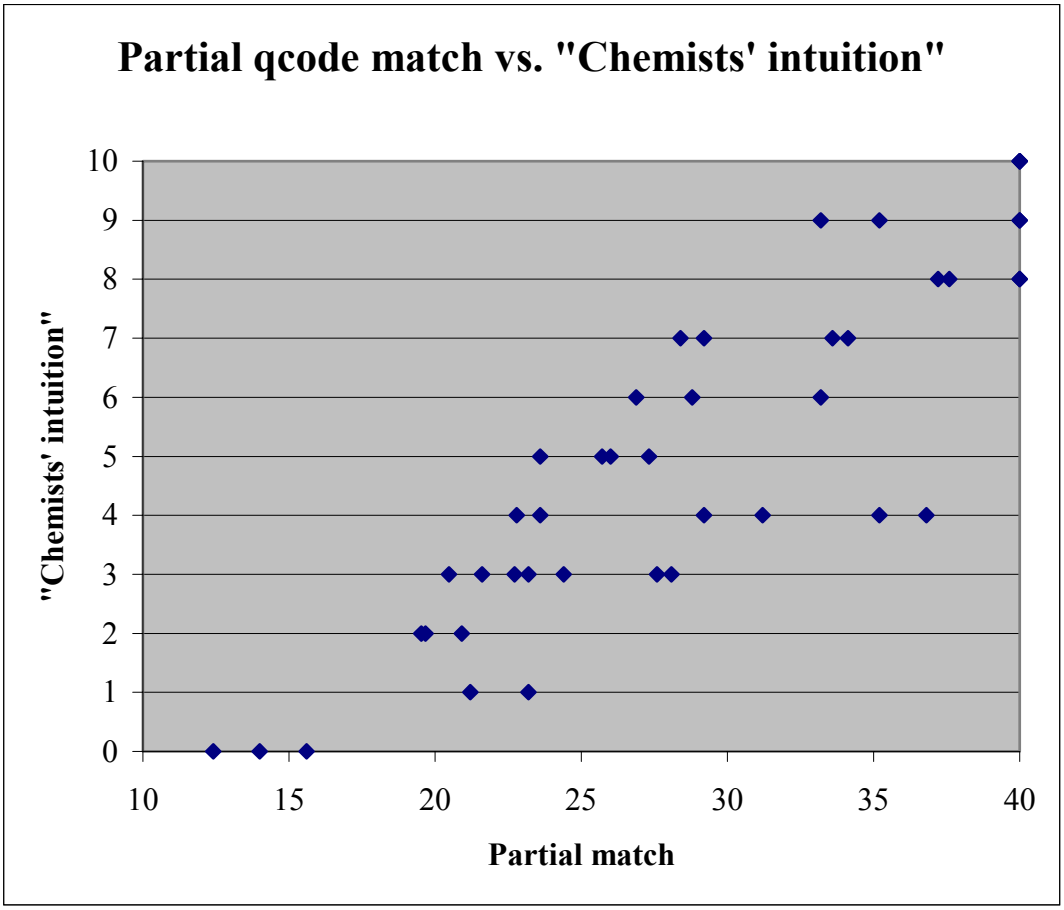


Figure 12: In the above graph, the exact partial match (times 100) on the x axis, and a 'chemist's intuition' as to how good the match is on the y axis. For partial matches over 40, the values have been changed to 40, which is considered a practical maximum (there were several such values in the dataset). The matches were atom to atom matches for a variety of fragments generated from the base structure of Compound 1. Note also that the exact match is not shown. The following numerical conversions for

Other libraries and utilities

Aside from the main programs, there are a number of other programs that exist either to make life easier, patch known problems in the current implementation, or perform some other miscellaneous tasks.

In the root directory of the project, there's a program called Compile_all_fudge_scripts.pl. This program began as a 'quick way' for me to compile all of the C code in the project, but it has evolved into a multi-platform makefile maker, and project wide compiler. Running this (on your local machine) should compile all of the C code in the entire project, as well as the XSUBs needed

by the Perl programs that use get_qcode_deviance(). The project currently compiles effortlessly on Linux/PPC, Linux/ix86, and DEC alpha/OSF4 machines. System specific hints and configurations can be found in general/os_specific. On a related note, for every directory that has a compileable C program, there is a configure.pl file. This file will make a Makefile in the current directory, with all appropriate system specific options. It can also compile the program in a variety of ways, to support debugging, profiling, etc. Type "configure.pl –h" in any of these directories to see what options for configuring your Makefile are available.

There are several other files in the root directory of note. COMPATABILITY discusses any decisions that have been made that may affect portability. It also mentions any special libraries that the user may need to install on their system to be able to compile/use the package.

To_do.txt is full of exactly what it says. It notes any current limitations in various parts of the software. Some of the tasks may have been completed, and if they have been, it should be noted here.

The general subdirectory contains a variety of other 'generally useful' libraries. The program chkmem is a utility that is useful for determining if a machine is capable of allocating a given quantity of RAM, and is used by qdb_local_submit before submitting queries.

The core of our chemistry paradigm for the C code in the current project is encapsulated in the atom.h and atom_handling.c files, which together, represent our atom handling library. These files define, and provide functions for manipulating, our atom data type.

vector.c and vector.h are very basic (and quite inefficient) libraries for handling simple vector access and manipulation. They also provide very rudimentary support for some linear algebra functions.

total_atom_byte_size.c is a small utility that will tell you how big a single atom type in memory is. It can be used to make estimates of the size of large scale programs that use this atom type.

my_socket.c is a library for using internet TCP/IP sockets. It simplifies their usage, and gives some utility for receiving data, which is normally quite tedious, due to buffering considerations.

rc_file_handling.pl is an old style Perl library for getting options from the resource files used in this project. The only current files we use of this sort is in qdb/.qdb_checkrc, which has all of the configuration options used by various programs in the package.

clean_environment.pl is a library for un-tainting environment variables. Perl has a mechanism that allows the user to know when a variable may have come from an 'unsafe' source. If the relevant option is selected, Perl will not allow tainted variable

to be used to output anything to the system. Occasionally, we need environment variables, and sometimes we need them in program for which internet security is a very important part of the programs function. In these cases, we manually un-taint the variables, and each instance of this is commented, with a risk assessment.

The doc subdirectory provides a (very old) Overview of the project, and a short tutorial on how to use CVS on the DEC cluster.

The genff and sim subdirectories contain the stub of a library whose original intent was to provide the classical energy evaluation necessary for the torsion fitting of the program. The relevant files are .ff_form, which is a plain text file describing the form of the force field desired, and nrgforce.c and nrgforce.h. The nrgforce library is designed to provide seamless integration with the atom_handling library, but more importantly, it is capable of run-time force field configuration, and hides nearly all details of other (proposed) functionality from the calling program.

The graveyard directory contains programs which have been abandoned in favor of redesigned programs. It may (or may not) contain code that could be useful for further development, but should not be used routinely in the program's normal operation.

The log2str directory contains just a bit of previous work not done by myself (log2str converts a log file to a str file). It also contains one function that is frequently used, called get_bond_order.c. This function assigns a bond order based on the atom labels, and the distance between them.

The one_timers directory contains programs that needed to be written to do one time functions (primarily database management), but there is no long-term need for their reuse. Once again, they may contain useful code to help meet future needs of the project, so they have been saved.

The perl_modules directory contains the Perl equivalent of C libraries. It is unfortunate that a couple of the modules have also ended up in general, but moving them to this directory would 'break' some existing programs. LINALG.pm is a module for performing linear algebra with native Perl data types. The most important capability of the library is that it provides a simple way to get dihedrals angles in accordance with the standard chemist's convention [37]. NETFLOCK.pm is a module to provide file locking over NFS networks. It is a voluntary locking scheme, which means that in order for the locking to work, all programs that use a given file must use the same library.

The original design of the program required that the software not be dependant on the computing environment. This means that the user should not be required to use the same commercial *ab initio* program as we do, nor should they be required to have the same job queuing system as we do. Two Perl modules were written to serve this purpose. g98_functions provides easy ways to interface with Gaussian 98's input and output, without requiring the calling program to 'know' which library it's using. If a

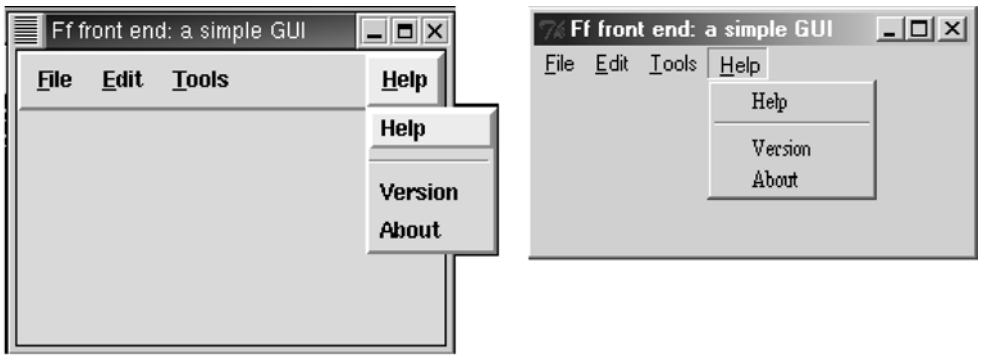


Figure 13: The program fffront.pl as it appears in GNOME (left) and Windows (right). In all x-windows implementations, the Help menu is supposed to be on the right side of the menubar, there is no such convention for Windows.

user has another *ab initio* program, they can simply copy this library, and re-write the functions to duplicate the behavior of the original ones. Similarly, local_functions provides an interface to behaviors specific to the users computing platform.

The qdb directory has seen the majority of development, and as such, has a variety of utilities that are not a part of the core implementation. format_for_g98.pl can be used to create .com files (for viewing with an appropriate molecular renderer) from .raw files, the format used by the database. kickstart_torsion_drivers.pl is a 'patch' program, to restart all of the torsion drivers, after killing qdb_local_submit. With some re-writing, this program will become obsolete. If you run this program, you will need to restart qdb_local_submit before the jobs will be resubmitted. As mentioned previously, the .qdb_checkrc file contains all of the configuration options that the various programs in the package use. format_connectivity.sh is a small utility to take a gaussian .com file with connectivity information, and create a corresponding file with connectivity in the style that our current simulation code uses. kqueryserver.sh will kill qdb_query_server regardless of what host it's currently running on. reghosts.sh is a small utility one can use to assist in setting up their ssh environment (which the current implementation of all inter-machine transactions is highly dependent on).

The runff directory contains a couple of 'proof of concept' programs. None of the work in this directory is ready for 'production use', but it may serve as a foundation for further development. fffront.pl is the beginning of a program designed to provide a GUI for all portions of the code base. When finished, it should have a database manager, and a force field creation manager. It is written in such a way that it runs with very similar results on both *nix systems, and Microsoft Windows (**Figure 13**). molren.pl is our own molecular renderer, and should eventually be able to read and render almost any molecular structure format known. It currently handles only our

own format, but even at its current level of development, creates quite nice renderings (**Figure 14**).

The shlib directory contains all XSUBs used in the program. Currently, the only shared capability we depend on is get_qcode_deviance(), but as the C and Perl portions of the code grow more interdependent, other XSUBS may be written.

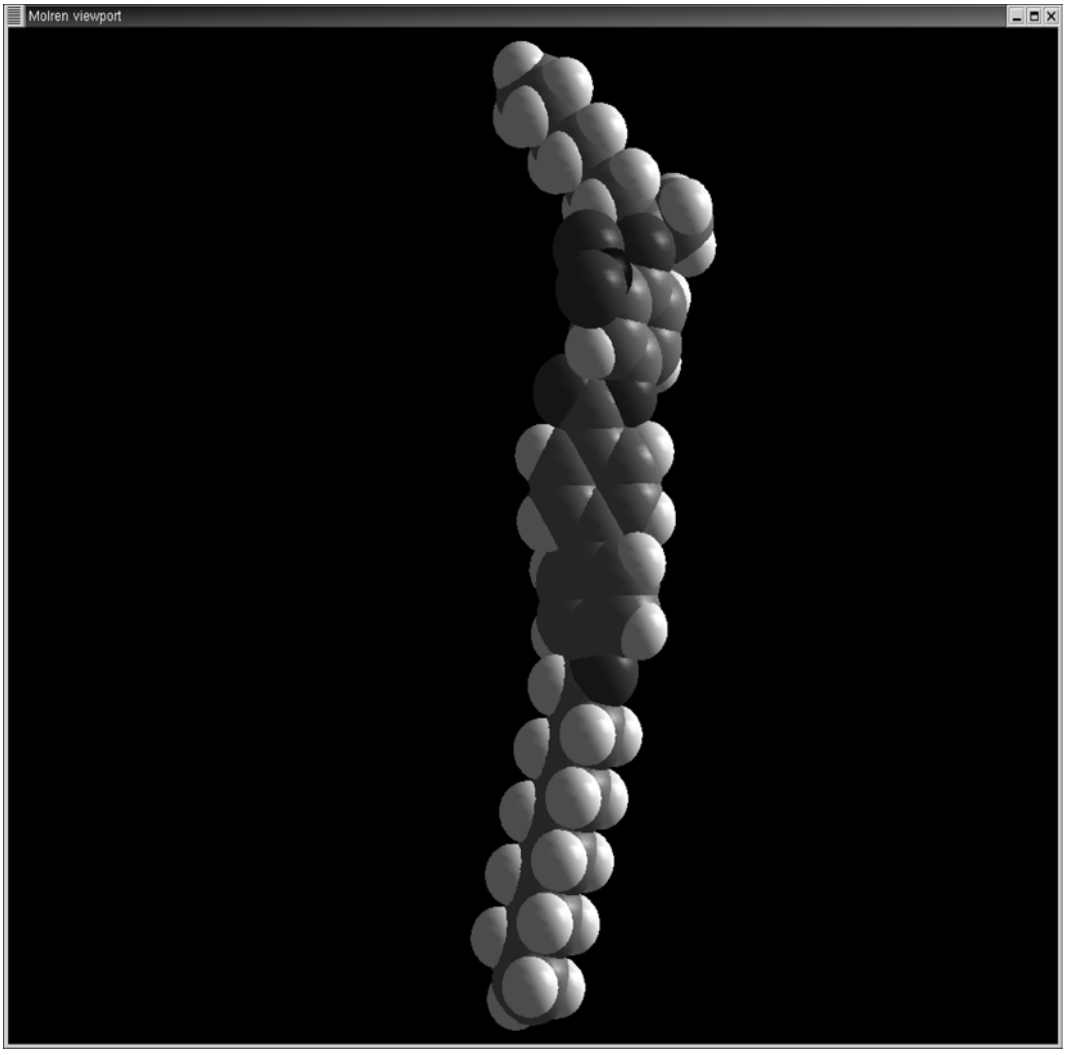


Figure 14: A rendering of Compound 1 from molren.pl.

Chapter 4

A tutorial

In this chapter we present a walkthrough of how the force field for our test compound (**Figure 7:** The compound for which force field was generated.) was generated. It will also cover 'variations' for procedures that are not encountered when generating this force field, but may be encountered for other compounds. It is intended to give the user of the software a template of how to do one of these, from beginning to end. Any data files that are generated by this run will be included in Appendix C.

Within this chapter, certain typographical conventions are used, to assist in clarity. These conventions were taken from "Programming Perl" [40]. *Italic* is used for path names, file names, and program names. `Constant width` is used in examples to show any literal output (or input) for programs, and relevant file contents. **`Constant width bold`** is used to indicate text that must be typed in exactly. *`Constant width italic`* is used to indicate that you must supply your own value. When there are optional values that you may have to supply, values in <angle brackets> represent mandatory values, while values in [square brackets] indicate optional values. If there are several valid choices <a|b|c>, they will be separated by the pipe character.

Two absolute paths will occur repeatedly in these examples, so we will shorten them. *qdb_path* is the path where your quantum chemistry database resides, in the case of the DEC cluster, this is */private_ffd/qdb*. The base path of the program distribution will be indicated by *ff_path*. After changing to a directory, subsequent commands are assumed to have originated from the last directory used. The command prompt will be indicated by a `%` as the first character on the line.

Getting Started

Before doing anything, make certain that your own environment is set up completely. The package frequently needs to communicate between the various machines in the cluster. To verify you are setup correctly, type:

```
% /ff_path/qdb/reghosts.sh
```

This will attempt to log you into all of the machines in the cluster. If you have to provide a password, or type anything in, (but "exit", which you should type at each new login), then the software will not work until you have ssh set up properly. Setting up ssh is beyond the scope of this walkthrough.

Additionally, you will need to compile all of the C programs and libraries in the package. To do this, change your current directory to **`./ff_path`**, and run:

```
% Compile_all_fudge_scripts.pl
```

In order for the program to run, there are several daemons that need to be running. Begin by changing your current directory:

```
% cd /ff_path/qdb
```

Start the query server daemon. This daemon may be started anywhere, but it is imperative that it is run on the machine indicated by .qdb_checkrc within this directory. If you're not certain, open .qdb_checkrc with your favorite editor, and find the line "`#query_server_host`". The next line is the host that the query server will be searched on. Note that you may need to make other edits to .qdb_checkrc to match your own computing environment. Start the daemon:

```
% qdb_query_server.pl &
```

The query server (as it is distributed with this thesis) may print out a lot of debugging information. This does not necessarily indicate that it's not working correctly, it just hasn't been removed yet. If you want to avoid having to see this, start the query server in another window, or simply redirect standard out to /dev/null.

Note that all of the daemons in the package are designed to catch SIGQUIT, and finish up gracefully. This is the preferred method for 'killing' the daemons. To find out what process id number (PID) the program is, type:

```
% ps –elf | grep qdb_query_server.pl | grep –v grep
```

You can then kill the appropriate program with:

```
% kill –SIGQUIT pid
```

If you are running this demo off of the enclosed CD (or an ISO image of the cd can be acquired from ffdev.sourceforge.net), all of the calculations already exist in the sample database, so no new *ab initio* calculations need to be run. If this is the case, please skip the next paragraph.

Now, we need to start up the local submission daemon. This daemon must be running on a machine that has access to the scratch directories of every machine, which also must be called /scratch_machinename. This is so the server can move the jobs to the correct machine before starting the Gaussian 98 calculations, to save on network communication. On the DEC cluster, this machine is jabberwock. Log into that machine, if necessary, before typing:

```
% qdb_local_submit.pl &
```

The path to patience

Now that we are ready to proceed, we'll begin with the fragmentation. Type the following to get started:

```
% qdb_check < samples/dave1.xyz > ff1.txt

Beginning 94 atom match queries.  Each dot represents 5 atoms.

                     \.................../

                     \.................../

Begin bond queries:  \.................../
```

The second two progress bars will print periods as the program does its work. ff1.txt will contain much of the information necessary for the final force field, but may require further processing. Copy ff1.txt into the qdb_input_server directory as follows:

```
% cp ff1.txt /qdb_path/control/qis/input/ff1.txt
```

Once again, if you are using the sample database, you will have no need to run any *ab initio* calculations. In this case, you may skip the next step. If you run the input server when all of the information is already in the database, the server will do nothing but go to sleep, waiting for some input that would need to have calculations run on it. Run the input server:

```
% qdb_input_server.pl
```

The input server may make new entries into the database, or run one or more torsion_driver.pl daemons. One can check the database for unfinished torsions by typing:

```
% chkincompletetorions
```

If there are incomplete torsions, they may or may not belong to your compound. The ff1.txt file we generated is human readable, so the curious can look through it to see which torsions on which fragments will be required to parameterize torsions within the parent molecule. Conceptually, there are only two types of entities that need to be mapped from fragments onto the parent. These are atoms, and bonds. Three and four body interactions all have either an atom, or a bond, that they are centered on. A typical line (from the bonds section) looks like this:

```
Dir: C20H16O3-0 Parent bond 10-15: Qdb bond 10-15: qdb homo
```

The Dir section is the name of the database entry for the fragment that will be used for this particular bond. The numbering for the bonds are all zero based, which means, depending on your method of visualization, you may need to add one to the atom numbers to get the correct bond (gaussview uses a 1 based numbering system). The ‘qdb’ at the end of the line indicates that the fragment exists in the database when the program was run, it may say ‘frag <#>’, if an appropriate fragment did not exist, in which case, the fragment specifications will appear at the end of the file. The homo (at the end) means that either there were no tetrahedral stereogenic carbons in

the molecule, or that the parent molecule and the fragment molecule have the same absolutely configuration at all tetrahedral stereogenic carbons. It could also be 'enantio', or 'diastereo'. If it is enantio, the torsion vs. energy will have to be reflected, before being fitted, if its diastereo, the fragment would not be appropriate. Since none of the test cases would have this problem, the code to handle these variations is not currently developed, though they will trigger errors if detected.

If qdb_input_server made any new entries into the database, they will automatically be run by qdb_local_submit (which you started earlier). For a record of what jobs was submitted to what machine, and when, read /qdb_path/control/qdb_local_submit_err.log. In the case that there were new fragments submitted, they can be found in /qdb_path/control/qis/in_progress, under the name of the file you submitted, in this case, ff1.txt. Since the input server is not 'finished' yet, the user must wait for the selected calculations to finish, and manually copy any new fragments into the database, for example, with something like this:

```
% mv /qdb_path/control/qis/in_progress/ff1.txt/C2H4O-3 /qdb_path/
```

After adding any new fragments to the database, you must run the maintenance utility to 'finish' the database entry:

```
% cd qdb_maintenance_utilities

% qdb_utilities -ua

% cd ..
```

Hopefully, you will have not had to wait too long for the *ab initio* calculations to finish, or better yet, perhaps all of the entries are already in the database!

Completion

All of the *ab initio* calculations are finished, and you're ready to complete your force field. Before completing the force field, you need to make sure that qdb_query_server.pl is running, see the previous section for information on how to start (and stop) this daemon. At this point, you need to regenerate the ff1.txt file with qdb_check (unless qdb_input_server didn't have to start any new jobs for you). Follow the above instructions to do so. In future implementations, it will be left (in a finished form) in the /qdb_path/control/qis/output directory, and you'll not need to regenerate it.

To generate the data necessary for completion of <u>any</u> force field, run:

```
% /ff_path/finstr/prepfinalff.pl ff1.txt > ff1.fff
```

ff1.fff (final force field) is a (barely) human readable file, which contains all the data necessary to complete a force field of any design. If you do not use Matthew Glaser's simulation/torsion fitting code, then this is the point that the software ends, for you. If you do choose to read it, read on!

Closure

While the generic force field is finished, there is nothing like a useable force field yet. This is very much dependent on what simulation software you'll be using. Here, I discuss the usage of Matthew Glaser's torsion fitting, and simulation code. In order to use some of the features of makestr.pl, you will need to have the programs build_single and minimize in your path. To continue, do something like in the following example:

```
% /ff_path/finstr/makestr.pl ./ff1.fff


o)  Overwrite the directory structure and initialize
r)  Refresh the directory structure without destroying existing files
s)  Skip all initialization, and go into interactive mode immediately
t)  Try to fit and verify all torsions, this option is dangerous, and
    will definately take some time.  It will also _not_ initialize the
    directory structure, so you should refresh or overwrite if you're
    not certain the directories are properly set up.
q)  Quit before doing anything
What shall we do? (o|r|s|t|q) [s] o
Initialization progress: \........./
                          \........./
Entering interactive mode:

1)  Change current fragment/torsion (C8H18O-0, 1)
2)  Delete all information for current fragment/torsion
3)  List all torsions with their status
4)  Fit current torsion
5)  Check log file from last fitting run
6)  Verify current torsion
7)  View graph of fit
8)  Declare this torsion finished
9)  Quit
Your choice? (1|2|3|4|5|6|7|8|9)  9
```

Now run it again:

```
% /ff_path/finstr/makestr.pl ./ff1.fff

o)  Overwrite the directory structure and initialize
r)  Refresh the directory structure without destroying existing files
s)  Skip all initialization, and go into interactive mode immediately
t)  Try to fit and verify all torsions, this option is dangerous, and
    will definately take some time.  It will also _not_ initialize the
    directory structure, so you should refresh or overwrite if you're
    not certain the directories are properly set up.
q)  Quit before doing anything
What shall we do? (o|r|s|t|q) [s] t
Note that even after running the torsions, you will need to manually
check them to make sure the fits are good, etc.  Feel free to simply
re-run this program after the batch is done, then select s (to skip
the directory initialization).  Also, be certain to enter the new
values into the master force field.
a)  Run all possible fits and verifies
u)  Run all unfinished (as marked in the master/completed_torsions
             file)
o)  Run only torsions for which there is no fit or verify directory
q)  Quit
```

```
Your choice? (a|u|o|q) a
User requested 160 tasks
```

This will begin the fitting and verification of all of the necessary torsions. This process will likely take at least fifteen minutes, and may take as long as a couple of hours, so feel free to take a break. After it finishes, take a look at the graphs for each of the fits (simply re-run the program, skip initialization in the first step, and follow the menus). If there are problems with any of the fits, any other corrections would need to be done manually. When you are content with any given fit, select the 'Declare this torsion finished' option to enter the new parameters into the final force field. After all torsions have been entered, the final force field will be done, and ready for simulation. It will be in ./myff/master/master.mff, and the structure will be in ./myff/master/master.str. Note that a sample (completed) ./myff directory is included on the CD.

Happy simulating!

Chapter 5

What's New, revisited

While Chapter 1 mentioned many of the features of FFDev with respect to a light history and background of quantum chemistry, statistical mechanics, and computer simulation, a more thorough and succinct presentation of the novelty and usefulness of the current work is called for.

High quality force fields of arbitrary forms from first principles

There is a true plethora of force fields in existence [6], and available to the academic community. We propose, and have implemented, a procedure for the rapid generation of custom, appropriate, and disposable force fields from *ab initio* data. Since the generation of a single custom force field is routine, we expect to be able to quickly test a variety of forms and parameters, and allow other users to generate force fields most suitable for their own applications. A variety of other benefits arise from our approach.

Background

Despite the large number of force fields available to researchers today, our own requirements found them all lacking in some important area or another. Specifically, the types of simulations we do require that the potential energy of a molecule as a function of the various dihedrals be as precise and accurate as possible. Many have made custom force fields for their own (specific) purposes, including ourselves [7]. The process of developing one's own force field, however, is fraught with difficulties.

Like many others before, we wished to use *ab initio* data as the basis of our force field, and to generate a classical expression that most closely reproduces the quantum chemical energy surface. In our experience, even a researcher skilled at generating custom force fields will require several weeks to several months to create a single force field. The procedure involves numerous transcriptions, and scores of objective decisions. Humans are all too error prone when it comes to transcription, and the sheer number of objective decisions that need to be made seems to defy recording and reporting (in a journal article, for example).

Our solution to the most obvious problems was to generate the force fields with software. This serves to both document the procedure we used, and to automate the creation of future force fields.

Motivation

Force fields are the foundation for any kind of simulation, and contain two parts; the form of the force field, and the parameters. The form is a function that gives the

energy as a function of the positions of the members of the system. The parameters are the actual constants that are put into the form to give it the correct behavior.

Whenever a molecular simulation is run, there are three potential sources of error. Firstly, the model used for the simulation may not accurately represent what's happening at the molecular level. Note that the model includes information about the method we use to run the simulation (molecular dynamics, Monte Carlo, etc.), as well as other simplifying assumptions, such as a mean field. Secondly, the form of the force field may not be capable of precisely representing the energies of the system. Thirdly, like the form, the parameters used in the force field may be at fault. Note that all three of these are intimately intertwined, and they cannot necessarily be separated from one another cleanly. Regardless, we make the distinction to try to understand the source of inaccuracies in simulation.

It is critical to note here that parameters for one form of a force field are not transferable to other forms. Unfortunately, all too often in simulation literature, this subtle fact is lost. Van Gunsteren [23] discusses this problem very thoroughly. In addition to what kind of terms are summed to give the total energy (such as bond stretching, etc.), the form also includes the following: Where there any cutoffs used in evaluating the columbic or van der Waals forces? What were the cutoffs? What type of cutoff was it? Were one-four interactions included, excluded, partially included? What combination rules were used for hetero-dispersion terms? Were the van der Waals forces evaluated with a Lennard-Jones potential, or an Exponential-6 potential?

To answer all of the previous questions, and the many that were omitted, one must be able to take a look at the program code used to evaluate the energy expression. In most applications, the form of the force field is 'hard wired' into the code.

Once the form of the force field is (completely) known, the parameters must be called into question. Why were the values chosen? What assumptions were made in selecting the values? There are so many questions of this nature that they can never be 'manually' enumerated in a publication of the force field.

It is our belief that the most accurate and objective source of data for parameterization of force fields is from *ab initio* calculations, which can be 'arbitrarily' exact. Once one can feel confident that the parameters for whatever form of force field they're using are 'as good as possible', simulation reveals shortcomings in either the model or form of the force field; the uncertainty about the parameters is gone. If generating force fields for arbitrary forms, and generating appropriate parameters for that form becomes routine, then rapid 'screening' of forms for a given model opens the door for rapid refinement of the form, until a suitable form for the problem at hand emerges.

Procedures and Justification

One of the fundamental requirements of all force fields is that the atoms be assigned a type, as the individual energy terms require the atoms to have some kind of name, or identity. Different research groups have come to widely divergent conclusions about how many atom types are 'necessary' to represent the range of chemical variability in a given molecule. Our solution to this (now long standing) argument was to allow every topologically and stereochemically inequivalent atom to have its own atom type (this is a slight misnomer, as enantiotopic atoms a certain distance from asymmetric stereogenic carbons are allowed to have the same atom type). This is done by using a descriptor scheme (qcodes) developed by Edgardo Garciá [36], and our own stereogenic carbon descriptor scheme.

Since we wished to parameterize our force fields from *ab initio* data, we had to make the assumption that properties of atoms (or bonds) in a large molecule can be adequately represented by atoms or bonds from smaller molecules with similar electronic and topological structure. By generating our own qcode comparison metric, we can determine which smaller (and therefore amenable to quantum chemistry calculations) molecules would be suitable proxies for atoms and bonds in the larger molecule. We have dubbed the process of generating a list of small molecules necessary to represent a large molecule 'fragmentation'. During this process, we also generate a 'map', which indicates which fragment atoms and bonds will be 'stand-ins' for atoms and bonds on the larger molecule.

Ab initio calculations are very time consuming. Our prototype work has shown that we need to generate approximately 1/6 the number of fragments as there are atoms in the molecule. If we needed to do quantum chemical calculations on all of those fragments for every force field, our productivity would be severely limited by computer time. To alleviate this problem, we have developed a quantum chemistry database, as a way to archive previous calculations. This allows the data to be reused indefinitely. It also allows the form of the force field to change arbitrarily, since the underlying data remains accessible.

Unlike conventional force fields, we are not limited to a certain portion of the periodic table for which parameters have been determined. Any atom that can be used in an *ab initio* calculation can be used in one of our force fields.

In many ways, our approach may seem like overkill. We are able to refine our form and parameters until we come 'arbitrarily close' to exactly reproducing the *ab initio* potential energy surface. Conventional wisdom declares that, while the intra-molecular interactions may be important, the inter-molecular interactions completely dominate the bulk behavior. (For clarity, we use the common vernacular that considers bonded interactions to be intra-molecular, and non-bonded interactions to be inter-molecular; even though non-bonded interactions occur between atoms in the same molecule.) The topic of how to get inter-molecular interactions (columbic and van der Walls) from *ab initio* calculations is one of very active research right now,

and we haven't begun to tackle it, instead opting to concentrate on the intra-molecular potential. Why such precision?

The simplest retort to this question is: Why not? We have found it relatively easy to get arbitrarily good intra-molecular parameters, and, though the uncertainty is much less than the uncertainty in inter-molecular parameters, the precision is available to keep up with future advancements. Additionally, some models use a mean field in a vacuum, and require only intra-molecular parameters; these types of simulations can benefit greatly from the additional precision.

By feeling confident that our intra-molecular potential is accurate, we can turn our attention to the inter-molecular portions of the force field. Since we can tune how we get intra-molecular parameters from our database, and routinely generate new force fields, we are able to prototype, test, and refine our force fields rapidly. Polarizable charge models are gaining much popularity in the current literature. Parameterization of these models is nearly impossible to do from experiment, which means researchers must instead rely on quantum chemical calculations. **Atomic** (atom centered) polarizability is a fine concept (as are point charges on nuclei), but there exists no quantum mechanical operator for either, unless the entire molecule is a single atom. One can envision a great number of ways to do this, and we look forward to being able to join the current researchers in trying to solve this problem.

Conclusions

The ability to rapidly create many force fields of arbitrary form, from a well defined procedure, is a great boon to anybody interested in doing molecular simulation. It is well accepted that different force fields are 'better' at some kinds of simulations than others. Imagine rapidly generating twenty different kinds of force fields for a particular task, and evaluating the results of simulations using each of them. This would allow a person doing simulations to very quickly find the most appropriate force field for their current problem.

Many perceived shortcoming of existent force fields are resolved by generating a 'disposable' force field when you need it. Firstly, the entire procedure is fully documented (via the source code), and anybody can reproduce the results. Secondly, since generation of new force fields is routine, we are freed to concentrate our efforts on improving the form of our force fields. Thirdly, by assigning a different atom type for every unique atom in a system, our force fields are both flexible, and ultimately appropriate for whatever the current task may be. Finally, we can reach 'arbitrary' precision, provided the property in question can be treated and solved in a quantum mechanical calculation.

Chapter 6

Wrapping it all up

In this chapter, we will wrap up all of the loose ends left during the previous chapters. Specifically, we will discuss where the software and other supplementary materials can be found, what our accomplishments are, and what science we hope to promote in the future based on this work.

Supplementary materials

The supplementary materials for the presented work are in digital format, and I have chosen two independent places to 'officially' archive it. Firstly, if you have an 'official' copy of this thesis, there will be an attached CD. The CD has four files in the root directory.

The 'ff' directory contains all of the code in the project, as well as compiled executables for a Linux 2.2x/686 kernel, though the code should be easy to recompile for your own system. There is an ff1.fff file, which is referenced in Chapter 4, and is an 'almost finished' force field. There is a truncated *ab ititio* database in the 'qdb' directory, as it is required by the demo. Finally, there is a 'myff' directory, which is created if you follow the last step in the tutorial, and have access to Matthew Glaser's simulation code.

All of the data necessary to follow the tutorial in Chapter 4 is on that CD. If you have come by this document by other means, you can find a gzipped ISO of the CD at ffdev.sourceforge.net. The 'thesis final' release of the software will be available there, as well as any 'current' releases. ffdev.sourceforge.net will be the permanent home of the project, so if you are interested in contributing to the project, or know of someone that would be, please visit that site.

Accomplishments

This work has made several ground breaking advances in the preparation of classical interaction potentials for atomistic simulation. First and foremost, we have demonstrated that it is possible to completely automate the process of taking a single (potentially large) molecule, and create from scratch (*ab initio* data) all of the data necessary to create a force field completely from first principles. We have further demonstrated the re-use of expensive *ab initio* quantum chemical calculations, and made the 'data mining' necessary for this task simple for the end user. These two

tasks serve as a proof of concept that creation of purely *ab initio* force fields is possible.

By casting our force field into a form suitable for Boulder Model binding site simulations, we have shown two things. Firstly, we have demonstrated a practical application of the automated force field creation. Secondly, we have provided further evidence of the usefulness of the simple Boulder Model mean field approach for determining both the sign and magnitude of macroscopic polarization.

Finally, and perhaps most importantly, we have opened the door to a completely new approach to the refinement of force fields. The focus for improvement of a force field of a given form can now easily be treated as a problem of how we parameterize it (from fundamentally sound input), instead of the historical approach of tweaking parameters without justification.

Future work

We have by no means created the be all and end all of force field creation. In fact, perhaps our biggest accomplishment is in the number of new research directions we have created. As mentioned in Chapter 2, our force field is by no means derived strictly from *ab initio* data, though the final torsion fitting serves to sweep the inadequacies in the empirical parameters into the torsion terms. Several very interesting possibilities arise with our new methodology.

We have used incredibly generic bond stretching and angle bending parameters in our own force field. We consider this a reasonable approximation for our purposes, as they have little bearing on the overall shape of the molecule. In order to get classical force fields that are capable of reproducing infrared spectra of molecules, we would need much more sophisticated parameterization. Firstly, we would need to extract second derivatives of the energy with respect to nuclear motion for the relevant parameters. This is in fact data easily accessible in some kinds of *ab initio* calculations, so would fit very well into our data extraction approach. Secondly, we would need to add coupling terms, another task that lends itself well to extraction from *ab initio* calculation data.

Peter Tieleman, a membrane biophysicist at the University of Calgary told me several years ago: ”If you want to do solution phase simulations, quantum mechanics is practically useless …” While his statement may be true to an extent, we remain optimistic that the ‘real’ answer lies in understanding inter-molecular interactions at the quatum chemical level. To this end, we have several ideas for getting arbitrarily precise parameters for either the Lennard Jones potential we’re currently using, or for parameterizing any other form of intermolecular potential. There is also currently work being done on doing *ab initio* calculations in ‘effective solvent fields’, though we feel that this (semi-empirical) approach suffers from the same limitations as other semi-empirical approaches. This is an area that would be very interesting to pursue in the future.

Finally, one of the most exciting new fields of work in force field development involves the usage of polarizable charge models. These models all allow the charges to either float off of the nuclei, or allowing the charge to redistribute itself within the same molecule. Regardless of the form of force field that we use, our methodology for mapping from small fragments onto large parent compounds should prove ubiquitous for this parameterization.

In closing …

Force field creation need not be an activity limited to the few experts in the world. What started as a simple request grew into a suite of programs suitable for the simple, rapid, on-demand creation of strictly appropriate force fields for arbitrarily large molecules. Admittedly, it is only a beginning; yet we believe our unique approach, once fully realized, could revolutionize the way force fields are created, refined, and used today, and for the foreseeable future.

Bibliography

1) a) Walba, D. M., Slater, S. C., Thurmes, W. N., Clark, N. A., Handschy, M. A., Supon, F. J., *J. Am. Chem. Soc.*, **1986** 108, 5210; b)Walba, D. M., Vohra, R. T., Clark, N. A., Handschy, M.,A., Xue, J., Parmar, D. S., Lagerwall, S. T., Skarp, K., *J. Am. Chem. Soc.*, **1986**, 108, 7424; c) Walba, D. M., Clark, N. A., c) Walba, D. M., Clark, N. A., *Ferroelectrics*, **1988**, 84, 65, e) Walba, D. M., Razavi, H. A., Clark, N. A., Parmar, D. S., *J. Am. Chem. Soc.*, **1988**, 110, 8686.

2) a) Bartolino, R., Doucet, J., Durand, G., *Ann. Phys. (Paris)*, **1978**, 3, 389, b) Yoshizawa, A., Kikuzaki, H., Fukumasa, M., *Liq. Cryst.*, **1995**,18, 351.

3) a) Maier, W., Saupe, A. Z., *Naturf.*, **1958**, A13, 564, b) Maier, W., Saupe, A.Z., *Naturf.*, **1959**, A14, 882.

4) Allen, M. P., Tildesley, D. J. *Computer Simulation of Liquids*, Oxford Science Publications: Oxford, New York, 1994.

5) Meyer, R. B., "Structural Problems in Liquid Crystal Physics" in *Molecular Fluids*, Balian, R, Weill, G. , Eds.; Gordon and Breach, London, 1976. This paper presents a written account of work presented at the "Summer School of Theoretical Physics", held in Les Houches, France, France, August, 1973. Meyer also presented this idea, along with preliminary experimental results, in a famous talk at the Vth International Liquid Crystal Conference in Stockholm, in 1974.

6) The field of molecular dynamics has exploded in recent years, and a great variety of software has been developed for either generating force fields for molecular dynamics simulations, or for actually running the simulations. There are many more than listed here, but a partial list includes: AMBER[a], BRUGEL[b], DEDAR[c], CHARMM[d], EGO[e] ENCAD[f], FOCUS[g], GROMACS[h], GROMOS[i], MOIL[j], NAMD[k], POLARIS[l], UHBD[m], X-PLOR[n] YASP[o], CHARMm AND DISCOVER[p], AND SYBYL[q]. a) Pearlman, D. A., Case, D. A., Caldwell, J. W., Ross, W. S., Cheatham III, T. E., DeBolt, S., Ferguson, D., Seibel, G., Kollman, P., *Comput. Phys. Commun*,**1995**, 91, 1, b) Delhaise, P., van Belle, D., Bardiaux, M., Alard, A., Hamers, P., van Cutsem, E., Wodak, S. J., *J. Mol. Graphics*, **1985**, 3, 116, c) Carson, M., Hermans, J., *Molecular Dynamics and Protein Structure*, Hermans, J., Ed.; University of North Carolina, Chapel Hill, 1985, pp. 165-166, d). Brooks, B. R, Bruccoleri, R. E., Olafson, B. D., States, D. J., Swaminathan, S., Karplus, M., *J. Comput. Chem.* , **1983**, 4, 187, e) Eichinger, M., Grubmüller, H., Heller, H., *User Manual for EGO-VIII, Release 1.0*, Universität München: München, 1995, f) Levitt, M., Hirshberg, M., Sharon, R., Daggett, V., *Comput. Phys. Commun.,* **1995**, 91, 215, g) Lemon, A. P., Dauber-Osguthorpe, P., Osguthorpe, D. J., *Comput. Phys. Commun.*, **1995**, 91, 97, h) Berendsen, H. J. C., van der Spoel, D., van Drunen, R., *Comput. Phys. Commun.*, **1995**, 91, 43 i) van Gunsteren, W. F., Billeter, S. R., Eising, A. A., Hünenberger, P. H., Krüger, P., Mark, A. E., Scott, W. R. P., Tironi, I. G., *Biomolecular simulation: The gromos96 Manual and User Guide*,

Hochschulverlag an der ETH Zürich: Zürich, 1996, j) Eiber, R., Roitberg, A., Simmerling, C., Goldstein, R., Li, H., Verkhivker, G., Keasar, C., Zhang, J., Ulitsky, A., *Comput. Phys. Commun.*, **1995**, 91, 159, k) Nelson, M., Humphrey, W., Kufrin, R., Gursoy, A., Dalke, A., Kale, L., Skeel, R., Schulten, K., *Comput. Phys. Commun.*, **1995**, 91, 111, l) Lee, F. S., Chu, Z. T., Warshel, A., *J. Comput. Chem.*, **1995**, 14, 161 **1993** m) Madura, J. D., Briggs, J. M., Wade, R. C., Davis, M. E., Luty, B. A., Ilin, A., Antosiewicz, J., Gilson, M. K., Bagheri, B., Scott, L. R., McCammon, J. A., *Comput. Phys. Commun.*, **1995**, 91, 57, n) Brünger A. T., *X-PLOR: A System for X-ray Crystallography and NMR*, Howard Hughes Medical Institute and Yale University: New Haven, 1996, o) Müller-Plathe, F., *Comput. Phys. Commun.*, **1993**, 78, 77, p) Accelrys, Burlington, MA 01803, 1997, q) Tripos Inc., St. Louis, MO 63144, 2002.

7) a) Glaser, M. A., Clark, N. A., Garciá, E., Walba, D. M., *Spectrochimica Acta Part A*, **1997**, 53, 1325, b) Garciá, E., Glaser, M. A., Clark, N. A., Walba, D. M., *J. Mol. Struct. (Theochem)*, **1999**,464, 39, c) Glaser, M. A., Clark, N. A., submitted to *Liquid Crystals*, **2002**.

8) Tsaparlis, G., *Chemistry Education: Research and Practices in Europe*, **2001**, 2(3), 203.

9) Heisenberg, W.Z., *Zeitschrift der Physik*, **1925**, 33, 879.

10 Born M., Jordan, P., *Zeitschrift der Physik*, **1925** ,34, 858.

11) a) Schrödinger, E., *Annals der Physik*, **1926**, 79, 361, b) Schrödinger, E., *Annals der Physik*, **1926**, 79, 489, c) Schrödinger, E., *Annals der Physik*, **1926**, 79, 734.

12) Dirac, P. A. M., *Proc. Roy. Soc (London) A*, **1929**, 123, 714.

13) In 1998, John Pople won the nobel prize in Chemistry "for his development of computational methods in quantum chemistry." For a complete bibliography of John Pople's work from 1950 to 1989, see *Int. J. Quant. Chem.*, **1990**,38, 355-371.

14) No published reference can be found, but it is known that Gordon E. Moore (one of the co-founders of Intel) first made his statement in 1965.

15) Intel Corporation. Moore's Law. http://www.intel.com/research/silicon/mooreslaw.htm (accessed May 2002).

16) Biggus, J. Sketching the History of Statistical Mechanics and Thermodynamics (From about 1575 to 1980). http://history.hyperjeff.net/statmech.html (accessed May 2002).

17) Circa 150 BC, Hero of Alexandria, *Pneumatics*. For an english translation of this work, see Woodcroft, B. THE PNEUMATICS OF HERO OF ALEXANDRIA. http://www.history.rochester.edu/steam/hero/ (accessed May 2002).

18) Waterston, J. J., *Thoughts on the Mental Functions*, self-published, 1843.

19) a) Metropolis, N., Rosenbluth, A.W., Rosenbluth, M. N., Teller, A. H., Teller, E. *J. Chem. Phys.*, **1953**, 21, 1087, b) Rosunbluth, M. N. Rosenbluth, A. W., *J. Chem. Phys.*, **1954**, 21, 881, c) Alder, B. J. Wainwright, T. E., *J. Chem. Phys.*, **1959**, 31, 459.

20) Gibson, J. B., Godland, A. N., Milgram, M., Vineyard, G. H., *Phys. Rev.*, **1960**, 120, 1229.

21) Allinger, N. L., *J. Am. Chem. Soc.*, **1977**, 99, 8127.

22) a) Maple, J. R., Hwang, M.-J., Tosckfisch, T. P., Dinur, U., Waldman, M., Ewig, C. S., Hagler, A.T., J. Comput. Chem, **1994**, 15, 162, b) Famulari, A., Specchio, R., Sironi, M., Raimondi, M., *J. Chem. Phys.*, **1998**, 108, 3296, c) Mahoney, M. Jorgensen, W., *J. Chem. Phys.*, **2000**, 112, 8910.

23) Van Gunsteren, W. F., Mark, A. E., *J. Chem. Phys.*, **1998**, 108, 6109.

24) Gaussian 98, Revision A.7, Frisch, M. J., Trucks, G. W., Schlegel, H. B., Scuseria, G. E., Robb, M. A., Cheeseman, J. R., Zakrzewski, V. G., Montgomery, Jr., J. A.., Stratmann, R. E., Burant, J. C., Dapprich, S., Millam, J. M., Daniels, A. D., Kudin, K. N,. Strain, M. C, Farkas, O., Tomasi, J., Barone, V., Cossi, M., Cammi, R., Mennucci, B., Pomelli, C., Adamo, C., Clifford, S., Ochterski, J., Petersson, G. A., Ayala, P. Y., Cui, Q., Morokuma, K., Malick, D. K., Rabuck, A. D., Raghavachari, K., Foresman, J. B., Cioslowski, J., Ortiz, J. V., Baboul, A. G., Stefanov, B. B., Liu, G., Liashenko, A., Piskorz, P., Komaromi, I., Gomperts, R., Martin, R. L., Fox, D. J., Keith, T., Al-Laham, M. A., Peng, C. Y., Nanayakkara, A., Gonzalez, C., Challacombe, M., Gill, P. M. W., Johnson, B., Chen, W., Wong, M. W., Andres, J. L., Gonzalez, C., Head-Gordon, M., Replogle, E. S. Pople., J. A, Gaussian, Inc., Pittsburgh PA, 1998.

25) a) Becke, A. D., *J. Chem. Phys.*, **1993**, 98, 5648- 5652, b) Pople, J. A., Head-Gordon, M., Fox, D. J., Raghavachari, K. Curtiss, L. A., *J. Chem. Phys.*, **1989**, 90, 5622 c) Curtiss, L. A., Jones, C., Trucks, G. W., Raghavachari, K. Pople, J. A., *J. Chem. Phys.*, **1990**, 93, 2537.

26) Foresman, J. B. Frisch, Æ. *Exploring Chemistry with Electronic Structure Methods, Second Edition*, Gaussian Inc.: Pittsburgh, PA, 1996.

27) Klyne, W., Prelog, V., *Experientia*, **1960**, 16, 521.

28) Mayo, S. L., Olafson, B. D., Goddard III, W. A., *J. Phys. Chem.*, **1990**, 94, 8897.

29) Breneman, C. M., Wiberg, K. B., *J. Comp. Chem.*, **1990**, 11, 361.

30) a) Price, M. L. P., Ostrovsky, D., Jorgensen, W. L., *J. Comp. Chem.*, **2001**, 22, 1340, b) Rizzo, R. C., Jorgensen, W. L., *J. Am. Chem. Soc.*, **1999**, 121, 4827, c)

Briggs, J. M., Nguyen, T. B.,. Jorgensen, W. L, *J. Phys. Chem.*, **1991**, 95, 3315, d) Jorgensen, W.L. et al., *J. Comput. Chem.*, **1993**, 14, 206.

31) a) Siepmann J.I., Karaborni S., Smit B., *Nature*, **1993**, 365, 330 b) Jorgensen W.L., Madura J.D., Swenson C.J., *J. Am. Chem. Soc.*, **1994**, 106, 6638.

32) a) Hourihan, M. O'Reilly Network: The Sanctity of Elements, or Why You Shouldn't be Double-clicking in a <textarea> [May 3, 2002]. http://www.oreillynet.com/pub/a/javascript/2002/05/03/megnut.html (accessed May, 2002), b) Rosenfeld, L., Morville, P., *Information Architecture for the World Wide Web: Designing Large-scale Web Sites*, O'Reilly & Associates: Sebastopol, California, 1998, c) Rational Software. http://www.rational.com (accessed May 2002), d) R. S. Pressman and Associates, Inc. RSP&A Project Planning and Management. http://www.rspa.com/spi/project-mgmt.html (accessed May 2002).

33) CPAN. CPAN/ports. www.perl.com/CPAN/ports/ (accessed May 2002).

34) Cozens, S. perl.com: Ten Perl Myths [Feb. 23, 2000]. http://www.perl.com/pub/a/2000/01/10PerlMyths.html (accessed May 2002),

35) See, for example, some of the programs from reference 6.

36) 1) Garciá, E., MATCHEM: A symbolic model for the computer representation and manipulation of chemical structures and reactions, PhD Thesis, Instituto de Qui'micha, Universidade de Brasilia, Brasilia DF, Brasil, 1994, b) Garciá, E., Instituto de Qui'micha, Universidade de Brasilia, Brasilia DF, Brasil, personal communications, c) Garciá, E., submitted to *J. Chem. Inf. and Comp. Sci.*, **2002**.

37) a) Perldoc.com. perlXStut. http://www.perldoc.com/perl5.6.1/pod/perlxstut.html (accessed May 2002), b) Perldoc.com. perlxs. http://www.perldoc.com/perl5.6.1/pod/perlxs.html (accessed May 2002).

38) a) Cahn, R.S., Ingold, C.K. Prelog, V., *Angew. Chem.*, **1966**, 78, 413, b) Cahn, R.S., Ingold, C.K. Prelog, V., *Angew. Chem. Internat. Ed. Eng.*, **1966**, 5, 385; c) Prelog, V., Helmchen, G., *Angew. Chem.*, **1982**, 94, 614-631, d) Prelog, V., Helmchen, G., *Angew. Chem. Internat. Ed. Eng.* , **1982**, 21, 567.

39) Booch, G., Rumbaugh, J., Jacobson, I. *The Unified Modeling Language User Guide*, Addison-Wesley: Reading, Massachusetts, 1999.

40) Wall, L., Christiansen, T., Orwant, J., *Programming Perl, 3rd Edition*, O'Reilly & Associates: Sebastopol, California, 2000.

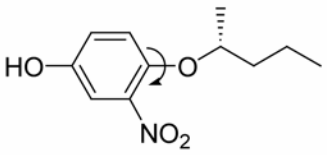
41) The Free Software Foundation. Licenses – GNU Project – Free Software Foundation (FSF). http://www.gnu.org/licenses/licenses.html (accessed May 2002).

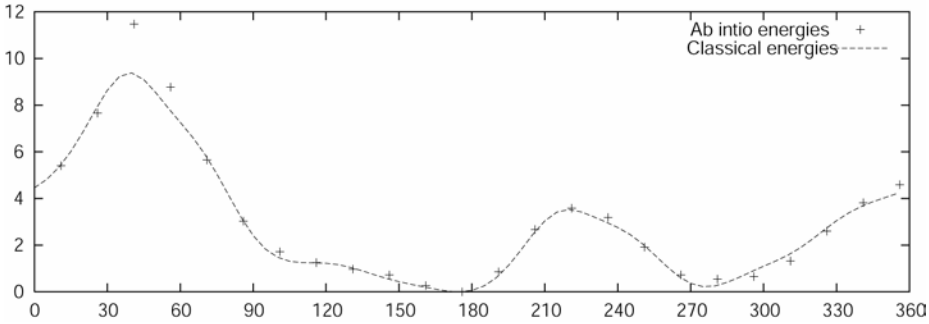
Appendix A

This appendix contains the actual fitting data for all of the dihedrals that went into the final force field, as well as the actual parameters of the cosine series we use to reproduce the ab initio torsional profile.

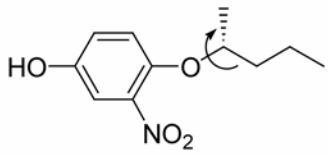
Compound 1

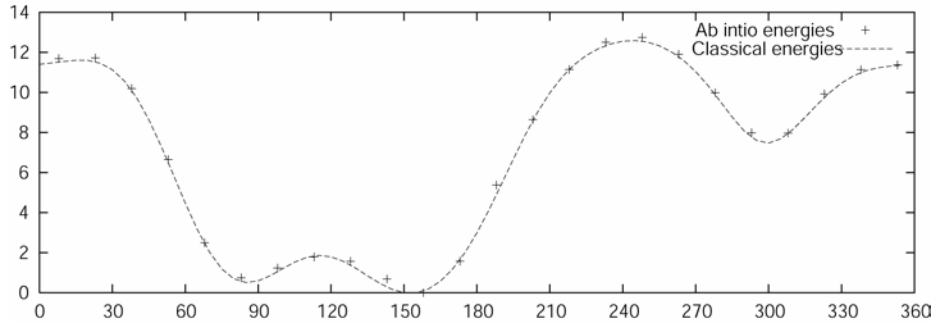
C11H15NO4-0_5-6-15-16
-8.196 15.993 -21.255 -31.435 129.370 26.806 -96.932
Classical energy offset: 0.0852031



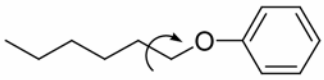


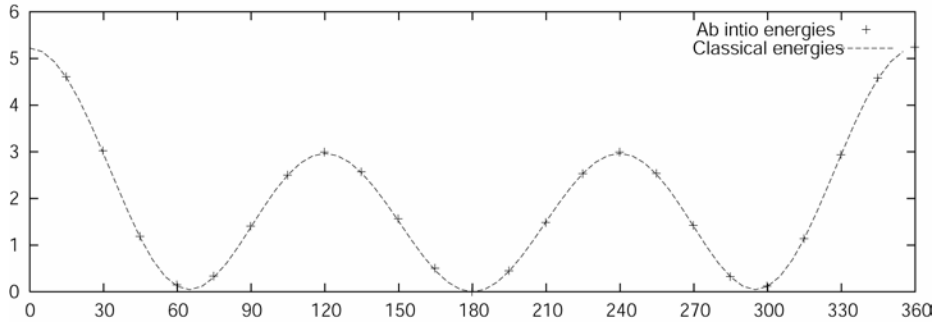
C11H15NO4-0_6-15-16-17
-2.521 -2.112 27.228 13.334 -47.415 -9.295 21.565
Classical energy offset: 0.130546



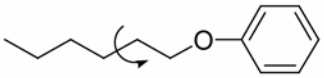


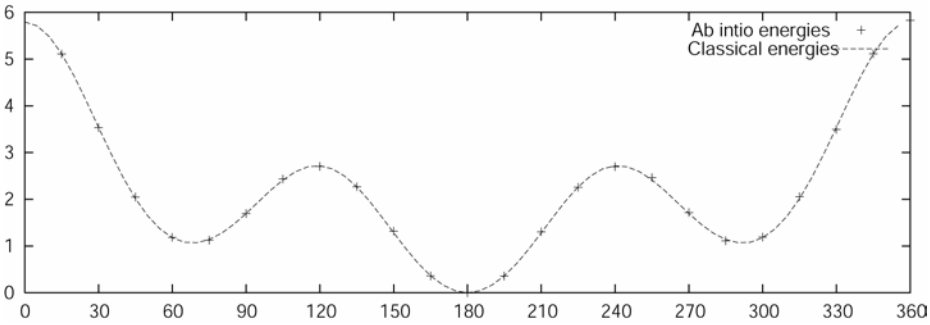
C12H18O-1_12-13-14-15
-0.429 -4.752 0.170 7.701 1.889 -0.372 -0.875
Classical energy offset: 0.0174484



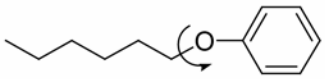


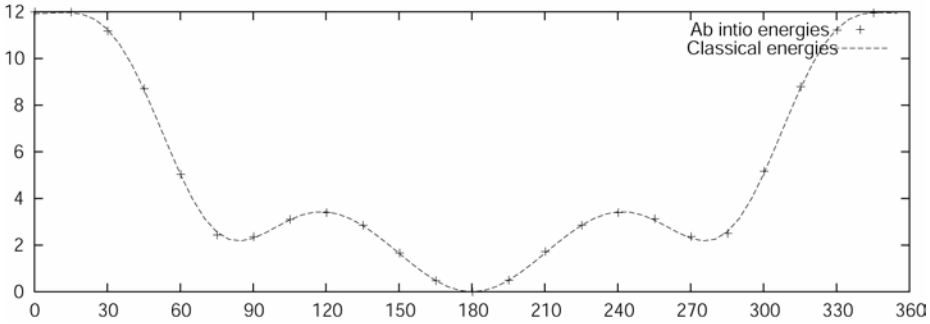
C12H18O-1_13-14-15-16
-0.138 -2.789 1.310 4.761 -1.586 0.907 1.459
Classical energy offset: 0.00544838



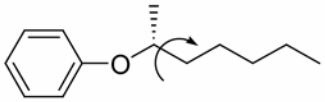


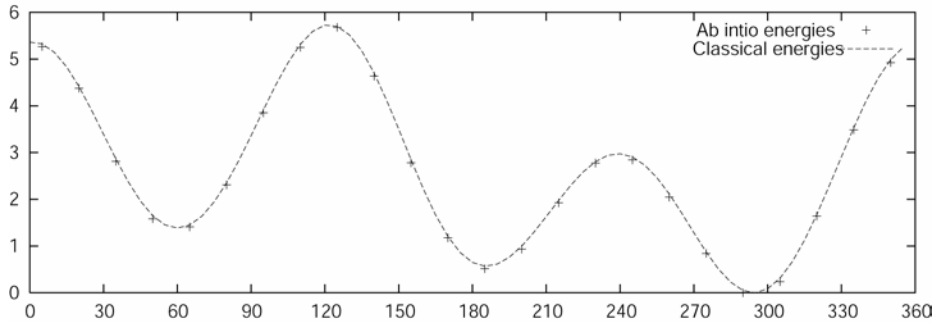
C12H18O-1_4-12-13-14
0.068 -4.233 2.736 10.958 -1.124 -6.567 -3.351
Classical energy offset: 0.0104484



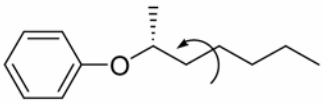


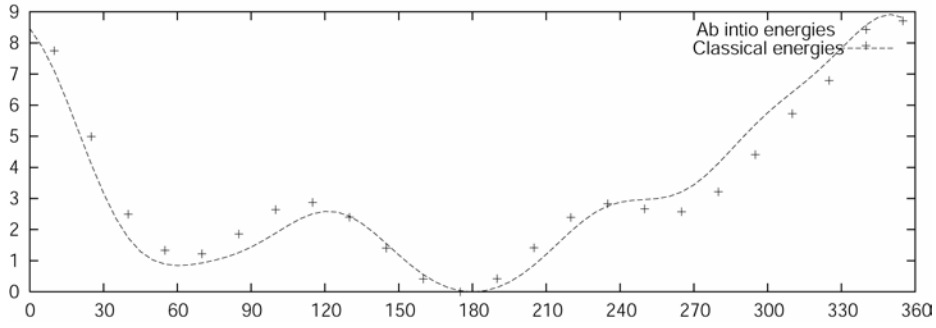
C13H20O-0_12-13-14-15
0.309 3.525 -18.230 -3.003 54.961 1.140 -36.870
Classical energy offset: -0.0511571



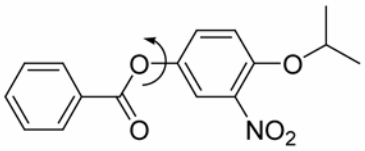


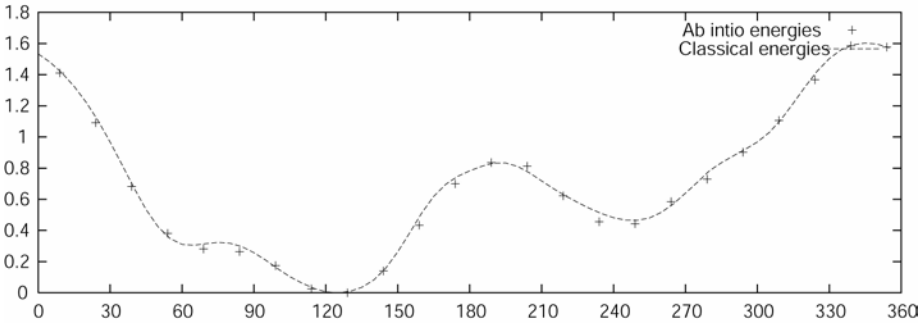
C13H20O-0_13-14-15-16
1.023 -2.371 2.089 0.376 -7.437 3.638 5.574
Classical energy offset: 0.129162



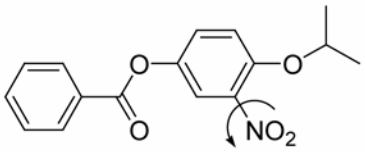


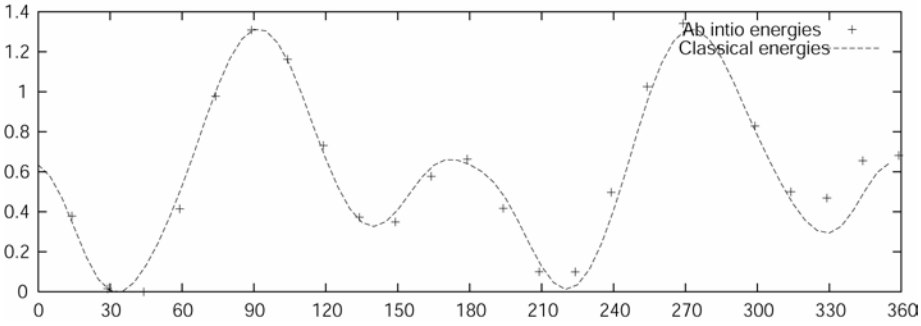
C16H15NO5-0_12-14-15-16
62.395 19.721 -9.194 20.015 8.417 -10.874 -2.108
Classical energy offset: -0.0139798



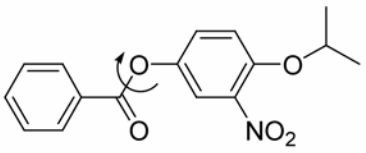


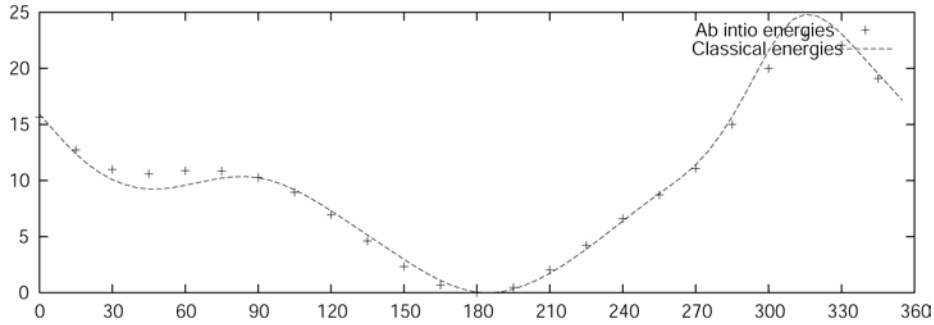
C16H15NO5-0_16-17-24-25
-97.394 -0.000 89.828 0.000 20.218 -0.000 -16.240
Classical energy offset: 0.0200807



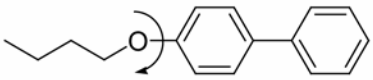


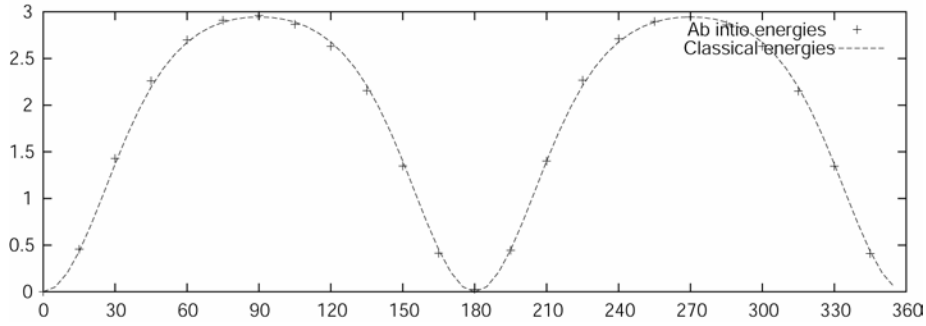
C16H15NO5-0_5-12-14-15
71.287 0.722 -14.007 -1.425 7.286 2.641 -2.044
Classical energy offset: -0.117573



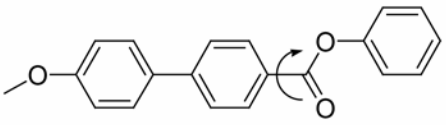


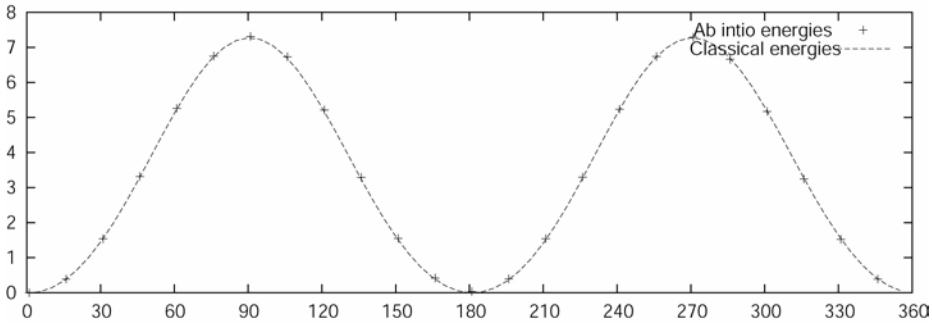
C16H18O-0_3-4-22-23
2.935 0.000 -1.282 -0.000 -2.115 0.000 -1.380
Classical energy offset: -0.00255197



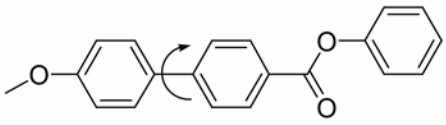


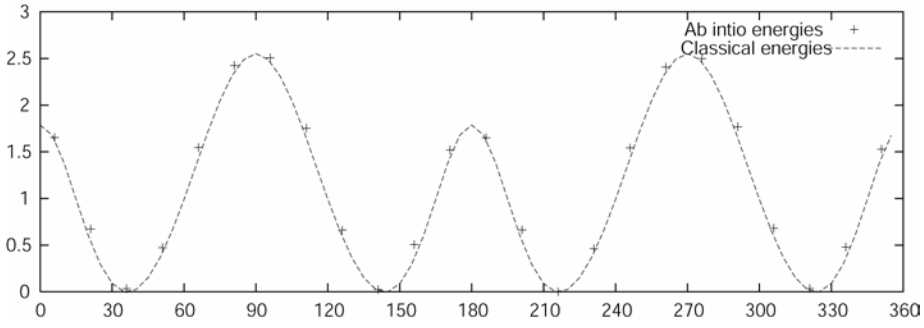
C20H16O3-0_13-14-21-22
43.527 -0.000 -75.258 0.000 -8.680 -0.000 5.587
Classical energy offset: 0.0173037



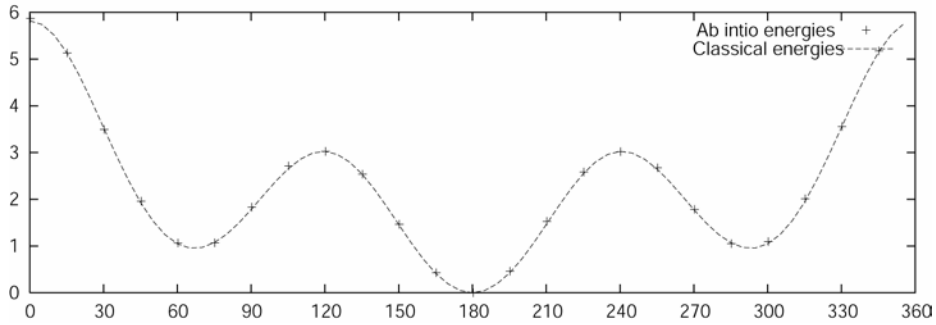


C20H16O3-0_2-1-11-12
-2.526 0.675 -8.070 -2.021 6.211 1.311 -3.533
Classical energy offset: 0.0724218

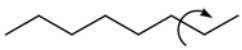


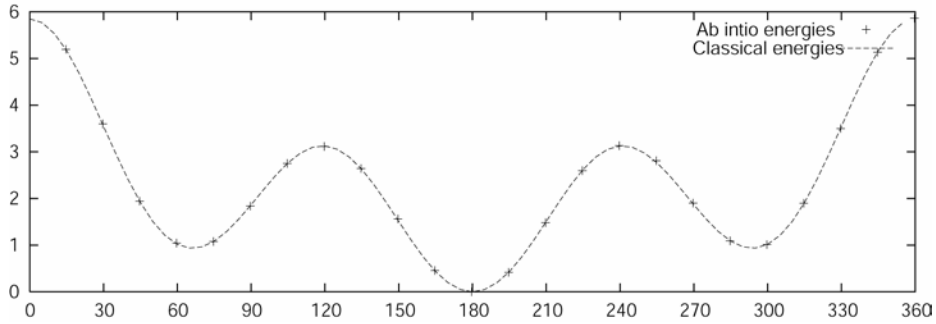


C7H16O-1_4-5-6-7
1.800 -3.434 1.272 5.663 -1.270 0.659 1.106
Classical energy offset: 0.02

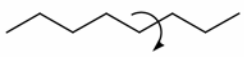


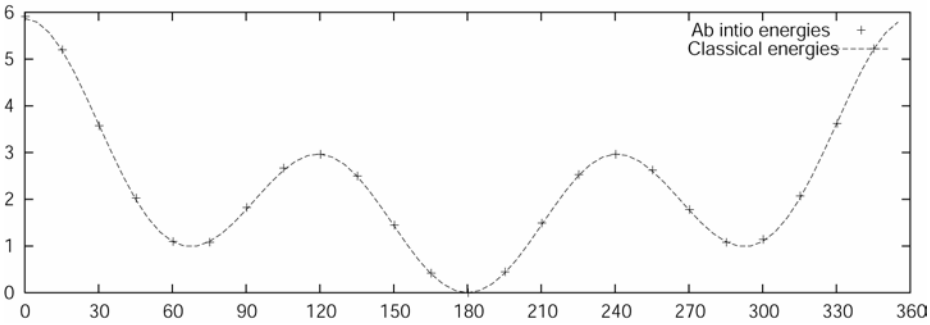
C8H18-0_1-2-3-4
1.871 -3.646 0.937 6.003 -0.667 0.554 0.775
Classical energy offset: 0.005



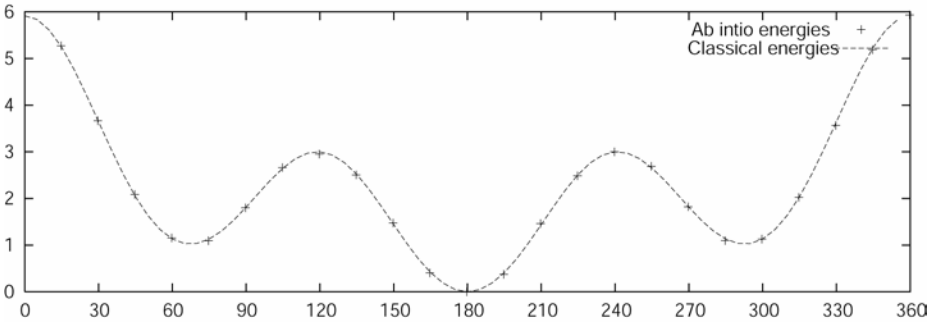


C8H18-0_2-3-4-5
1.792 -3.311 1.202 5.595 -0.966 0.629 0.900
Classical energy offset: 0.015

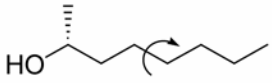


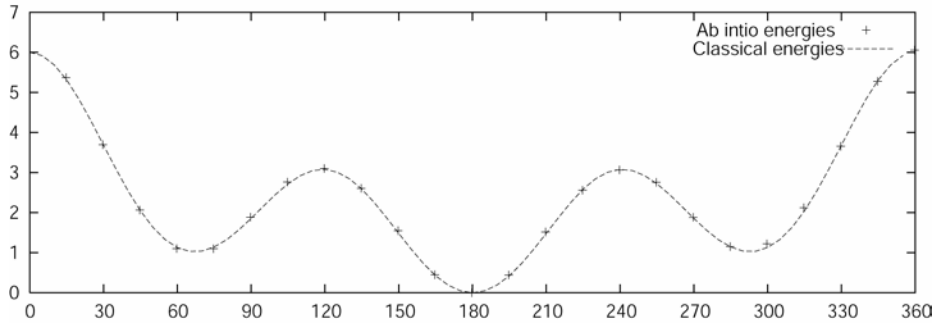


C8H18-0_3-4-5-6
1.809 -3.322 1.225 5.656 -1.127 0.602 1.024
Classical energy offset: -0.005

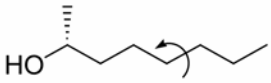


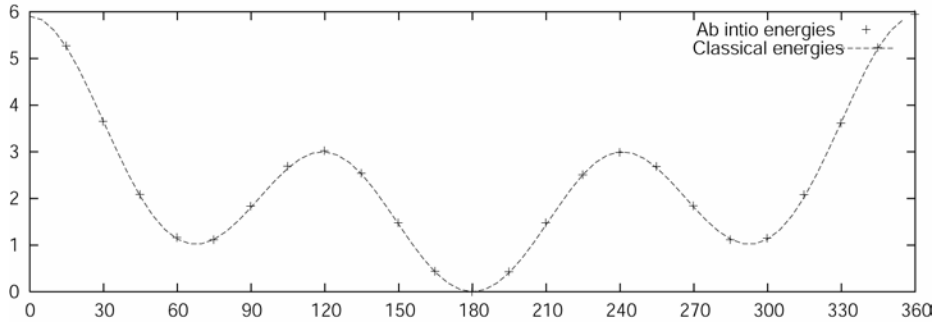
C8H18O-0_4-5-6-7
1.714 -3.440 1.190 5.795 -1.045 0.628 0.982
Classical energy offset: 0.0188579



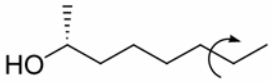


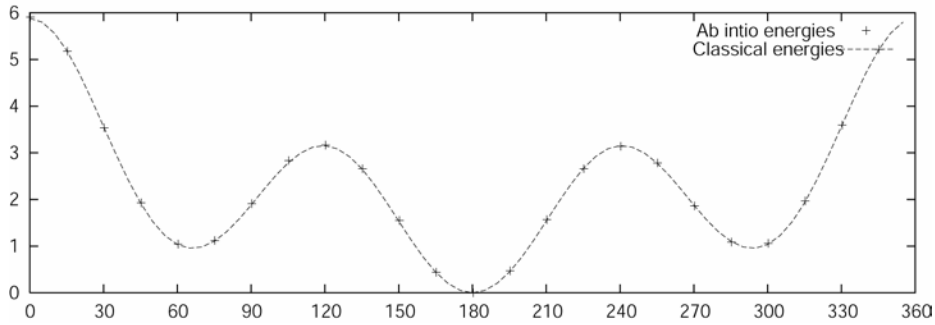
C8H18O-0_5-6-7-8
1.669 -3.330 1.219 5.684 -1.085 0.578 0.986
Classical energy offset: 0.0178579





C8H18O-0_6-7-8-10
1.730 -3.636 0.985 5.902 -0.716 0.665 0.781
Classical energy offset: 0.00985791





Appendix B

This appendix includes a graphical summary of the fragmentation of all three compounds, as well as a graphical summary of the atom and bond mapping for compound 1.

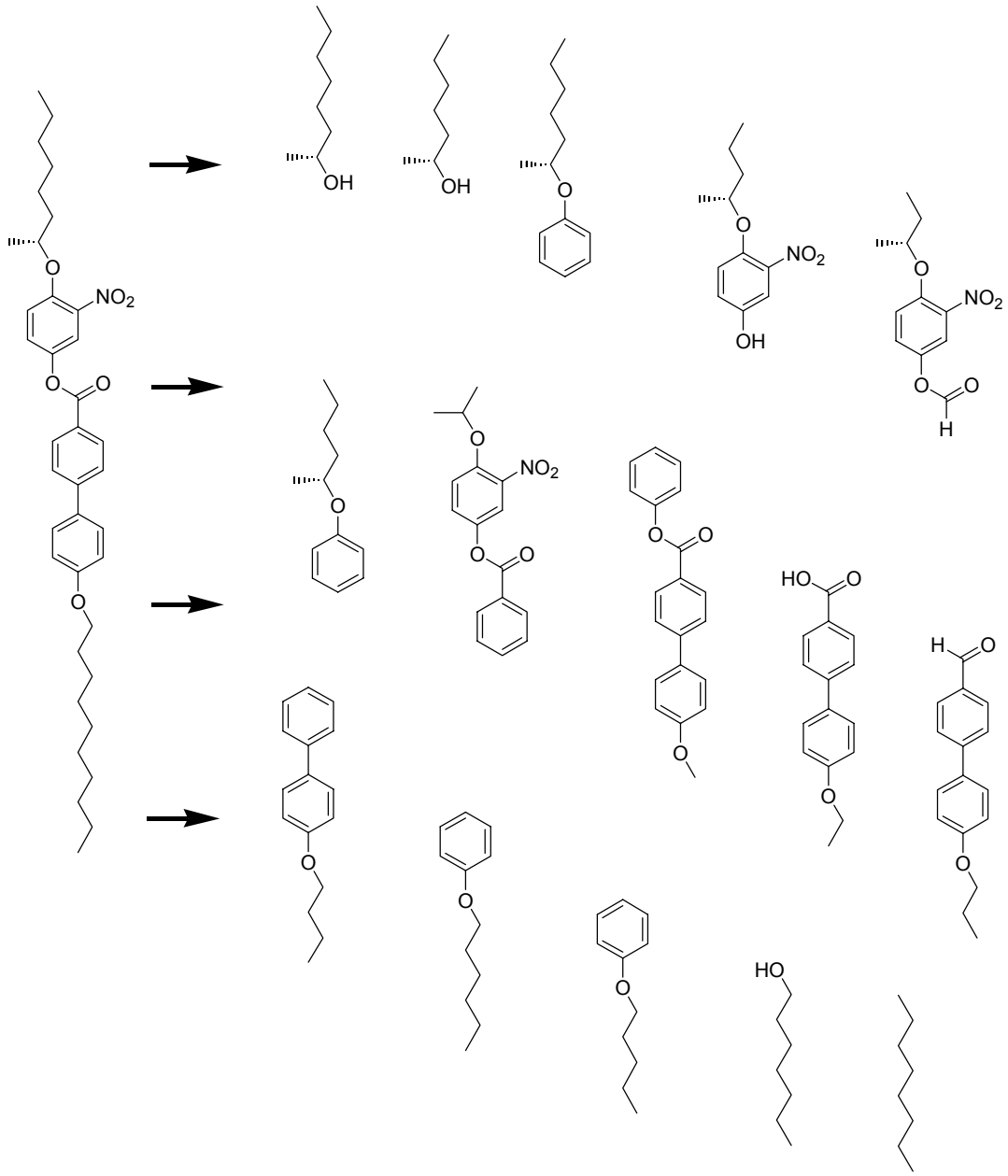
Note that in all of the mappings, the standard skeletal structure is presented, with omitted hydrogens in almost all cases. The actual atom and bond maps contained descriptions for mapping between all atoms and bonds.

In the Parent to Fragment Atom Mappings, all hydrogens mapped with the carbon they were attached to. Additionally, some selections are with a box (when there are multiple atoms that go to a single fragment), and some selections are directly from a parent atom to a fragment atom.

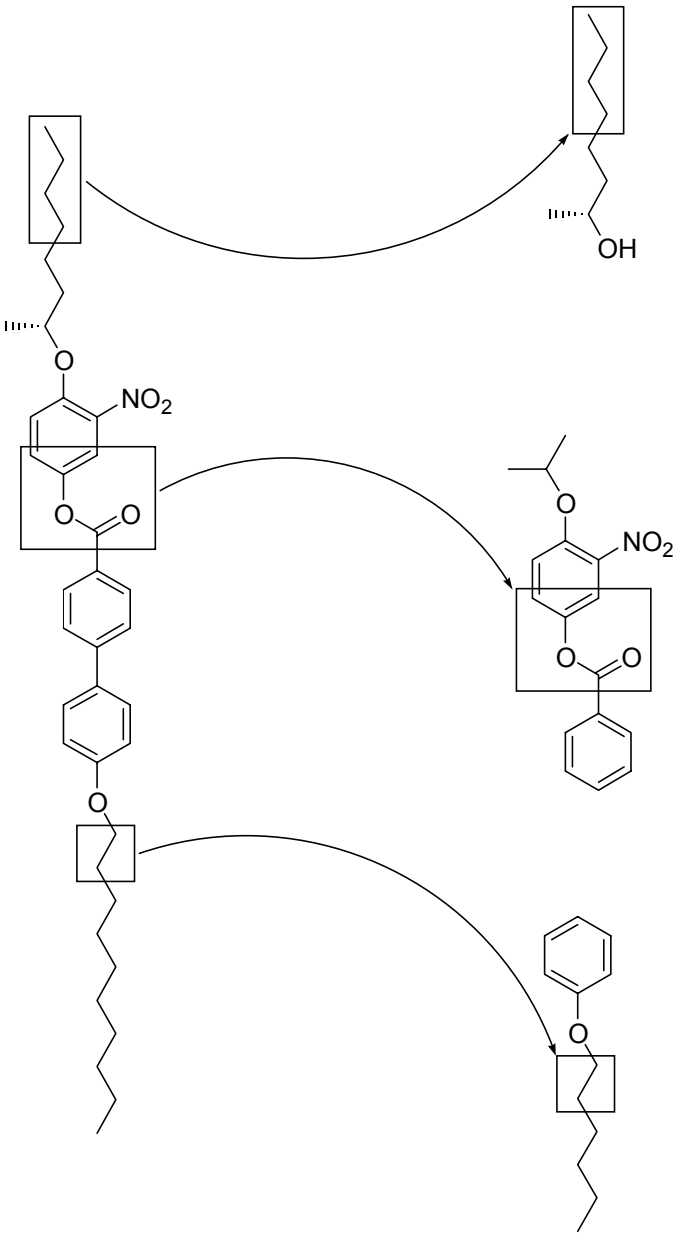
In the Parent to Fragment Bond Mappings, only bonds between heavy atoms are illustrated, for clarity. Also, they are matched by letter, instead of using arrows to indicate the correlation.

After the initial graphical presentation, the relevant sections of the atom and bond mapping sections of the output from qdb_check are included, as an example. The fragments will be in the sample database included on the CD. Remember, the atoms and bond numbering starts from 0, instead of 1, so depending on what program you use to visualize the molecules, you may need to add one to the values in the atom and bond maps.

Fragmentation of Compound **1**



Parent to Fragment Atom Mappings for Parent compound 1, Part 1

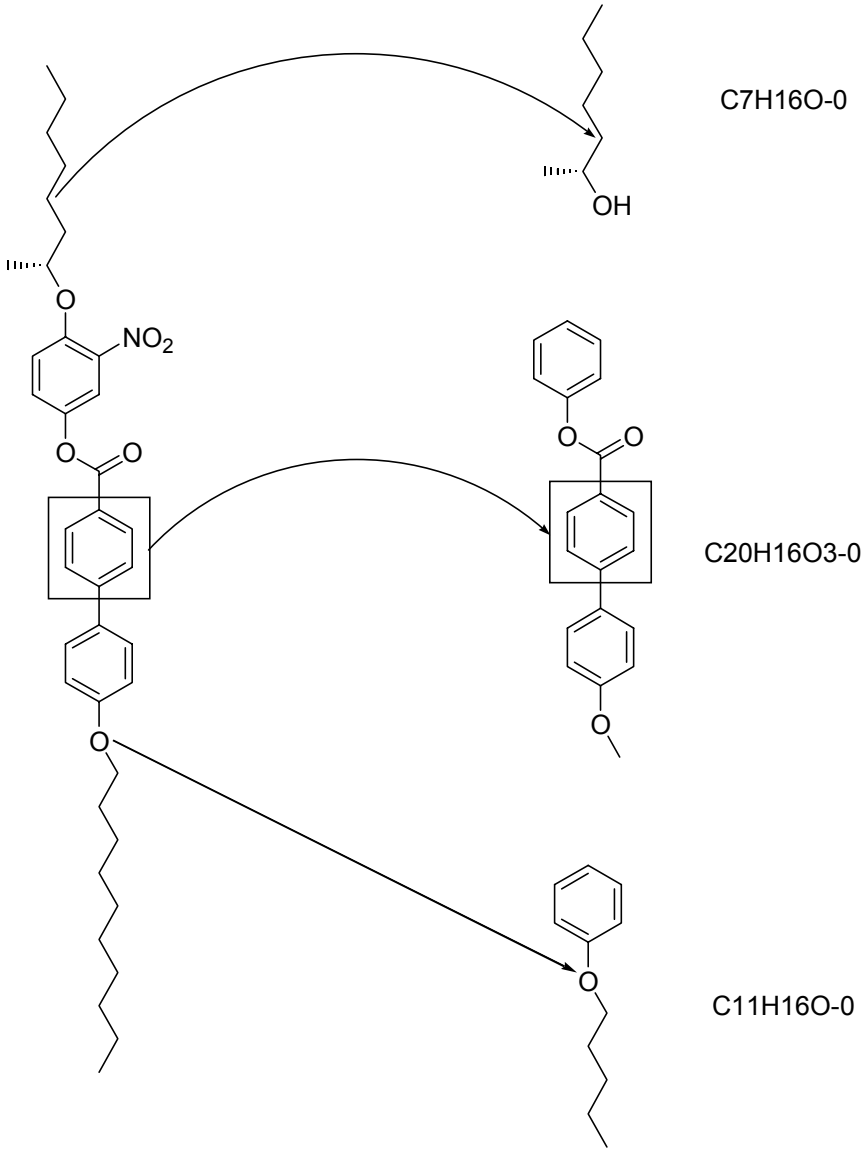


C8H18O-0

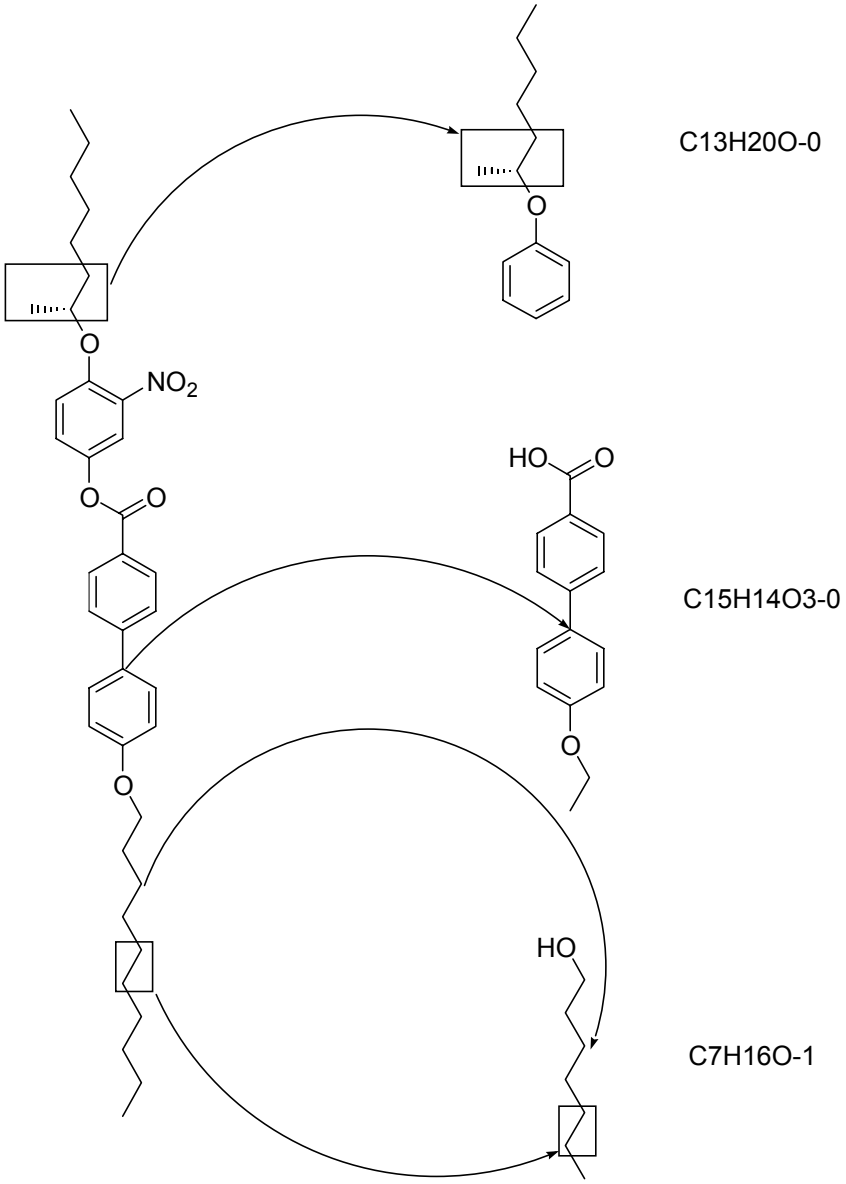
C16H15NO5-0

C12H18O-1

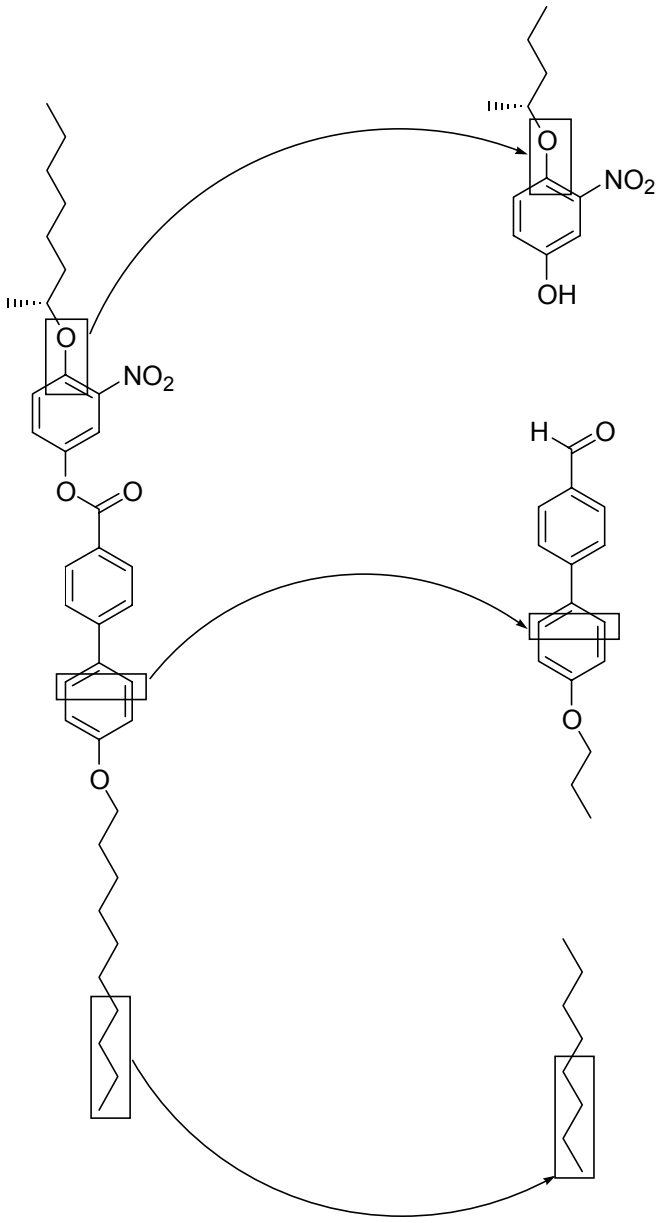
Parent to Fragment Atom Mappings for Parent compound 1, Part 2



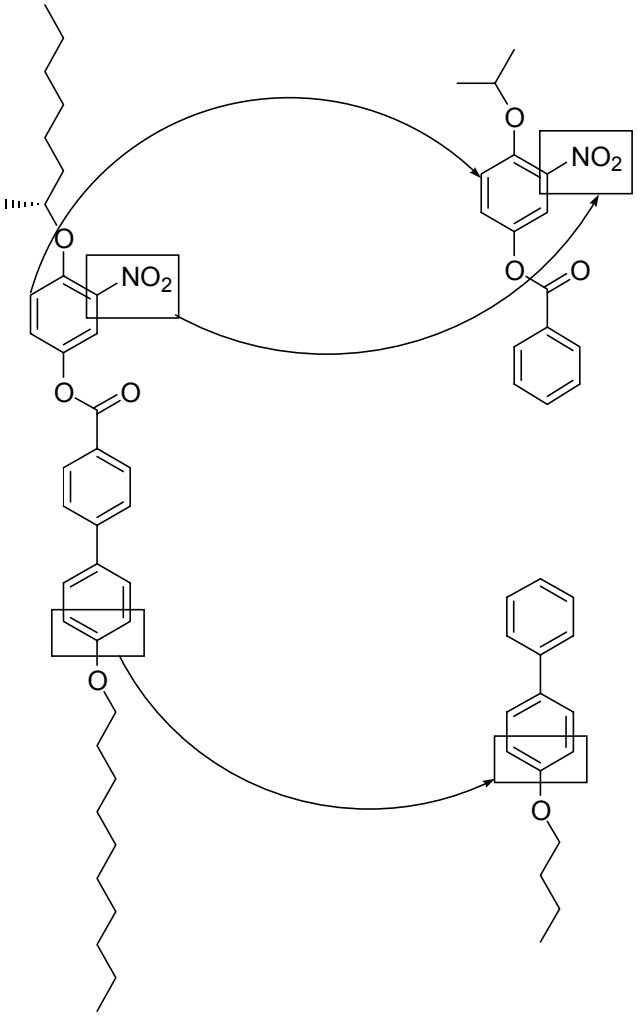
Parent to Fragment Atom Mappings for Parent compound 1, Part 3



Parent to Fragment Atom Mappings for Parent compound 1, Part 4



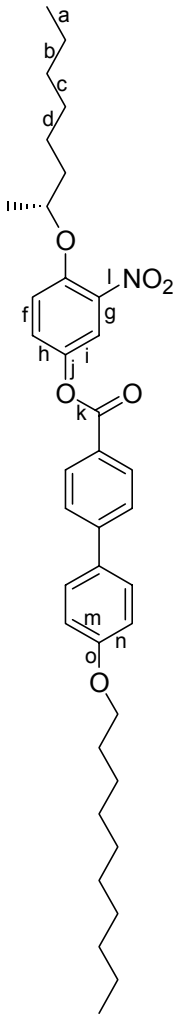
Parent to Fragment Atom Mappings for Parent compound 1, Part 5

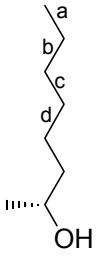


C11H15NO5-0

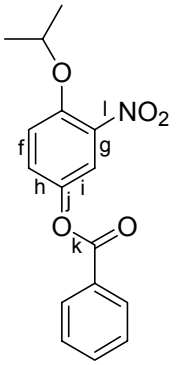
C16H18O-0

Parent to Fragment Bond Mappings for Parent compound 1, Part 1

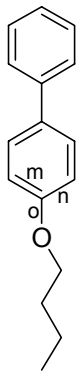




C8H18O-0

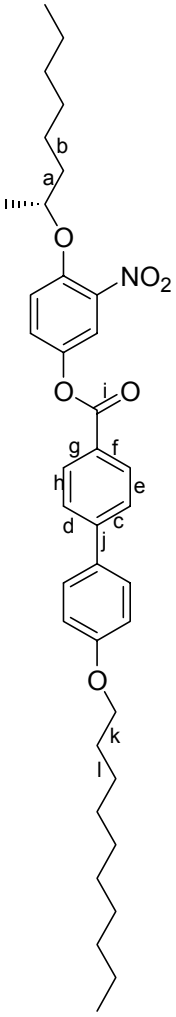


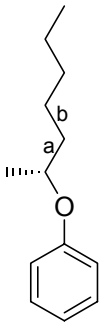
C16H15NO5-0



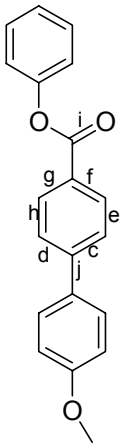
C16H18O-0

Parent to Fragment Bond Mappings for Parent compound 1, Part 2

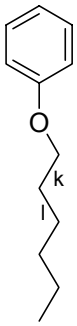




C13H20O-0

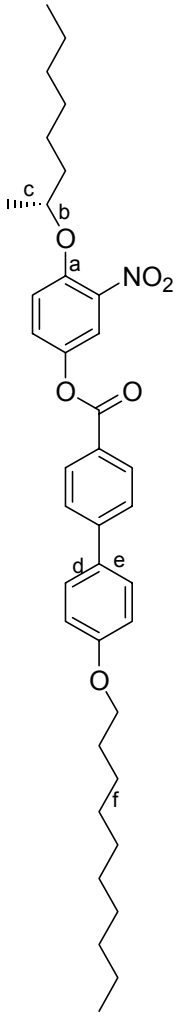


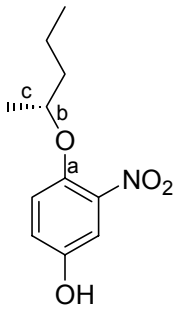
C20H16O3-0



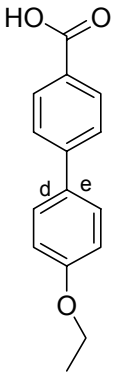
C12H18O-1

Parent to Fragment Bond Mappings for Parent compound 1, Part 3

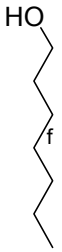




C11H15NO4-0

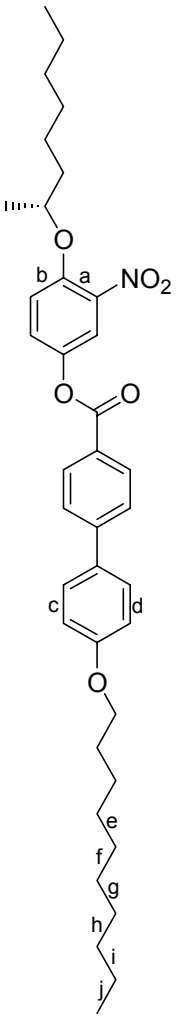


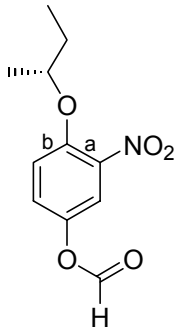
C15H14O3-0



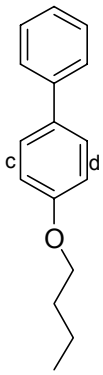
C7H16O-1

Parent to Fragment Bond Mappings for Parent compound 1, Part 4

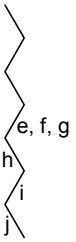




C11H13NO5-0



C16H16O2-0



C8H18-0

Atom Map List for Compound 1

Dir: C15H14O3-0 Parent atom 0: Qdb atom 0: qdb
Dir: C16H16O2-0 Parent atom 1: Qdb atom 1: qdb
Dir: C16H18O-0 Parent atom 2: Qdb atom 2: qdb
Dir: C16H18O-0 Parent atom 3: Qdb atom 3: qdb
Dir: C16H18O-0 Parent atom 4: Qdb atom 2: qdb
Dir: C16H16O2-0 Parent atom 5: Qdb atom 1: qdb
Dir: C16H16O2-0 Parent atom 6: Qdb atom 9: qdb
Dir: C16H18O-0 Parent atom 7: Qdb atom 7: qdb
Dir: C16H18O-0 Parent atom 8: Qdb atom 7: qdb
Dir: C16H16O2-0 Parent atom 9: Qdb atom 9: qdb
Dir: C20H16O3-0 Parent atom 10: Qdb atom 10: qdb
Dir: C20H16O3-0 Parent atom 11: Qdb atom 15: qdb
Dir: C20H16O3-0 Parent atom 12: Qdb atom 12: qdb
Dir: C20H16O3-0 Parent atom 13: Qdb atom 13: qdb
Dir: C20H16O3-0 Parent atom 14: Qdb atom 12: qdb
Dir: C20H16O3-0 Parent atom 15: Qdb atom 15: qdb
Dir: C20H16O3-0 Parent atom 16: Qdb atom 16: qdb
Dir: C20H16O3-0 Parent atom 17: Qdb atom 17: qdb
Dir: C20H16O3-0 Parent atom 18: Qdb atom 17: qdb
Dir: C20H16O3-0 Parent atom 19: Qdb atom 16: qdb
Dir: C16H15NO5-0 Parent atom 20: Qdb atom 11: qdb
Dir: C16H15NO5-0 Parent atom 21: Qdb atom 12: qdb
Dir: C16H15NO5-0 Parent atom 22: Qdb atom 13: qdb
Dir: C16H15NO5-0 Parent atom 23: Qdb atom 14: qdb
Dir: C16H15NO5-0 Parent atom 24: Qdb atom 15: qdb
Dir: C11H13NO5-0 Parent atom 25: Qdb atom 6: qdb
Dir: C11H15NO4-0 Parent atom 26: Qdb atom 5: qdb
Dir: C11H13NO5-0 Parent atom 27: Qdb atom 8: qdb
Dir: C16H15NO5-0 Parent atom 28: Qdb atom 19: qdb
Dir: C16H15NO5-0 Parent atom 29: Qdb atom 20: qdb
Dir: C11H13NO5-0 Parent atom 30: Qdb atom 11: qdb
Dir: C16H15NO5-0 Parent atom 31: Qdb atom 22: qdb
Dir: C11H13NO5-0 Parent atom 32: Qdb atom 13: qdb
Dir: C11H13NO5-0 Parent atom 33: Qdb atom 14: qdb
Dir: C11H13NO5-0 Parent atom 34: Qdb atom 14: qdb
Dir: C11H16O-0 Parent atom 35: Qdb atom 11: qdb
Dir: C11H15NO4-0 Parent atom 36: Qdb atom 14: qdb
Dir: C13H20O-0 Parent atom 37: Qdb atom 12: qdb
Dir: C13H20O-0 Parent atom 38: Qdb atom 13: qdb
Dir: C13H20O-0 Parent atom 39: Qdb atom 14: qdb
Dir: C7H16O-0 Parent atom 40: Qdb atom 5: qdb
Dir: C13H20O-0 Parent atom 41: Qdb atom 16: qdb
Dir: C13H20O-0 Parent atom 42: Qdb atom 16: qdb
Dir: C8H18O-0 Parent atom 43: Qdb atom 8: qdb
Dir: C7H16O-0 Parent atom 44: Qdb atom 10: qdb
Dir: C7H16O-0 Parent atom 45: Qdb atom 10: qdb
Dir: C8H18O-0 Parent atom 46: Qdb atom 11: qdb
Dir: C8H18O-0 Parent atom 47: Qdb atom 11: qdb
Dir: C8H18O-0 Parent atom 48: Qdb atom 13: qdb
Dir: C8H18O-0 Parent atom 49: Qdb atom 14: qdb
Dir: C8H18O-0 Parent atom 50: Qdb atom 14: qdb
Dir: C8H18O-0 Parent atom 51: Qdb atom 16: qdb
Dir: C8H18O-0 Parent atom 52: Qdb atom 17: qdb
Dir: C8H18O-0 Parent atom 53: Qdb atom 17: qdb

```
Dir: C12H18O-1 Parent atom 54: Qdb atom 12: qdb
Dir: C12H18O-1 Parent atom 55: Qdb atom 13: qdb
Dir: C12H18O-1 Parent atom 56: Qdb atom 13: qdb
Dir: C12H18O-1 Parent atom 57: Qdb atom 15: qdb
Dir: C12H18O-1 Parent atom 58: Qdb atom 16: qdb
Dir: C12H18O-1 Parent atom 59: Qdb atom 16: qdb
Dir: C7H16O-1 Parent atom 60: Qdb atom 8: qdb
Dir: C7H16O-1 Parent atom 61: Qdb atom 10: qdb
Dir: C7H16O-1 Parent atom 62: Qdb atom 10: qdb
Dir: C8H18O-0 Parent atom 63: Qdb atom 8: qdb
Dir: C8H18O-0 Parent atom 64: Qdb atom 11: qdb
Dir: C8H18O-0 Parent atom 65: Qdb atom 11: qdb
Dir: C7H16O-1 Parent atom 66: Qdb atom 11: qdb
Dir: C7H16O-1 Parent atom 67: Qdb atom 12: qdb
Dir: C7H16O-1 Parent atom 68: Qdb atom 12: qdb
Dir: C7H16O-1 Parent atom 69: Qdb atom 11: qdb
Dir: C7H16O-1 Parent atom 70: Qdb atom 12: qdb
Dir: C7H16O-1 Parent atom 71: Qdb atom 12: qdb
Dir: C8H18-0 Parent atom 72: Qdb atom 10: qdb
Dir: C8H18-0 Parent atom 73: Qdb atom 11: qdb
Dir: C8H18-0 Parent atom 74: Qdb atom 11: qdb
Dir: C8H18-0 Parent atom 75: Qdb atom 16: qdb
Dir: C8H18-0 Parent atom 76: Qdb atom 17: qdb
Dir: C8H18-0 Parent atom 77: Qdb atom 17: qdb
Dir: C8H18-0 Parent atom 78: Qdb atom 4: qdb
Dir: C8H18-0 Parent atom 79: Qdb atom 20: qdb
Dir: C8H18-0 Parent atom 80: Qdb atom 20: qdb
Dir: C13H20O-0 Parent atom 81: Qdb atom 29: qdb
Dir: C13H20O-0 Parent atom 82: Qdb atom 30: qdb
Dir: C13H20O-0 Parent atom 83: Qdb atom 30: qdb
Dir: C13H20O-0 Parent atom 84: Qdb atom 30: qdb
Dir: C8H18O-0 Parent atom 85: Qdb atom 23: qdb
Dir: C8H18O-0 Parent atom 86: Qdb atom 26: qdb
Dir: C8H18O-0 Parent atom 87: Qdb atom 26: qdb
Dir: C8H18O-0 Parent atom 88: Qdb atom 26: qdb
Dir: C8H18-0 Parent atom 89: Qdb atom 22: qdb
Dir: C8H18-0 Parent atom 90: Qdb atom 0: qdb
Dir: C8H18-0 Parent atom 91: Qdb atom 0: qdb
Dir: C8H18-0 Parent atom 92: Qdb atom 0: qdb
```

Bond Map List for Compound 1

```
Dir: C15H14O3-0 Parent bond 0-1: Qdb bond 0-1: qdb homo -
Dir: C15H14O3-0 Parent bond 0-5: Qdb bond 0-1: qdb homo
Dir: C20H16O3-0 Parent bond 0-10: Qdb bond 0-10: qdb homo -
Dir: C16H16O2-0 Parent bond 1-2: Qdb bond 4-5: qdb homo -
Dir: C16H16O2-0 Parent bond 1-6: Qdb bond 5-9: qdb homo +
Dir: C16H18O-0 Parent bond 2-3: Qdb bond 2-3: qdb homo -
Dir: C16H18O-0 Parent bond 2-7: Qdb bond 4-8: qdb homo +
Dir: C16H18O-0 Parent bond 3-4: Qdb bond 2-3: qdb homo
Dir: C16H18O-0 Parent bond 3-35: Qdb bond 3-21: qdb homo
Dir: C16H16O2-0 Parent bond 4-5: Qdb bond 4-5: qdb homo
Dir: C16H18O-0 Parent bond 4-8: Qdb bond 4-8: qdb homo +
Dir: C16H16O2-0 Parent bond 5-9: Qdb bond 5-9: qdb homo +
Dir: C20H16O3-0 Parent bond 10-11: Qdb bond 10-15: qdb homo
Dir: C20H16O3-0 Parent bond 10-15: Qdb bond 10-15: qdb homo
Dir: C20H16O3-0 Parent bond 11-12: Qdb bond 14-15: qdb homo
Dir: C20H16O3-0 Parent bond 11-16: Qdb bond 15-19: qdb homo +
Dir: C20H16O3-0 Parent bond 12-13: Qdb bond 12-13: qdb homo
Dir: C20H16O3-0 Parent bond 12-17: Qdb bond 12-17: qdb homo +
Dir: C20H16O3-0 Parent bond 13-14: Qdb bond 12-13: qdb homo
Dir: C20H16O3-0 Parent bond 13-20: Qdb bond 13-20: qdb homo
Dir: C20H16O3-0 Parent bond 14-15: Qdb bond 14-15: qdb homo
Dir: C20H16O3-0 Parent bond 14-18: Qdb bond 12-17: qdb homo +
Dir: C20H16O3-0 Parent bond 15-19: Qdb bond 15-19: qdb homo +
Dir: C20H16O3-0 Parent bond 20-21: Qdb bond 20-21: qdb homo
Dir: C16H15NO5-0 Parent bond 20-22: Qdb bond 11-13: qdb homo -
Dir: C16H15NO5-0 Parent bond 22-23: Qdb bond 13-14: qdb homo
Dir: C16H15NO5-0 Parent bond 23-24: Qdb bond 14-15: qdb homo
Dir: C16H15NO5-0 Parent bond 23-28: Qdb bond 14-19: qdb homo
Dir: C16H15NO5-0 Parent bond 24-25: Qdb bond 15-16: qdb homo
Dir: C16H15NO5-0 Parent bond 24-29: Qdb bond 15-20: qdb homo +
Dir: C11H13NO5-0 Parent bond 25-26: Qdb bond 6-7: qdb homo -
Dir: C16H15NO5-0 Parent bond 25-32: Qdb bond 16-23: qdb homo
Dir: C11H13NO5-0 Parent bond 26-27: Qdb bond 7-8: qdb homo
Dir: C11H15NO4-0 Parent bond 26-36: Qdb bond 5-14: qdb homo -
Dir: C16H15NO5-0 Parent bond 27-28: Qdb bond 18-19: qdb homo
Dir: C16H15NO5-0 Parent bond 27-30: Qdb bond 18-21: qdb homo +
Dir: C16H15NO5-0 Parent bond 28-31: Qdb bond 19-22: qdb homo +
Dir: C16H15NO5-0 Parent bond 32-33: Qdb bond 23-24: qdb homo
Dir: C16H15NO5-0 Parent bond 32-34: Qdb bond 23-24: qdb homo
Dir: C12H18O-1 Parent bond 35-54: Qdb bond 11-12: qdb homo -
Dir: C11H15NO4-0 Parent bond 36-37: Qdb bond 14-15: qdb homo
Dir: C13H20O-0 Parent bond 37-38: Qdb bond 12-13: qdb homo -
Dir: C11H15NO4-0 Parent bond 37-39: Qdb bond 15-17: qdb homo +
Dir: C11H15NO4-0 Parent bond 37-81: Qdb bond 15-27: qdb homo
Dir: C13H20O-0 Parent bond 38-40: Qdb bond 13-15: qdb homo
Dir: C13H20O-0 Parent bond 38-41: Qdb bond 13-16: qdb homo +
Dir: C13H20O-0 Parent bond 38-42: Qdb bond 13-16: qdb homo +
Dir: C8H18O-0 Parent bond 40-43: Qdb bond 5-8: qdb homo -
Dir: C13H20O-0 Parent bond 40-44: Qdb bond 15-19: qdb homo +
Dir: C13H20O-0 Parent bond 40-45: Qdb bond 15-19: qdb homo +
Dir: C8H18O-0 Parent bond 43-46: Qdb bond 8-11: qdb homo +
Dir: C8H18O-0 Parent bond 43-47: Qdb bond 8-11: qdb homo +
Dir: C8H18O-0 Parent bond 43-48: Qdb bond 8-13: qdb homo
Dir: C8H18O-0 Parent bond 48-49: Qdb bond 13-14: qdb homo +
```

```
Dir: C8H18O-0 Parent bond 48-50: Qdb bond 13-14: qdb homo +
Dir: C8H18O-0 Parent bond 48-51: Qdb bond 13-16: qdb homo
Dir: C8H18O-0 Parent bond 51-52: Qdb bond 16-17: qdb homo +
Dir: C8H18O-0 Parent bond 51-53: Qdb bond 16-17: qdb homo +
Dir: C8H18O-0 Parent bond 51-85: Qdb bond 16-23: qdb homo
Dir: C12H18O-1 Parent bond 54-55: Qdb bond 12-13: qdb homo +
Dir: C12H18O-1 Parent bond 54-56: Qdb bond 12-13: qdb homo +
Dir: C12H18O-1 Parent bond 54-57: Qdb bond 12-15: qdb homo
Dir: C12H18O-1 Parent bond 57-58: Qdb bond 15-17: qdb homo +
Dir: C12H18O-1 Parent bond 57-59: Qdb bond 15-17: qdb homo +
Dir: C12H18O-1 Parent bond 57-60: Qdb bond 15-18: qdb homo
Dir: C12H18O-0 Parent bond 60-61: Qdb bond 15-19: qdb homo +
Dir: C12H18O-0 Parent bond 60-62: Qdb bond 15-19: qdb homo +
Dir: C7H16O-1 Parent bond 60-63: Qdb bond 8-11: qdb homo -
Dir: C8H18O-0 Parent bond 63-64: Qdb bond 8-11: qdb homo +
Dir: C8H18O-0 Parent bond 63-65: Qdb bond 8-11: qdb homo +
Dir: C8H18-0 Parent bond 63-66: Qdb bond 10-13: qdb homo -
Dir: C8H18-0 Parent bond 66-67: Qdb bond 13-14: qdb homo +
Dir: C8H18-0 Parent bond 66-68: Qdb bond 13-14: qdb homo +
Dir: C8H18-0 Parent bond 66-69: Qdb bond 10-13: qdb homo
Dir: C8H18-0 Parent bond 69-70: Qdb bond 13-14: qdb homo +
Dir: C8H18-0 Parent bond 69-71: Qdb bond 13-14: qdb homo +
Dir: C8H18-0 Parent bond 69-72: Qdb bond 10-13: qdb homo
Dir: C8H18-0 Parent bond 72-73: Qdb bond 13-14: qdb homo +
Dir: C8H18-0 Parent bond 72-74: Qdb bond 13-14: qdb homo +
Dir: C8H18-0 Parent bond 72-75: Qdb bond 7-10: qdb homo
Dir: C8H18-0 Parent bond 75-76: Qdb bond 16-17: qdb homo +
Dir: C8H18-0 Parent bond 75-77: Qdb bond 16-17: qdb homo +
Dir: C8H18-0 Parent bond 75-78: Qdb bond 4-7: qdb homo
Dir: C8H18-0 Parent bond 78-79: Qdb bond 4-5: qdb homo +
Dir: C8H18-0 Parent bond 78-80: Qdb bond 4-5: qdb homo +
Dir: C8H18-0 Parent bond 78-89: Qdb bond 19-22: qdb homo
Dir: C11H15NO4-0 Parent bond 81-82: Qdb bond 27-28: qdb homo +
Dir: C11H15NO4-0 Parent bond 81-83: Qdb bond 27-28: qdb homo +
Dir: C11H15NO4-0 Parent bond 81-84: Qdb bond 27-28: qdb homo +
Dir: C8H18O-0 Parent bond 85-86: Qdb bond 23-24: qdb homo +
Dir: C8H18O-0 Parent bond 85-87: Qdb bond 23-24: qdb homo +
Dir: C8H18O-0 Parent bond 85-88: Qdb bond 23-24: qdb homo +
Dir: C8H18-0 Parent bond 89-90: Qdb bond 22-23: qdb homo +
Dir: C8H18-0 Parent bond 89-91: Qdb bond 22-23: qdb homo +
Dir: C8H18-0 Parent bond 89-92: Qdb bond 22-23: qdb homo +
```

Appendix C

This appendix contains the source for the more ‘important’ portions of the software system. Not all programs are included, but the major ones are. They are organized by directory. The sections are titled by the directory name, and subtitled by the general purpose of the program, unless there is only one type of program in that directory, as outlined in Chapter 3. Oftentime, programs in one directory rely on libraries or routines in other directories. This would be made clear by included the config.pl from each directory, but for the sake of brevity, these programs have been omitted; they can be found either on the enclosed CD, or at the permanent home of the project.

In this edition of the thesis, the contents of Appendix C are omitted. They can be found on the companion CD, as the full source in the directory ‘/ff’, or (a more current version) can be downloaded from ffdev.sourceforge.net.