I. Quantal Effects in Biochemical Cooperativity and a Proposed Mechanism for the Differentiation of Calcium Signaling in Synaptic Plasticity
II. Evolutionary Algorithms for the Optimization of Methods in Computational Chemistry

Thesis by

William Chastang Ford

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

2012

(Defended September 26, 2011)

to my Mom, Nicole, Natalie, Mike,

Bill G., Bill H., Joe, Mamadou, and Joel.

Abstract

In Part 1 of this thesis, we propose that biochemical cooperativity is a fundamentally non-ideal process. We show quantal effects underlying biochemical cooperativity and highlight apparent ergodic breaking at small volumes. The apparent ergodic breaking manifests itself in a divergence of deterministic and stochastic models. We further predict that this divergence of deterministic and stochastic results is a failure of the deterministic methods rather than an issue of stochastic simulations.

Ergodic breaking at small volumes may allow these molecular complexes to function as switches to a greater degree than has previously been shown. We propose that this ergodic breaking is a phenomenon that the synapse might exploit to differentiate Ca^{2+} signaling that would lead to either the strengthening or weakening of a synapse. Techniques such as lattice-based statistics and rule-based modeling are tools that allow us to directly confront this non-ideality. A natural next step to understanding the chemical physics that underlies these processes is to consider *in silico* specifically atomistic simulation methods that might augment our modeling efforts.

In the second part of this thesis, we use evolutionary algorithms to optimize

in silico methods that might be used to describe biochemical processes at the subcellular and molecular levels. While we have applied evolutionary algorithms to several methods, this thesis will focus on the optimization of charge equilibration methods. Accurate charges are essential to understanding the electrostatic interactions that are involved in ligand binding, as frequently discussed in the first part of this thesis.

Contents

List of Figures

List of Tables

Algorithms and Source Code

Part I

Quantal Effects in Biochemical Cooperativity and a Proposed Mechanism for the Differentiation of Calcium Signaling in Synaptic Plasticity

Chapter 1

Motivation for the Use of Quantal Effectors in Chemical Reaction Networks

1.1 My Historical Interest in Stochastic Modeling

My motivation for the work contained in Part I of my thesis started before I matriculated at Caltech. I spent two years studying stochastic chemical reaction networks under the late Dr. Joel Stiles at the Pittsburgh Supercomputing Center. Joel was a co-author of a software package used to perform particle-based reaction-diffusion simulations of biochemical systems in arbitrarily complex three-dimensional space. Joel called this "Microphysiology."

Much of my PhD has been spent developing methods in computational chemistry. As an undergraduate, I began working with a simulation package called MCell, a stochastic reaction-diffusion simulation package. I transitioned into developing methods of stochastic simulation and optimization that will be discussed later in Part II of this thesis.

1.2 Simulating More Than Quintillion States of Macromolecular States on a Personal Computer

An early project that I worked on involved modeling a ”full-state” model of calcium / calmodulin / calcium calmodulin kinase II. This model, as conceived, contained more than 10^{19} states and was the most complex (high number of states vs. the number of molecular constituents) proposed model of a biochemical system that I was aware of. I devised of a method to model it that is similar to past methods applied to hemoglobin. I initially implemented this method as a wrapper around MCell, a reaction diffusion simulating code. I treated the system as if hemoglobin were in a lattice, and I manipulated reaction rates as a function of the states of the other subunits. The rate law for the nodes of the lattice is a function of the occupation of the subunits of CaMKII's neighbours. This is analogous to the ”lattice-based” methods of Terrance L. Hill and other theoretical chemists.

In the process of building these models of Ca/CaM/CaMKII, I became aware that other modelers were doing something similar to what theoretical chemists had done in the past, but without explicitly building upon their work. “Rule-based modeling” has reformulated a solution to patch-up the problem of combinatorial explosions without addressing its cause, our formulation of solutions. I will address the fact that we are growing out of the simplistic description of chemical reaction networks that mass-action requires. This is the impetus to develop a language to explicitly describe chemical reactions that are governed by non-ideal interac-

tions as quantal chemical reactions. We need to develop a more general language and understanding of quantal chemical interactions that is not domain-specific, or taught only to graduate level chemists, physicists, or computer scientists.

1.3 Overview of Part I

In Chapter 2 of this thesis, I will review our recent standard treatments of chemical reaction networks. In Chapter 3, I will then draw similarities between the use of lattice statistics and rule-based modeling for the simulation of biochemical systems.

In Chapter 4 I will discuss a biological system where our later findings might provide useful insight.

In Chapter 5, I will review the computational methods that I use that should be accessible to undergraduate students. I will also propose that we should not look toward differential methods as a reference point by which to base the validity of the use of stochastic models to represent systems that are inherently stochastic.

In the following chapters, I will present several case studies that give insight into a general treatment of non-ideal systems. When possible, I will refer to features that are analogous to other formulations. In the first case, the pH-dependent protonation of the zwitterion of glycine is modeled, and the various solutions are compared.

In the second case, the complexation of four dioxygen molecules by hemoglobin is modeled in an effort to account for the apparent cooperative relationship, or the

enhanced binding affinity with respect to the successive uptake of oxygen, that is observed experimentally between the four different subunits of the hemoglobin molecule. In the third case, I will lay out the foundation for building "toy models" of calcium (Ca^{2+}), calmodulin (CaM), and calcium-calmodulin kinase II (CaMKII). I will subsequently refer to these as Ca^{2+} / CaM / CaMKII models as they were the impetus for the other case studies. I first witnessed the quantal effects of cooperativity when modeling a system of Ca^{2+} / CaM / CaMKII. I initially described it in a candidacy report that I wrote in the beginning of 2009.

Chapter 2

Introduction to Rate Theory

The idea of a rate constant is a misconception. Enzymes can catalyze reactions after association with their substrate. The association of a non-competitive inhibitor, often distal to an active-site of a protein, can decrease the reaction rate of that protein. Even the state of local subunits of a macromolecule may affect the reactivity of other subunits. Though it is glossed over, *this phenomena is non-ideal and bound by those consequences*. In this portion of my thesis, we will develop the idea of quantal effectors as they apply to cooperative chemical reactions. The major proposition of this work is that although we can model large systems with the equivalent of many quadrillion states, we are still bound by the fact that we are modeling quantal effects.

In collision theory of gas-phase kinetics, there is the assumption that in a dilute gaseous environment, the propensity for the collision or association of an A-B pair to result in a reaction is not affected by the existence of other non-reacting elements. At the same time, it states that the rate of this reaction is governed by $Ae^{\frac{E_A}{RT}}$. In biochemistry, E_A is known to be affected at times by non-reacting molecules (for example, neighboring subunits in a macromolecular complex). Transition state

theory deals with this complication by including the addition of a non-reacting effector in another reaction, thereby allowing a reaction to proceed with a different E_A.

We circumvent the assumption that reactants cannot be affected by non-reactants by including the various combinations that might affect a reaction rate explicitly. Alternatively, instead of explicitly stating each and every combination that might affect a reaction rate, we implement reaction-rate functions, such as the Arrhenius equation or a "rule-based modeling" type of relationship.

2.1 Chemical Reaction Theory

2.1.1 Mass-Action

For a detailed account of the history of the law of mass action see Lund [1965].

$$\text{Rate of Reaction} \propto k[\text{A}][\text{B}]$$

where *k* is the reaction rate constant, [A] is the concentration of one reactant, and [B] is the concentration of another reactant in a bimolecular reaction.

The following relationship is relevant and can be found in MacQueen [1967].

$$\Delta G^{\theta} = -RT \log_e K \tag{2.1}$$

$$\Delta G^{\theta} = \Delta H^{\theta} - T\Delta S^{\theta} \tag{2.2}$$

$$K = \exp\left(\frac{-\Delta G^{\theta}}{RT}\right) = \exp\frac{-\Delta H^{\theta}}{RT} \exp\frac{\Delta S^{\theta}}{R} \tag{2.3}$$

The particulars of the function are not as important as gaining an appreciation for the form of the function (for further details see Wright [2004]).

Chemical reactions have been described thermodynamically using Maxwell-Boltzmann statistics for more that a century. Maxwell-Boltzmann statistics are derived from the theory that reactions can be broken up into specific classes and described by the stoichiometry or the order of the reaction Gillespie [1976].

- For example:
  - Zero Reactants / Zeroth-Order Reaction / "Birth Processes"

$$* \rightarrow \ reaction\ products$$

    rate = k.

  - One Reactant / First-Order Reaction/ "Unimolecular Processes"

$$S_j \rightarrow \ reaction\ products$$

    rate = $k[S_j]$.

– Two Reactants / Second-Order Reaction/ "Bimolecular Processes"

$$S_j + S_k \rightarrow \ reaction\ products (j \neq k)$$

rate = $k[S_j][S_k]$

$$2S_j \rightarrow \ reaction\ products$$

rate = $k[S_j]^2$.

– Three Reactants / Third-Order Reaction/ "Termolecular Processes"

$$S_i + S_j + S_k \rightarrow \ reaction\ products (i \neq j \neq k)$$

rate = $k[S_i][S_j][S_k]$

$$S_j + 2S_k \rightarrow \ reaction\ products (j \neq k)$$

rate = $k[S_j][S_k]^2$

$$3S_j \rightarrow \ reaction\ products$$

rate = $k[S_j]^3$.

While higher-order reactions are theoretically possible, we typically consider reactions to be decomposable into zero-, first-, or second-order processes. The theory behind this is that it is improbable for more than two independently diffusing particles to interact or collide with one another at the exact same time. For indepen-

dently diffusing particles, this is most likely true. In the case of macromolecules, reactive units tend to diffuse together.

Various updates that have been made to the initial theories have brought us closer to modeling the physical realities of chemical systems.

2.1.2 Rate Theories Derived From Mass-Action

Consider the following equation to represent the general form for a reaction rate function.

$$k = (pEF)e^{\frac{-E}{RT}} \tag{2.4}$$

$$k = (pEF)e^{\frac{-(E_1+E_2)}{RT}}$$

$$k = (pEF)e^{\frac{-\Delta H}{RT}}e^{\frac{\Delta S}{R}}$$

Equation 2.4 has components pEF, E, R, T, ΔH, and ΔS, which equal the pre-exponential factor, energy, a gas constant, temperature, change in enthalpy, and change in entropy, respectively. The form of this function is seen in the Arrhenius equation, collision theory, and transition state theory.

$$\ln k = -(\frac{E_a}{RT}) + \ln A$$

$$k = Ae^{(-\frac{E_a}{RT})} \qquad \text{(Arrhenius equations)}$$

$$k = pZe^{(-\frac{E_a}{RT})}$$

$$k = P_r Ze^{(-\frac{E_a}{RT})} \quad \text{(Collision theory)}$$

$$k = \kappa \frac{kT}{h} \frac{Q^{\neq *}}{QxQyz} e^{(-\frac{E_0}{RT})} \quad \text{(Transition state theory)}$$

2.2 "Reaction Rate Constants" and Non-Ideal Reactions

Above, I use quotes when discussing rate constants because the term is a misnomer and interferes with our ability to talk about non-ideal reactions in a more general context. Reaction rates are not always constant and should be thought of as functions that are affected by both macroscopic factors and microscopic factors, akin to extrinsic and intrinsic properties of systems.

The most famous rate function is the Arrhenius equation:

$$k = Ae^{\frac{-E_a}{RT}} \tag{2.5}$$

where k is the reaction rate constant, A is a pre-exponential term, E_A is the energy of activation, R is the gas constant, and T is the temperature. The Arrhenius equation treats the rate of the reaction as a function of temperature and can be thought of as a reaction rate equation or as a "rule" that describes the relationship of rate and temperature. "Rule" is a term coined in rule-based modeling while "reaction rate equation" is a more descriptive term. Typically, chemists do not implement

this function consistently.

- Often when chemists consider a change in temperature, they explicitly invoke this equation or similar equations. *This is because it is the same reaction under different conditions.* The explicit treatment results in understanding that the rate of reaction is a function of temperature.

- Often when chemists consider a change of energy of activation, they ignore the dynamic nature of reaction rates. An example of this is the treatment of auto-catalytic systems such as hemoglobin/O_2. We explicitly elaborate out all non-ideal effects that lead to changes in reaction rates. In doing so, we treat the binding of O_2 to hemoglobin's four sites as as many as sixteen different reactions.

A non-ideal reaction exhibits behavior that allows other molecular elements that are neither reactants or products to affect its rate of progression. These non-ideal interactions occur at the individual level and are not directly compatible with common calculus-based models. This class of interactions that are discrete in nature can be described by mass-action if every non-ideal interaction is elaborated out. This treats a single complicated reactions as a set of reactions. For highly non-ideal systems, this results in a large number of simple reactions. This is termed combinatorial complexity.

Rule-based modelers often state that their method resolves the combinatorial complexity of chemical reactions. This definition speaks to treating the symptoms of a problem without realizing the fundamental cause. We will discuss this further

in the section on “rule-based modeling.”

2.3 Reaction Effectors

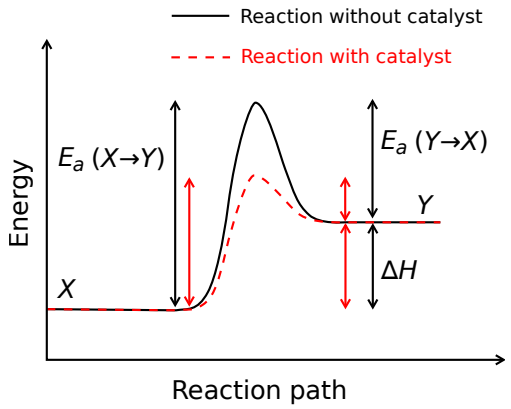


Figure 2.1: The relationship between activation energy (E_a) and enthalpy of formation (ΔH) with and without a catalyst, plotted against the reaction coordinate. $E_a(X- > Y)$ and $E_a(X- > Y)$ are the activation energies for the forward and reverse reactions, respectively. The highest energy position (peak position) represents the transition state. With the catalyst, the energy required to enter transition state decreases, thereby decreasing the energy required to initiate the reaction. Wikipedia [2009]

In biochemistry, there are promoters and inhibitors of chemical reactions that affect the rate of reactions, but do not necessarily interact directly with the reacting elements. We will refer to this class of molecules and molecular domains as “effectors”. They may affect the reaction coordinate of reacting biological molecules by altering the activation energy, as depicted in Figure 2.1. The Arrhenius equation (2.5) shows us that a change in E_A leads to a change in the rate constant of a reaction k.

Enzymes, a subgroup of biological effectors, are the biological equivalent of catalysts. Figure 2.1 illustrates an example of how one might describe the action

of a catalyst. Enzymes and catalysts are modeled as if they lower the activation energy of a reaction.

2.4 The Consequences of a Model with Many States: Combinatorial Complexity

For more than a century, rate equations have been used to describe systems by posing questions in the following form: *"How much of a molecule in some state X exists at time T?"* This is the result of the population-based framework that is required when invoking the law of mass-action. Biochemists and biophysicists have sought to use these models to help understand a variety of processes, from gene transcription, to oxygen delivery to tissues, to cell signal transduction.

The molecules that are modeled in these processes exist in various states, and each state is typically represented by an individual data element. For example, a molecule bound to a substrate exists in a different state than a molecule of the same type in its unbound form or bound to a different substrate. Furthermore, macromolecules in which multiple subunits act as effectors of other subunits are often described as a combination of the individual states of the subunits, such that the number of states of the macromolecule is equal to:

$$(\text{possible states of } subunit_1) \times (\text{possible states of } subunit_2) \ldots \times (\text{possible states of } subunit_n).$$

Thus, a homo-oligomer can to be described by a^b data elements, where *a* equals

the number of states in which a single subunit can exist, and b equals the number of subunits in a macromolecule. One example of this is cooperative binding, where inter-subunit interactions allow for increasing binding affinity upon subsequent binding events.

Modelers have grown accustomed to building models in which the number of data elements (i.e., the model's data-space) equals the number of states of all molecules modeled in the system (i.e., the model's state-space). Following this dogma, as the number of subunits in an oligomer increases, the number of data elements needed to describe its states increases exponentially. This is sometimes referred to as a combinatorial explosion (a^b). In this work, we present a framework in which we are able to represent the full state-space of approximately a^b states with a significantly smaller data-space, $a \times b$. In the following chapters we will show that the combinatorial complexity is a function of the theory of mass action rather than a truth of real chemical reaction networks.

As an aside: It is more impressive to model chemical reaction networks without defining reactants, products, effectors, and reactions. By employing reactive molecular dynamics force-fields such as Reaxff (van Duin et al. [2001]), one can model complex biochemical systems without predefining many of the features required in standard or modified models of chemical reaction networks. Code for optimization of Reaxff will be provided at the end of the second part of this thesis.

Chapter 3

Quantal Effectors on Lattices

In this chapter, we introduce the terminology used to describe molecular elements that exhibit explicitly non-ideal behavior. We define these terms here so that we might describe the quantal effects that we see in future chapters. We will introduce quantal effectors that behave in an explicitly discrete manner. We will also discuss their early description as cooperative effectors and how they were treated in the mass-action framework. Next we will touch on statistical mechanics, specifically lattice theory as it has been applied to hemoglobin to model cooperative effects.

Following this discussion of discretization of hemoglobin subunits to points on a lattice, we will discuss how these ideas have been re-framed and extended by rule-based modelers. Finally, we will attempt to unify the insights made toward the understanding of hemoglobin with a general theme that we have revisited: that we do not have an appropriate language to describe molecular elements that affect reactions in a discrete manner, but are neither reactants nor products.

3.1 Introducing Quantal Effectors

Cofactors, promoters, and non-competitive inhibitors can and should be considered "effectors" of reactions. We have excluded competitive inhibitors from this list because they directly interact with the binding site of at least one of the reacting molecules and, therefore, could play a very different role than effectors. A very important subclass of effectors are the subunits/domains of molecules and macromolecules that affect the reactivity of other subunits/domains. We will refer to these as "quantal effectors".

Quantal effectors are molecular entities that are explicitly non-ideal. Naming these elements quantal effectors gives the user insight into the quantal behavior that may become apparent under certain conditions. An alternative name that is equally or more apt is non-ideal effector. Such a term is important because it is not domain-specific, and it provides insight into general truths. Domain-specific terms—such as lattices, agents, and rules—have specific meanings that are difficult to distinguish from the methods with which they are associated.

Since the early 20th century, we have been aware of complex reaction dynamics shown by molecular complexes. In 1925, Adair showed that hemoglobin has multiple binding sites for oxygen and that these sites potentiate one another when bound. While the reacting dynamics of hemoglobin can be coaxed into a mass-action type of model, it is non-ideal in that hemoglobin subunits autocatalyze one another in a non-ideal manner that is only partially governed by mass-action, as quantal effectors.

3.2 Distinguishing Reactions with Quantal Effectors from Higher-Order Reactions

Adding a quantal effector to a reaction may appear to be equivalent to increasing the molecularity of the reaction, but it is explicitly not equivalent. The quantal effector is more likely to affect only a small subset of the ensemble.

Consider a termolecular reaction that is ideal in that it obeys mass-action dynamics. Termolecularity implies that the reaction occurs due to the diffusion of three particles colliding with one another at the same instant of time. There are two points to consider here:

- The probability of collision is much lower for three particles diffusing in a solution than it is for a macromolecule with multiple sites and a particle diffusing in a solution.

- In a reaction, any elements that are representative members of the class of reactants may interact with one another. Often in macromolecules subunits maintain their associations, resulting in less promiscuous reaction schemes. Ising models of chemical systems tend to maintain their node/connection structure.

Therefore, quantal effectors should be considered more likely and and less promiscuous than true higher-order reactions.

3.3 Cooperativity

Cooperative binding requires that a molecule, such as hemoglobin or glycine, has more than one binding site. In *Binding and Linkage* [1990], Wyman refers to the phenomena of charge repulsion between the protonation sites of glycine as "linkage" rather than "negative cooperativity" based upon the belief that an allosteric shift is required for a cooperative event to occur. It should be noted that the allosteric model of hemoglobin/O_2 that was proposed by Monod, Wyman, and Changeaux in [1965] earned them a Nobel prize. In this thesis, we are not attempting to challenge the notion of allostery being essential to hemoglobin cooperativity. We do not wish to use multiple words to describe phenomena whose mathematical treatment would be equivalent. Therefore, we will lump "cooperativity," "allostery," "causal rules," "linkage," and "coupling" into one general phenomena governed by non-ideal interactions. We call this phenomena "cooperativity" out of respect for its early description (Adair [1925b]).

We generalize cooperative processes to be those reactions that are explicitly non-ideal, in that an element that is not a product or reactant affects the progress of a reaction. Again, we refer to these elements as quantal effectors. For example, the charge repulsion that a protonation site of glycine experiences due to the other site being occupied is a negative cooperative effect. In this case, the protonated terminus that is not taking part directly in the subsequent protonation inhibits the reaction. Conversely, if the binding of the ligand oxygen at one binding site of hemoglobin increases the affinity for another oxygen at a different site on the

same macromolecule, this is considered to be positive cooperativity. The quantal effects in this case are the other subunits, meaning that for each binding site of hemoglobin, there are three quantal effector sites that might affect reactions at that site in an explicitly non-ideal manner.

At this point, we can relate the idea of cooperativity back to some concepts of rate theory that were presented earlier. In the report *Energetics of Subunit Assembly and Ligand Binding in Human Hemoglobin* [1980], Gary Ackers suggests that "the dominate driving forces for cooperativity may be a combination of hydrogen bond formation, preferentially stabilizing the tetramer with large negative ΔS and small ΔH, and Bohr proton release, yielding large positive enthalpies and moderate but negative entropies".

3.4 Ising Models of Chemical Systems

In 1925, Ising proposed a model for the study of ferromagnetic systems. The Ising model has since been applied to many fields, including biochemistry and computational neuroscience (Hopfield [1982]). Lattice models of cooperative systems typically break up the molecular subunits of a macromolecule into nodes on a lattice. The points on a lattice have independent probabilities of transitioning until they are connected. Once connected, the states are either positively or negatively coupled, and the states of connected nodes are affected by the "coupling coefficient". An illustration depicting the relationship of Ising models to a lattice model of a molecule, glycine, is presented as Figure 3.1.

While there are many papers that treat cooperative systems using lattice methods, we will focus on a single paper that models hemoglobin as a set of points on a lattice using the relationships proposed by Pauling [1935]. Chay and Ho [1973] use lattice statistics to model hemoglobin as a two-dimensional Ising model with N = 4. They modeled different cooperative relationships between the subunits of hemoglobin in an attempt to elucidate the underlying mechanisms of its cooperativity. Their work is similar to what Pauling did in 1935 using mass action equations. The models of Chay are explicit examples of treating cooperative relationships as non-ideal processes. The non-ideal relationships represent the cooperative and causal relationships between molecular subunits of a macromolecule.

3.5 Rule-Based Modeling for Biochemists

Rule-based modeling is a field of simulation that, by citation, seems to derive from Gillespie's stochastic simulation algorithm [1976, 1977]. I contend that they also derive from the understanding of chemical physics that underlies lattice statistics of cooperative macromolecules. Both lattice statistics and rule-based modeling break up macromolecules into their constituent parts and explicitly consider the relationships between them. It is interesting that few, if any, rule-based modeling packages refer to the lattice-based statistical methods that they mirror. Lattice statistics is a field that is still active, and it is faced with issues of intractability that could be resolved with solutions from rule-based modeling.

Many rule-based modeling packages do not attempt to give their rules a physical basis. Moleculizer (Lok and Brent [2005]) is a package that does explicitly attempt to employ physical reasoning for its rules. For example, Lok and Brent [2005] presented a stochastic model for the effects of oligomerization on the chemical reaction rates of the corresponding monomer using a classical collision theory formulation as follows:

$$c_\mu = V^{-1}\pi d_{12}^2(8kT/\pi m_{12})^{1/2}\exp(-u_\mu/kT) \tag{3.1}$$

where c_μ is the stochastic reaction rate parameter, m_{12} is the adjusted mass, k is the Boltzmann constant, d_{12} is idealized collision cross-section diameter, V is the volume of the system, and $u\mu$ is analogous to the energy of activation. The approach taken by Lok and Brent is based on classical molecular collision theory.

One oddity of rule-based modeling is that the different camps of rule-based modelers do not have a unifying language to talk about the formulation of their models. I began rule-based modeling in 2006 without knowledge of the communities of rule-based modelers that began developing around 2004. My first models involved pausing stochastic simulations, altering their rates as a function of the states of the subunits of a "toy hemoglobin", and then restarting the simulation. When I discovered the rule-based modeling communities, I was frustrated that I could not clearly determine if we were doing analogous work.

This is because the language that the communities use is either partially self-generated (Lok and Brent [2005]), rooted in abstract computer science (Danos and

Laneve [2004]), or both (Blinov et al. [2004]). For example, Moleculizer (Lok and Brent [2005]) refers to molecular subunits as *mols*, while Kappa(κ) (Danos and Laneve [2004]) and BioNetGen (Blinov et al. [2004]) refer to subunits as *agents*. This is part of the impetus to create a unified language to describe these non-ideal relationships that lead to cooperative behavior.

Furthermore, various pieces of software have been developed to perform rule-based modeling without presenting generalized outlines and derivations of their methods, in the vein of what Gillespie has presented [1976, 1977]. These programs appear to treat molecules as individuals, rather than populations. This is an important point because they are essentially reversing the step in many mass-action based derivations of chemical reaction theory that break up complicated reactions into several simple reactions to fit into a population-based framework. This is a relevant point that is seemingly absent in rule-based modeling literature.

3.6 Unification of These Ideas with *Quantal Effectors*

To unify these ideas, it is important to look at the fundamental truths that these methods are trying to address. The language that we use to describe chemical reactions is limited by the view that reactions contain only reactants and products. In the 1970s, theoretical chemists, and more recently rule-based modelers, redefined rate laws to include the state of particles that are neither reactants or products. A fundamental feature of these particles is that they affect the reaction rate in a fundamentally non-ideal manner that is quantal in nature. Hence, we refer to them as

"quantal effectors".

Furthermore, at very small numbers, the simple non-ideal relationships between these complex elements can lead to behavior that is similar to ergodic-breaking. In quantum mechanics, quantal effects are a cause of ergodic-breaking, given that the *ensembles* considered are comparable. This again supports the name quantal effectors, as it is informative and corrects the misconception that deterministic and stochastic treatments should always converge. In this thesis, we will define the quantal effector limit beyond which one could safely expect macroscopic-like behavior.

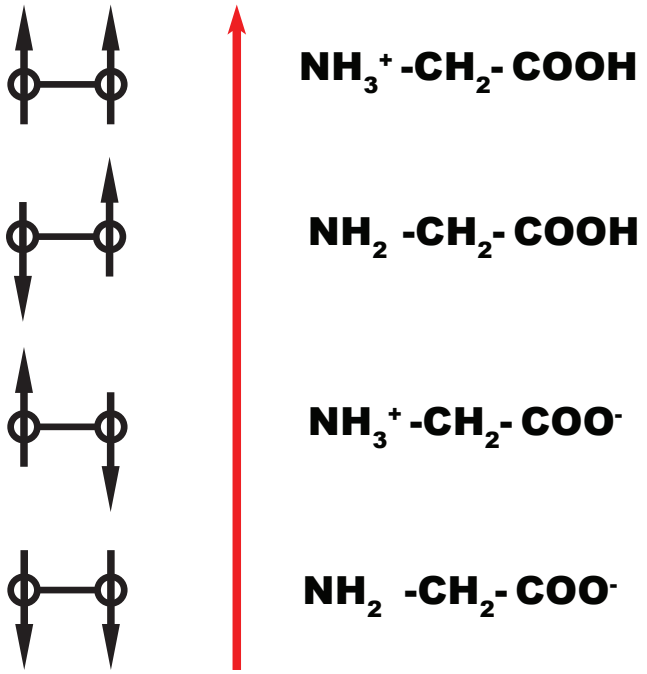


Figure 3.1: Lattice model of glycine. The red arrow represents either a magnetic field or a very low pH. On the right side of the arrow, we show the comparable states of glycine. We can model the protonation of glycine as a function of some force (solvent pH) with the same equations that we would use to model a ferromagnetic system as shown on the left. The direction of the arrows are equivalent to a state of protonation of the lobes of glycine.

Chapter 4

Synaptic Plasticity: A Process That Occurs in Small Volumes

While much of this thesis will discuss hemoglobin and glycine, it is not biophysically realistic to think about these molecules as naturally existing in small volumes with limited availability of ligand. We use them as tools to understand biological cooperativity on the smallest scale because they are useful tools in developing an understanding of cooperative biological processes that occur in small volumes. Having been a neuroscientist for the past 15 years, I have my biases. Synaptic plasticity is a process that occurs in small volumes and is thought to be at least partially mediated cooperative processes. A key component in synaptic plasticity is the differentation of low Ca^{2+} for the weakening of the strength of a synapse and high Ca^{2+} influx for the strengthening of synaptic connections. In subsequent chapters, I will use the life molecule, hemoglobin, to understand the mind molecule, calcium calmodulin kinase II (CaMKII).

4.1 Synaptic Plasticity

To the best of our knowledge, the synapse is the center of learning and memory (Bliss and Lomo [1973]). The synapse is a subcellular structure that has both a small volume and is diffusionally restricted under certain conditions (Bloodgood and Sabatini [2005], Sabatini et al. [2002]).

A beautiful rendition of the synapse—"The Synapse Revealed"—can be seen in Figure 4.1.

4.2 An Agnostic Approach to Modeling Calcium Calmodulin Kinase II

Calmodulin (CaM), a ligand of CaMKII, has two lobes, the N-terminal and C-terminal lobes. Each lobe can bind up to two Ca^{2+} ions, and even partially ligated CaM has the ability to phosphorylate CaMKII *in vitro* (Shifman et al. [2006]). Therefore, CaM can exist in at least nine different states, as the two lobes can each exist in the states of unbound, singly bound, and doubly bound to calcium. Each species of CaM can interact with a subunit of the dodecameric macromolecule CaMKII.

It has been reported that CaMKII itself can be phosphorylated in many positions including the following: T253 (Dosemeci and Reese [1993]), T286 (Lai et al. [1987], Miller and Kennedy [1986]), T305 (Colbran and Soderling [1990]), and T314 (Colbran and Soderling [1990]). Interestingly, dynamics that are apparent *in vitro* (Miller and Kennedy [1986]) may not exist *in vivo* (Mullasseril et al. [2007]). The

system is further complicated by the fact that that CaMKII exists as a heterogeneous mix of more than one isoform (α and β) in the synapse. While I have previously generated large models of CaMKII including many, but not all of these features, a true elucidation of a robust mechanism by which the synapse might differentiate between various calcium signals has not been shown.

4.3 Essential Features of a Molecular Switch that Can Differentiate Ca^{2+} Signals

Much of the work in synaptic plasticity research is directed toward understanding the switch-like behavior of CaMKII (Miller and Kennedy [1986]). In this thesis, we propose that the key to understanding how CaMKII might ignore weak Calcium signals while reacting to strong ones is a ubiquitious feature of positively cooperative molecules under a situation of limited resources. Furthermore, we agree with Lisman in that the "ensemble" that many bench-top experiments create is incomparable to the "ensemble" that exists in the cell (Sanhueza et al. [2011]). We hypothesize that two key features of the Ca^{2+}/CaM/CaMKII complex, bistability of CaMKII and limited resources of Ca^{2+}, allow for signaling differentiation that in isolation is a thermodynamically robust bench-top experiment.

4.3.1 CaMKII is Likely Bistable

In 2000, Zhabotinsky showed that CaMKII exists primarily in one of two states. In order for one to apply experimental results to a system including CaMKII to infer features of its activity in the synapse, the experiments should be performed under conditions that would be equivalent in size and molecular composition to the synapse. We will show that it is possible to predict non-ensemble behavior of CaMKII in the synapse based upon a simple understanding of cooperativity due to quantal effects mediated by quantal effectors. If one were to consider a synapse to be comparable to a bench-top experiment, this quantal behavior would be comparable to ergodic breaking.

4.3.2 The Synapse is a System of Limited Resources

There are not many Ca^{2+} ions in the synapse (Sabatini et al. [2002]). A low maintained local level of Ca^{2+} should be sufficient to circumvent the activation of CaMKII, even at local concentrations that might be thought sufficient to activate it. There are many analogies that can be drawn between a cooperative molecule in the synapse and the quantum mechanical treatment of a particle in a box.

4.4 What to Look for in the Case Studies

1. In Case I, glycine is useful because in our modified lattice model, it is directly analogous to spin pairing of electrons in a shell.

2. In the first half of Case II, Gibson's kinetic model of hemoglobin is discussed briefly to show that we can match kinetic as well as thermodynamic data.

3. In the second half of Case II, Ackers' thermodynamic model of hemoglobin is discussed to exemplify the scope of the problem. This system shows quantal behavior of positively cooperative models in small spaces.

4. In Case III, we will conclude with a "toy" model to show what the sufficient conditions are for non-ensemble-like behavior, and how this relates to the aforementioned points.

Figure 4.1: A zoomed in view of, The Synapse Revealed. Created by Graham Johnson 2004 $< graham@fivth.com >$ for The Howard Hughes Medical Institute Bulletin. Mr. Johnson's illustration of two synapses. The green spheres represent synaptic vesicles and are located in the presynaptic side of the synapse. The side directly opposing the presynaptic side is referred to as the postsynaptic side of the synapse in this case it is also known as a dendritic spine.

Chapter 5

Method Development

In this chapter, we initially discuss traditional calculus-based and stochastic methods of modeling chemical reaction networks. We then review and elaborate on an alternative method of stochastic modeling that has been named rule-based modeling.m Rule-based modeling was built for biochemists, but it often employs language that makes it inaccessible to biochemists and computational chemists. In this chapter, we redefine rule-based modeling for experimental and theoretical chemists. We do this without difficulty by referring to the history of fundamental components of rule-based modeling that existed before the coining of the term rule-based modeling for biological systems. Finally, we outline the theme by which rule-based modeling can be unified with existing chemical theory and its likely niche in the greater context of computational chemistry.

5.1 Deterministic Models of Chemical Reactions

In the late 19^{th} century, scientists began to create mathematical models of the evolution of chemical species in reactions using mass-action (Lund [1965]). The

use of calculus-based methods, such as differential equations, to model chemical species were a natural extension of this work. This type of modeling is termed deterministic because, given a model and a set of initial conditions, it will reproduce the same results on repeated trials. This is not the case for stochastic models which will be discussed later.

A standard deterministic approach is employed to provide a solution to a system of coupled differential equations that are defined by a detailed chemical reaction mechanism. Let us consider the stoichiometric reaction of A with B forming a discrete reaction product C:

$$A + B \underset{k_{-1}}{\overset{k_1}{\rightleftharpoons}} C \tag{5.1}$$

where k_1 is the forward second–order "rate constant" and k_{-1} is the reverse first-order "rate constant." The equilibrium constant for the reaction of Equation 5.1 is given by the law of mass-action, as follows:

$$K_1 = \frac{k_1}{k_{-1}}. \tag{5.2}$$

The complete kinetic description of the simple reaction of Equation 5.1 is as follows:

$$\frac{dA}{dt} = -k_{on}[A][B] + k_{off}[C] \tag{5.3}$$

$$\frac{dB}{dt} = -k_{on}[A][B] + k_{off}[C] \tag{5.4}$$

$$\frac{dC}{dt} = k_{on}[A][B] - k_{off}[C] \tag{5.5}$$

where *[A]* and *[B]* are the concentrations of reactants and *[C]* is the concentration of the reaction product. In a conventional formulation, the concentration units for an aqueous-phase reaction are expressed in units of moles per liter (i.e., mole L^{-1} or M); for a homogeneous gas-phase reaction, they are expressed in units of the numbers of molecules per cubic centimeter (i.e., molecules cm^{-3}). The forward rate constant k_{on} then has the corresponding units of either L $mole^{-1}$ s^{-1} or cm^3 $molec^{-1}$ s^{-1}, while the reverse rate constant k_{off} has units of s^{-1} in both cases.

5.2 Stochastic Simulations

A fundamental feature of stochastic simulations of particles is that they allow for complex interactions, such as those which might be described as non-ideal. Furthermore, they have a history that is nearly as old as the study of hemoglobin itself. A few years after Pauling [1935] proposed his model of hemoglobin/O_2 dynamics, Delbruck, who was also at Caltech at the time, proposed the use of stochastic methods for the study of autocatalytic systems. As early as the 1950s (Singer [1953], Renyi [1953]), it was shown that deterministic methods could not reproduce stochastic results. Singer's result was strongly based on statistical fluctuations leading to irreproducible results. Renyi showed that as molecular number approached one, deterministic methods deviated from stochastic results, but that the differences were minimal. In Case Studies III and IV, I will give results that show both minor and dramatic deviations.

A nice feature of stochastic simulations is that we are not confined by the same

limits as deterministic calculus-based models. We are not confined to talking about the average evolution of populations, and we can explicitly treat non-ideal interactions with reaction rate functions or "rules". The implementation of such non-ideal interactions was discussed by Pauling as early as 1935, but only explicitly elaborated in statistical models by Chay in 1973.

Gillespie's stochastic simulation algorithm (SSA) has become synonymous with stochastic simulation of chemical reaction networks. Gillespie's method is an exact method, meaning that it represents the probabilistic distribution without approximation. Gillespie [1976] proposed a generalized algorithm that can be used to reproduce the computational results of a deterministic model for well-mixed chemical reactions using stochastic approaches. In order to evaluate the merits of using the stochastic approach to quantitatively model cooperative chemical systems, we will use Gillespie's SSA to fit observed experimental results, and then compare these results to those obtained with a standard numerical integration of the coupled ordinary differential equations that describe the details of the chemical or biochemical reaction mechanism.

5.2.1 Gillespie's Stochastic Simulation Algorithm

According to Gillespie's SSA, reaction propensity functions are defined and evaluated for each reaction under consideration. The reaction propensity provides a measure of the likelihood that a chemical reaction or a single step in an overall reaction mechanism will occur (i.e., high propensity is equivalent to a high reaction

probability). The reaction propensity functions are used to generate a time-to-the-next-event function.

The reaction propensity is directly related to the probability that a specific reaction or a single step in a reaction mechanism will actually occur. The reaction propensity is formulated in terms of a reaction parameter which is analogous in its general form to the corresponding reaction rate, a deterministic rate.

The intrinsic reversibility (e.g., a reversible chemical reaction that reaches true equilibrium or reaches a steady-state condition such that $\Delta G = 0$, $\Delta G^{\circ} = -RT(\ln K)$, and $-d[A]/dt = -d[B]/dt = d[C]/dt$) of the reaction of Equation 5.1 allows for the reaction propensities of the forward and reserve steps to be written in a straightforward manner. For example, the reaction propensity for the forward bimolecular reaction between A and B is written as

$$propensity^2 = k_{on}(A \times B) \tag{5.6}$$

where k_{on} is a second-order stochastic reaction parameter, A is the total number of molecules of type A in the reacting system during a single time-step iteration, and B equals the total number of molecules of type B in the system during the same iteration. Therefore, the product $A \times B$ gives the total number of possible combinations of reaction events that can occur during a single time–step.

In the case of the reverse unimolecular reaction step, the corresponding reaction

propensity is given by:

$$propensity^1 = k_{off}(C) \tag{5.7}$$

where k_{off} is a first-order stochastic reaction parameter and C is the total number of molecules of type C in the reacting system over a single iteration or time-step.

In order to determine an appropriate time-step, we take the sum of the reaction propensities to generate total propensity (TP) as follows:

$$total\ propensity(TP) = \sum_i (reaction\ propensity). \tag{5.8}$$

This total propensity is then used to generate an exponential random variable that is equated to the time that the next individual molecular reaction will occur (τ). The characteristic time-step τ is thereby obtained from the following relationship:

$$\tau = \left| \frac{1}{(TP)} \ln\left(\frac{1}{URN}\right) \right| \tag{5.9}$$

where *URN* is a uniformly distributed random number between 0 and 1. It should be noted that, as the total propensity of the system increases, τ decreases.

After the time-step τ is determined, the time to the next reaction event is chosen. In turn, the reaction event that is to occur is chosen randomly with a bias toward reactions with a greater propensity. Finally, the number of molecules are

updated appropriately to reflect the reaction that occurred.

Gillespie's SSA approach [1976] was employed for the stochastic modeling comparison to the corresponding deterministic solutions. The SSA was implemented in Scipy.

1. Computation is initiated at time $t = 0$ with the iteration number also set to zero.
2. The reaction propensities are calculated.
3. The appropriate time–step τ is determined.
4. Given the computed reaction propensities, the specific reaction events during a single iteration are determined.
5. The current iteration is increased by 1 until an END value is reached at a maximum set time, or a limiting number of iterations is reached.
6. The total number of molecules in the reacting system is updated based upon the specific chemical reaction that is allowed to occur and the sequence is looped back to step 1 above.

Stochastic simulations were run to equilibrium for 200 repetitions and the data was averaged. Sample codes are provided in the Appendix.

While the SSA was derived from statistical mechanics, lattice-like models were not explored until approximately thirty years later in the form of rule-based modeling.

5.2.2 Functional Elements of Macromolecules

Typical biochemists might have little intuition concerning the $\pi-$calculus or $\kappa-$calculus, referred to by Danos and colleagues [2004]. Furthermore, renaming molecules as agents is by no means required. It has been generally understood

since the early 19th century that macromolecules are composed of functional elements that affect the rates of reactions.

Again, the earliest application of a stochastic approach to model an autocatalyst such as hemoglobin is attributed to Delbruck [1940]. Using Delbruck's approach, we normalize the molecular numbers to one Hb macromolecule. This is analogous to constructing a finite element around a single molecule. Additionally, my work is closely related to other rule-based modeling packages in that the most fundamental algorithms should be similar. These similarities relate to the realization that reaction rates are dynamic and that biochemical systems are non-ideal. One important difference that we will highlight is that differential equations and stochastic simulations predict different behavior for cooperative systems in small volumes. Another difference in my approach is that we will provide source code that others may run to gain insight into rate laws for non-ideal systems.

5.3 Additional Details

5.3.1 Hardware

Kinetic simulations were run on the following systems:

- A custom-built computer with the following specifications:
  - Intel Quad Core Duo
  - 8Gb of RAM
  - Abit IP-35Pro motherboard

  - running Ubuntu Linux

- An iMac running an intel core i5 processor

5.3.2 Software

Software codes written in Python Pyt23, Scipy, Matplotlib, and Pylab were used for both deterministic and stochastic modeling.

5.3.3 Optimization

Code optimization was constrained by the data of Mills and Ackers et al. [1976] using the Nedler-Mead method or the downhill simplex method . This algorithm is contained in Scipy(Sci) in the function scipy.optomize.fmin.

5.3.4 Data Digitization

Four data-sets of Mills [1976] were digitized using the open source application g3data.

5.3.5 Deterministic Methods

The corresponding systems of differential equations that correspond to the detailed reaction mechanisms for two different experimental case studies were numerically integrated using the variable-coefficient ordinary differential equation solver, with a fixed-leading-coefficient implementation (VODE) . This method was also implemented via Scipy through the integrate.ode method. Sample codes are

provided in the appendices. In some cases, Kinpy (Srinivasan) was used to auto-generate complicated reaction schemes.

Chapter 6

Case I. Glycine Protonation

The acid/base chemistry of the amino acid glycine provides a simple example of a chemical process involving protonation and deprotonation of a bifunctional molecule that exhibits negative cooperativity, where the two functional groups are the amino group -NH_2 and the carboxylic acid group -$COOH$. It should be noted that the negative cooperativity seen in glycine protonation is due to charge repulsion of protons that may bind to each terminus of glycine. An example of this type of reaction system is illustrated in Figure 6.1 (Williamson [2008]).

The chemical reaction network depicted in Figure 6.2 represents the various states of protonation and deprotonation of the glycine molecule as a function of solution pH. Of particular interest is the zwitterion form of glycine, in which both a positively charged, protonated amino group co-exists with a negatively charged, deprotonated carboxylate group. During the sequential protonation and deprotonation processes involving the neutral glycine molecule, negative cooperativity is realized during the addition of either proton, which effectively decreases the affinity of glycine for the subsequent proton.

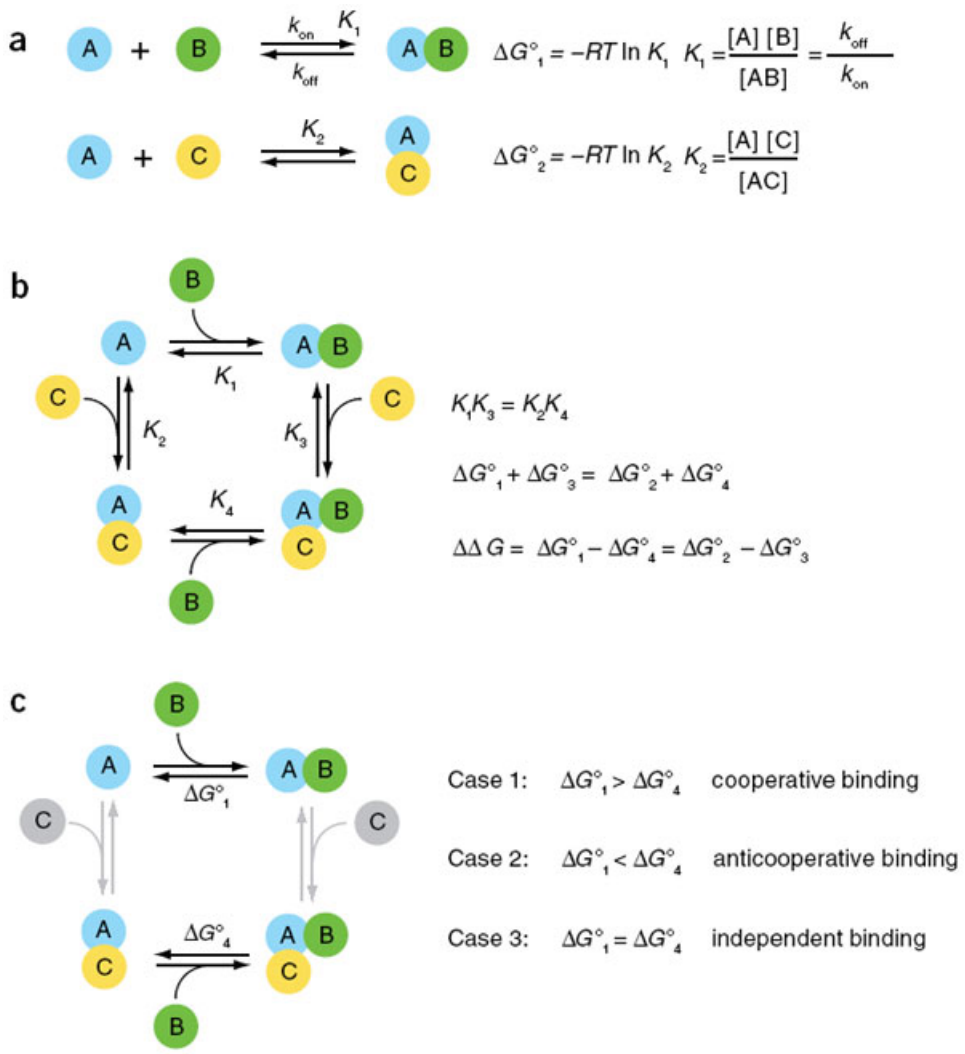


Figure 6.1: a) A hypothetical set of bimolecular complexes between component A and two other components (B and C), with the rate constants, equilibrium constants, and free energies for complex formation. b) A thermodynamic cycle for formation of a ternary complex ABC by two different possible routes: either B binds first or C binds first. There are four equilibrium constants that describe the formation of the various complexes. Because they converge on the common product ABC, the thermodynamics must be independent of the pathway chosen around the cycle, and constraints are placed on the relative values of the equilibrium constants and hence the free energies. The thermodynamic coupling free energy ($\Delta\Delta G$) gives the difference between the binding of one component in the presence of the other. c) The definition of cooperativity in terms of binding of B in the presence or absence of C. The two vertical binding reactions are in gray to emphasize the comparison of ΔG°_1 and ΔG°_4 . If B binds preferentially in the presence of C, the binding is cooperative. If B binds worse in the presence of C, the binding is anticooperative. In the third case, binding of B is independent of C, and there is no cooperativity. Adapted from Williamson et al. [2008].

6.1 Model Description

6.1.1 Standard Scheme

The standard formuation of glycine protonation dynamics is depicted in Figure 6.2. While there are only two binding sites that can exist in one of two states in our system, we elaborate all possible eventualities to get around the inherently non-ideal nature of the system. This elaboration of possible trajectories leads to the formation of webbed structures, as shown in Figure 6.2. The reaction network illustrated in Figure 6.2 can be written as follows:

$$\mathrm{NH_2CH_2CO_2^-} + \mathrm{H^+} \overset{K_a}{\rightleftharpoons} \mathrm{^+NH_3CH_2CO_2^-} \tag{6.1}$$

$$\mathrm{NH_2CH_2CO_2^-} + \mathrm{H^+} \overset{K_b}{\rightleftharpoons} \mathrm{NH_2CH_2CO_2H} \tag{6.2}$$

$$\mathrm{^+NH_3CH_2CO_2^-} + \mathrm{H^+} \overset{K_b c_{ab}}{\rightleftharpoons} \mathrm{^+NH_3CH_2CO_2H} \tag{6.3}$$

$$\mathrm{NH_2CH_2CO_2H} + \mathrm{H^+} \overset{K_a c_{ab}}{\rightleftharpoons} \mathrm{^+NH_3CH_2CO_2H} \tag{6.4}$$

where K_a and K_b are the primary acid association constants for the addition of protons to form the protonated amino and carboxy groups, respectively; c_{ab} is a cooperativity factor that accounts for the electronic interactions between the two sites.

The data elements for this scheme represent populations of molecules in various states. For this two-particle formulation, we elaborate each state in Table 6.1. For clarity, we show that the population sizes that can be modeled with this for-

mulation are limited only by the precision of the simulation machine in Table 6.2.

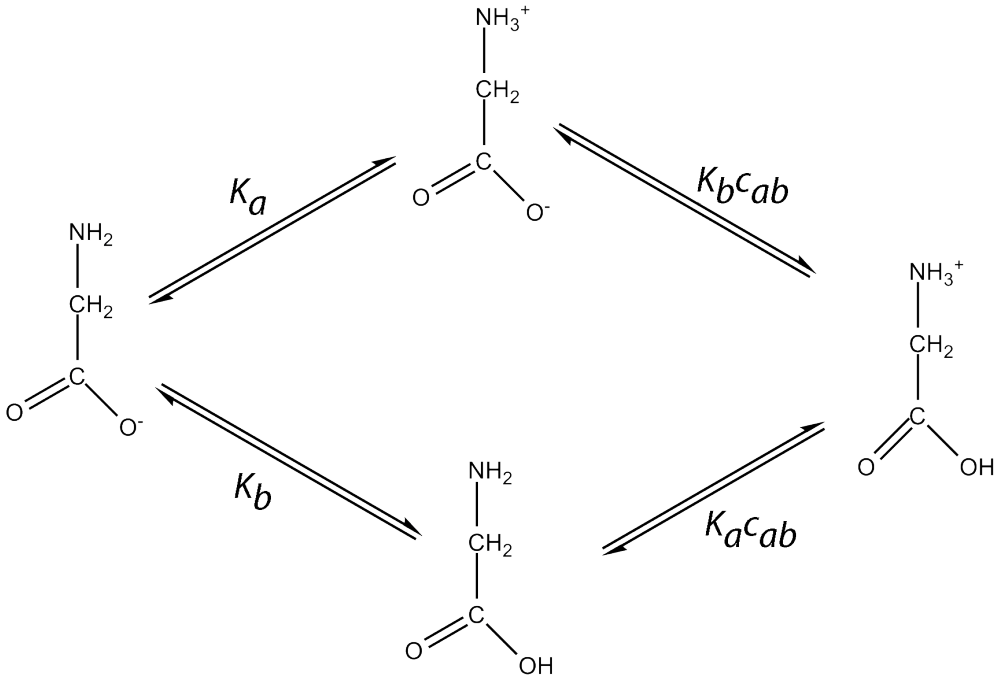


Figure 6.2: The chemical reaction network representing the various states of protonation and deprotonation that exist for the glycine molecule as a function of solution pH. Of particular interest is the zwitterion form of glycine in which a positively charged, protonated amino group co-exists with the negatively charged, deprotonated carboxylate group. The primary acid association constants K_a and K_b account for the equilibrium gain of a proton to form the protonated amino and carboxy groups, respectively. c_{ab} is a cooperativity factor accounting for the electronic interactions between the two sites. The corresponding values of K_a, K_b, and c_{ab} are $K_a = 5.0 \times 10^9 M^{-1}$, $K_b = 2.0 \times 10^4 M^{-1}$, and $c_{ab} = 0.01$. The relatively small value of c_{ab} indicates a strong electrostatic repulsion between sites.

Table 6.1: A standard tableau depicting the reactions required to determine the speciation of glycine as a function of pH.

	Reactants	Products	Equation
1	$NH_2CH_2COO^-$,H^+	$^+NH_3CH_2COO^-$	6.1
2	$NH_2CH_2COO^-$,H^+	NH_2CH_2COOH	6.2
3	$^+NH_3CH_2COO^-$,H^+	$^+NH_3CH_2COOH$	6.3
4	NH_2CH_2COOH,H^+	$^+NH_3CH_2COOH$	6.4

Table 6.2: A standardized formulation for glycine protonation vs. pH. The data elements are valued from zero to infinity.

Data Element	Value Range	State
Y[1]	$0 - \infty$	Fully Deprotonated
Y[2]	$0 - \infty$	Carboxyl Terminus Protonated
Y[3]	$0 - \infty$	Amino Terminus Protonated
Y[4]	$0 - \infty$	Fully Protonated

6.1.2 Standard Formulation of Glycine/H+ with Differential Equations

The coupled system of differential equations are written as follows:

$$\frac{d[A]}{dt} = -K_a \times k_{off}[H][A] - K_b \times k_{off}[H][A] + k_{off}[B] + k_{off}[C] \tag{6.5}$$

$$\frac{d[B]}{dt} = K_a \times k_{off}[H][A] - (c_{ab} \times K_b \times k_{off})\,[H][B] + k_{off}[D] - k_{off}[B] \tag{6.6}$$

$$\frac{d[C]}{dt} = K_b \times k_{off}[H][A] - (c_{ab} \times K_a \times k_{off})\,[H][C] + k_{off}[D] - k_{off}[C] \tag{6.7}$$

$$\frac{d[D]}{dt} = (c_{ab} \times K_a \times k_{off})\,[H][C] + (c_{ab} \times K_b \times k_{off})\,[H][B] - 2 \times k_{off}[D] \tag{6.8}$$

where A = $NH_2CH_2COO^-$, B = $^+NH_3CH_2COO^-$, C = NH_2CH_2COOH, and D = $^+NH_3CH_2COOH$. The break-down of the tableau for Equations 6.5–6.8 are depicted

in Table 6.1 and represented graphically in Figure 6.1.

6.1.3 Standard Formulation of Glycine/H+ with Stochastic Equations

The standard stochastic approach to simulate the above system of differential equations is as follows:

$$prop_1 = sK_a \times k_{off} \times H \times A \tag{6.9}$$

$$prop_2 = sK_a \times k_{off} \times H \times A \tag{6.10}$$

$$prop_3 = sK_b \times k_{off} \times c_{ab} \times H \times B \tag{6.11}$$

$$prop_4 = sK_a \times k_{off} \times c_{ab} \times H \times C \tag{6.12}$$

$$prop_5 = k_{off} \times B \tag{6.13}$$

$$prop_6 = k_{off} \times C \tag{6.14}$$

$$prop_7 = k_{off} \times D \tag{6.15}$$

$$prop_8 = k_{off} \times D \tag{6.16}$$

where sK_x is the stochastic version of the deterministic K_x. While the A, B, C, and D values from the previous subsection represent molar quantities, these stochastic A, B, C, and D represent explicit molecule counts.

6.1.4 Modified Scheme of Glycine/H+ Employing Quantal Effectors

An alternative approach to modeling the protonation/deprotonation of glycine as a branched reaction is to represent the termini separately, as shown in Figure 6.3. Since the affinity constants for the two termini are orders of magnitude apart in value, we can break up the reactions and model the system with the following mechanism:

$$(\mathrm{NH_2}-) + \mathrm{H^+} \quad \overset{K_a}{\rightleftharpoons} \quad (^{+}\mathrm{NH_3}-) \tag{6.17}$$

$$(-\mathrm{COO^-}) + \mathrm{H^+} \quad \overset{K_b \times c_{ab}}{\rightleftharpoons} \quad (-\mathrm{COOH}). \tag{6.18}$$

Equations 6.17 and 6.18 approximate the true steady-state condition for the degree of glycine protonation. However, the system is more accurately represented by Equations 6.19 and 6.20.

$$(\mathrm{NH_2}-) + \mathrm{H^+} \quad \overset{if\mathrm{COO^-}(K_a)else(K_a \times c_{ab})}{\rightleftharpoons} \quad (^{+}\mathrm{NH_3}-) \tag{6.19}$$

$$(-\mathrm{COO^-}) + \mathrm{H^+} \quad \overset{if\mathrm{NH_2}-(K_b)else(K_b \times c_{ab})}{\rightleftharpoons} \quad (-\mathrm{COOH}) \tag{6.20}$$

where the rates for the forward reactions should be read as: *"if the other terminus is in state x, then this reaction will proceed with rate y"*. Otherwise, it will proceed with rate z (*"if the rate is stated as if x (y) else(z)"*). This type of scheme includes the state

of the other terminus in each reaction, thereby explicitly including the non-ideal nature of the reaction. To be clear, this is a three-particle formulation that includes reactants, products, and quantal effectors.

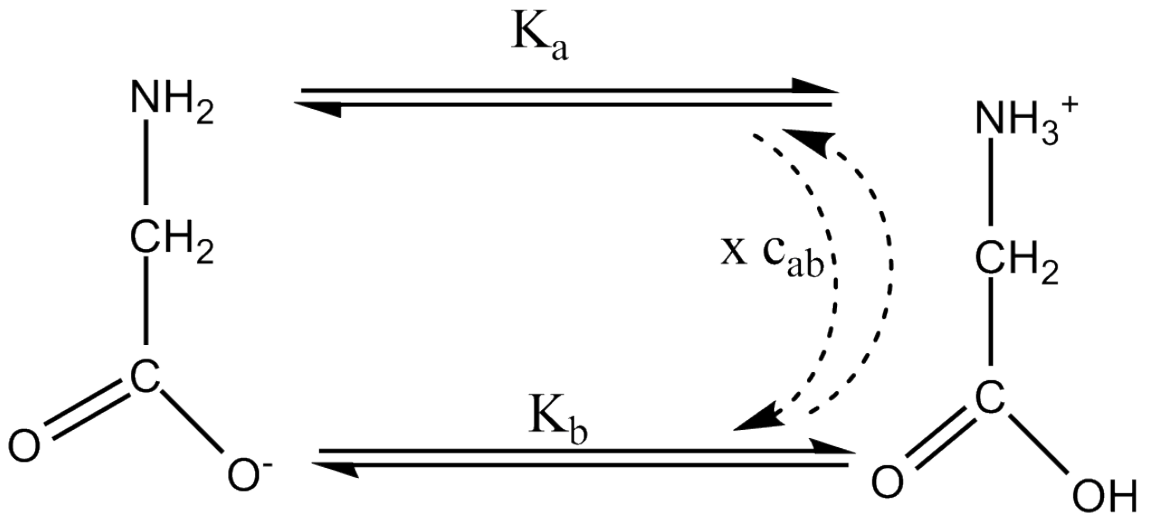


Figure 6.3: Glycine protonation state as impacted by "effectors." The state of the non-reacting species affects the reacting species (e.g., the rate at which the amino terminus is protonated is affected by the protonation state of the carboxylate terminus). This is a depiction of an explicit use of a reaction rate function. The dotted lines are indicating the direction of deprotonation of the opposite terminus since this system exhibits negative cooperativity with respect to protonation.

In the scheme shown below, the reaction rate constants are replaced by reaction rate functions where the rate of protonation of a terminus is directly affected by the protonation state of the opposing terminus. Equations 6.19 and 6.20 represent a logistic three-particle formulation of glycine/H^+ association. An analytic formulation of the three-particle representation of glycine/H^+ follows.

$$(\mathrm{NH_2}-) + \mathrm{H}^+ \xrightleftharpoons{K_a \times c_{ab}^{(-\mathrm{COOH})}} (^+\mathrm{NH_3}-) \tag{6.21}$$

$$(-\mathrm{COO}^-) + \mathrm{H}^+ \xrightleftharpoons{K_b \times c_{ab}^{(^+\mathrm{NH_3}-)}} (-\mathrm{COOH}) \tag{6.22}$$

The break-down of the tableau for Equations 6.19 and 6.20 (a logistic treatment of "the rules") or 6.21 and 6.22 (an analytic treatment of "the rules") is depicted in Table 6.3.

Table 6.3: A "rule-based" type reaction scheme for glycine/H^+

	Reactants	Products	Quantal Effectors	Old Equation	New Equation
1	NH_2-,H^+	$^+NH_3$-	-COO^-/-COOH	6.1, 6.3	6.21
2	-COO^-,H^+	-COOH	NH_2-/$^+NH_3$-	6.2,6.4	6.22

Table 6.4: A modified formulation of the data model for glycine protonation

Y[1]	Y[2]	Y[3]	Y[4]	Glycine Protonation State
1	1	0	0	Fully Deprotonated
1	0	0	1	Carboxyl Terminus Protonated
0	1	1	0	Amino Terminus Protonated
0	0	1	1	Fully Protonated

6.1.5 Modified Formulation of the Glycine/H^+ Model

$$\text{prop[0]} = K_a \times k_{off1} \times c_{ab}^{(\text{y[4]})} \times \text{y[0]} \times \text{y[1]} \tag{6.23}$$

$$\text{prop[1]} = K_b \times k_{off2} \times c_{ab}^{(\text{y[3]})} \times \text{y[0]} \times \text{y[2]} \tag{6.24}$$

$$\text{prop[2]} = k_{off1} \times \text{y[3]} \tag{6.25}$$

$$\text{prop[3]} = k_{off2} \times \text{y[4]} \tag{6.26}$$

To orient the reader to the explicit encoding that is being used in this model, we

have included a portion of Python code to simulate glycine binding here as Source Code 1. Lines 2–5 provide our encoding for a fully unbound glycine, as depicted in Table 6.4. Lines 7–10 are equivalent to Equations 6.23–6.26. For those not familiar with Python, "cab**y[4]" is equivalent to $c_{ab}^{y[4]}$.

Algorithms and Source Code 1: A snippet of code from Source Code 4 . Line 1 converts a molar quantity to discrete values.

```
1       y[0]=H*nA*vol
        y[1]=1
        y[2]=1
        y[3]=0
        y[4]=0
6       while tend>t:
            prop[0]=Ka*koff*cab**y[4] * y[0] * y[1]
            prop[1]=Kb*koff*cab**y[3] * y[0] * y[2]
            prop[2]=koff * y[3]
            prop[3]=koff * y[4]
```

6.2 Computational Results

Glycine was modeled using both deterministic and stochastic simulation methods, as shown in the green and blue curves in Figure 6.4. The red curve in Figure 6.4 shows the expected protonation profile, modeled with deterministic methods, for glycine if protonation of the C-terminus and the N-terminus were truly independent of one another. As the protonations of the termini of glycine are not independent, experimental curves of glycine protonation are better represented by the green and blue curves in Figure 6.4.

The tableau of the typical reaction scheme, as shown in Table 6.1, treats the ef-

fects of the state of the opposing binding-site in an indirect manner. For further details, see the deterministic treatment (Equations 6.5–6.8) and the standard stochastic treatment (Equations 6.9–6.16). The green curve is a standard deterministic fit using the scheme presented in Figure 6.2. The blue curve is a rule-based stochastic formulation of the glycine protonation curve, using the modified representation of glycine as presented in Figure 6.3. The stochastic reaction rate functions shown in Equations 6.19 and 6.20 or 6.21 and 6.22 treat the state of the non-reacting subunit as a parameter that affects the affinity of the reacting terminus for a proton. This three-particle class reaction scheme is also shown in Table 6.3.

The data elements for the standard formulation are real scalars in that they can be valued from 0 to infinity (∞), or the machine's closest approximation of infinity, as shown in Table 6.2. Note that effectors are explicitly included in the stochastic propensity functions for the reactions, allowing us to represent the deterministic reaction schemes with high fidelity and with fewer reaction expressions. This stochastic approach is a modified version of Gillespie's stochastic simulation algorithm, and it falls into the class of rule-based approaches.

6.3 Discussion

In both the standard and modified forms, four data elements are required to model the state of protonation of glycine as a function of solution pH. The standard, population-based data model can represent the activity of an infinite number of glycine molecules, as shown in Table 6.2; this is due to the use of scalars as

the essential data elements. In the modified formulation of glycine protonation, the specific data model must be duplicated for each added glycine molecule and the values are read like a truth table where 1 indicates the current sub-state of a termini of the glycine molecule, as shown in Table 6.4. We use either integer (0 or 1) or Boolean elements (True and False) as the essential data elements.

In the standard model, there are four reaction rate equations, while in the modified mode, there are eight stochastic reactions. Again, an increase in the population size does not increase the number of reactions required in the standard method, but in the modified system, the four reactions must be duplicated for each added molecule of glycine in the system. This distinction is due to the differences between a population model and a single-protein model, a complex, finite element method.

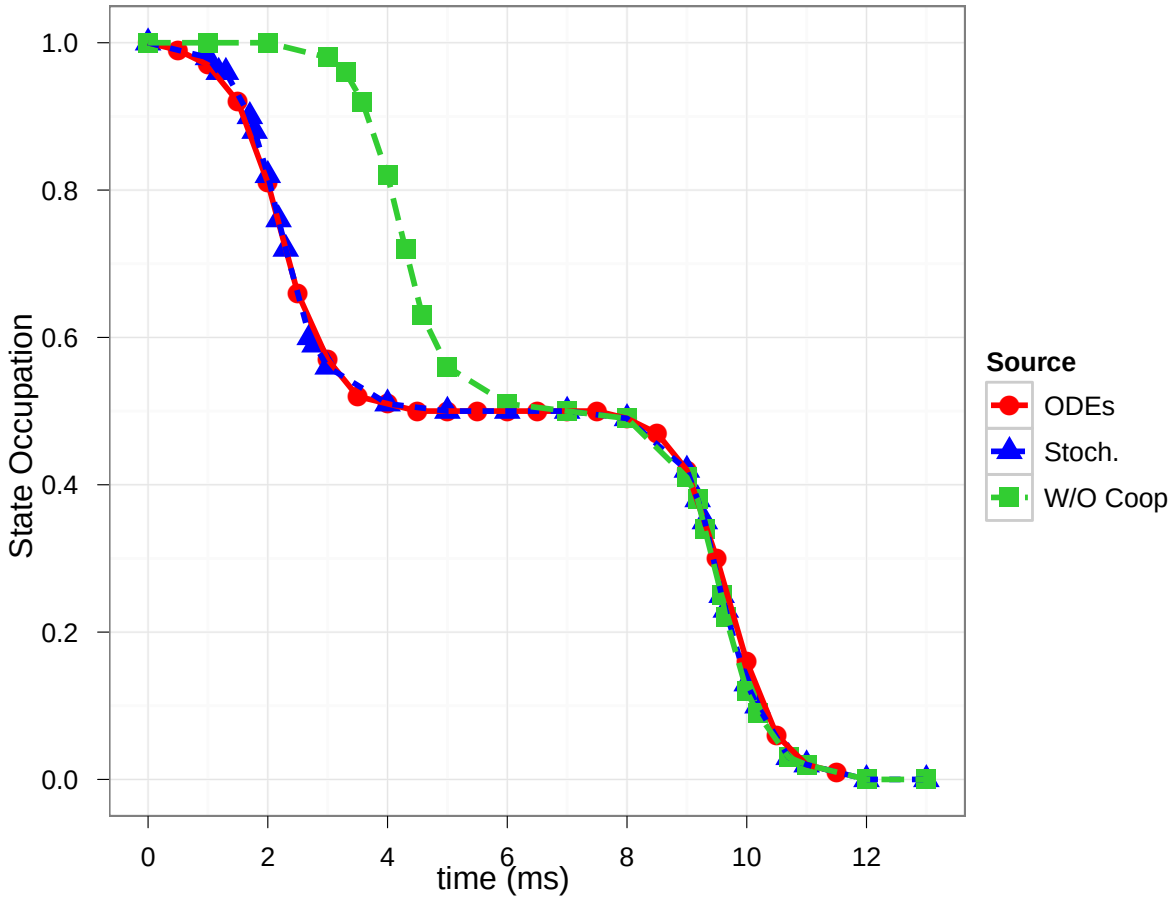


Figure 6.4: Simulation of glycine protonation. (Red) No charge repulsion. (Blue) A deterministic model of glycine protonation with charge repulsion. (Green) A stochastic simulation of glycine protonation with charge repulsion. Adapted from Figure 2-9a of Wyman and Gill [1990].

Chapter 7

Case II: Kinetic and Thermodynamic Models for Oxygen Uptake by Hemoglobin

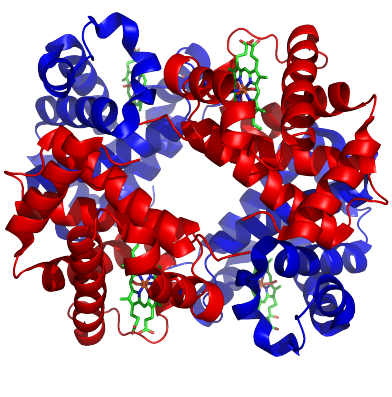
A single hemoglobin molecule consists of four protein subunits—two α and two β subunits (Figure 7.1(a)). The four polypeptide chains are bound to each other by salt bridges, hydrogen bonds, and hydrophobic interactions. The α subunits and β subunits contain 141 and 146 amino acid residues, respectively. The various combinations and interactions among the subunits are assigned the following notation: $\alpha x \beta y$.

Each subunit has an associated heme functional group. Each heme group, in turn, consists of a porphyrin bound to the equatorial positions of an Fe octahedral complex, so that the porphyrin encircles the Fe atom (Figure 7.1(b)). The Fe atom is also axially ligated to an imidazole nitrogen of a histidine residue of the surrounding protein. In the non-oxygenated state, the axial, sixth coordination site is occupied by an apically bound water molecule. In this unbound state, the iron is in the Fe(II) state.

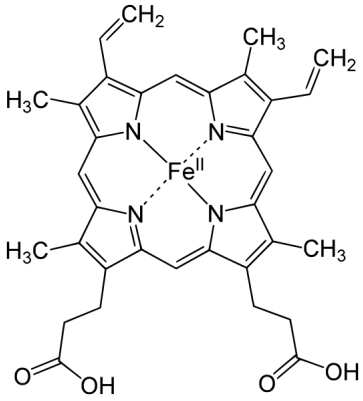
The hexa-coordinated Fe(II) center reacts with molecular oxygen to form, initially, an oxygen adduct or complex. After complexation, there is a nominal one-electron transfer from Fe(II) to the bound oxygen molecule, forming an Fe(III)-superoxo (O_2^-.) complex. Oxygen is complexed in a bent configuration, in which one oxygen atom forms a covalent bound with Fe, while the other oxygen is skewed at an angle (personal communication M.R. Hoffmann and N.B. Ford).

The reversible binding and release of O_2 in tetrameric hemoglobin is classified as a cooperative process, in which both the thermodynamics and kinetics of O_2 binding are enhanced as oxygen is added in sequence. For example, the first-bound oxygen will influence the conformational shape of the remaining binding sites, such that the subsequent oxygen additions are more favorable. The apparent cooperative effect is illustrated simply by the observed sigmoidal oxygen uptake curves as shown in Figure 7.1(c).

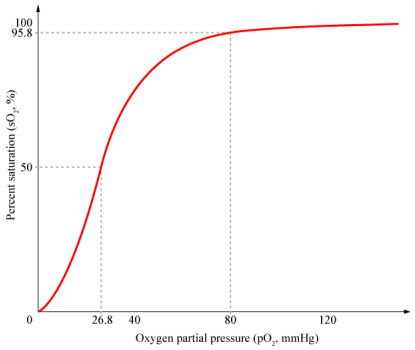
Adair [1925a] showed that, structurally, hemoglobin is a complex aggregate of four individual subunits. Furthermore, Adair proposed that the observed oxygen saturation curves for hemoglobin were the result of increasing equilibrium constants for the stepwise oxygen-binding steps, i.e., as a function of oxygenation level of hemoglobin, which in turn is a function of the solution phase O_2 concentration. The stepwise complexation of oxygen by hemoglobin is intrinsically non-ideal, in that hemoglobin's subunits can influence the extent and rate of oxygen binding to the other subunits. With these considerations in mind, Adair [1925a] proposed the



(a) ***Ribbon Structure***



(b) ***the Heme unit***



(c) ***Prototypical Saturation Plot***

Figure 7.1: a) A ribbon structural representation of hemoglobin showing the two α and two β subunits, each with an embedded heme group. b) The square-planar, top-down view of the primary heme group with Fe(II) bound in the center of the porphyrin ring. Not shown are the two apical open coordination sites above and below the ring to complete the octahedral configuration. c) The characteristic sigmoidal oxygen uptake curve exhibited by hemoglobin, which is also taken as an indicator of cooperativity.

following sequence of reversible reactions to describe O_2 uptake:

$$Hb + O_2 \overset{K_1}{\rightleftharpoons} Hb(O_2) \tag{7.1}$$

$$Hb(O_2) + O_2 \overset{K_2}{\rightleftharpoons} Hb(O_2)_2 \tag{7.2}$$

$$Hb(O_2)_2 + O_2 \overset{K_3}{\rightleftharpoons} Hb(O_2)_3 \tag{7.3}$$

$$Hb(O_2)_3 + O_2 \overset{K_4}{\rightleftharpoons} Hb(O_2)_4 \tag{7.4}$$

where K_1, K_2, K_3, and K_4 are the successive equilibrium binding constants. Given this stepwise equilibrium mechanism, the fraction of O_2 bound ($\bar{Y}$) is, therefore, given by

$$\bar{Y} = \frac{K_1 x + 2K_1K_2x^2 + 3K_1K_2K_3x^3 + 4K_1K_2K_3K_4x^4}{4(1 + K_1x + 2K_1K_2x^2 + 3K_1K_2K_3x^3 + 4K_1K_2K_3K_4x^4)}, \tag{7.5}$$

where *x* equals the concentration or partial pressure of oxygen.

Pauling [1935] graphically represented Adair's model for oxygen uptake by hemoglobin and then computed the corresponding oxygen saturation curves that result from different graphical topologies. Pauling fit the sets of possible saturation curves to actual experimental O_2-uptake data in order to deduce the apparent structural relationships among the four hemoglobin subunits. Pauling analyzed the possible topologies for a symmetric four-membered system. However, Pauling incorrectly concluded that hemoglobin has a "ring-like" structure rather than a "fully-connected" tetrahedral structure [Pauling, 1935].

Alternative models, such as the "allosteric", or concerted, model of Monod et

al. (MWC) [1965] and the "sequential" model of Koshland et al. (KNF) [1966], have been proposed to account for the detailed kinetics of O_2 uptake and binding by Hb. Ackers et al. [2005] studied the detailed thermodynamics of Hb/O_2, including the elusive intermediate states. They showed that the Monod et al. [1965] approach did not provide a good fit for the experimentally observed thermodynamics of oxygen binding. The model of Koshland et al. [1966] has also been shown to be lacking. Ackers [2006] proposed a model for O_2 binding by Hb that builds on decades of his own research and exhibits features of both the KNF and MWC models. Before we discuss the thermodynamic model of Ackers, we will discuss the kinetic model of Gibson.

7.1 Kinetic Model

In order to develop an alternative stochastic approach to model O_2 uptake by hemoglobin, we use the data of Gibson [1970] to quantitatively constrain the empirical data for sequential binding of O_2 to the four identifiable subunits of Hb. Gibson experimentally determined the kinetics of O_2 binding by Hb using stopped-flow UV-Vis spectrophotometry. With this rapid kinetic technique, he determined the individual apparent forward and reverse rate constants corresponding to the values of each one of the apparent equilibrium binding constants, K_1, K_2, K_3, and K_4. The kinetic and thermodynamic data are shown below in Table 7.1.

Table 7.1: Kinetic and thermodynamic data obtained by Gibson for the stoichiometric reaction of oxygen with hemoglobin

Affinity (M^{-1})	Forward-Rate ($M^{-1}\ s^{-1}$)	Reverse-Rate (s^{-1})
$K_1 = 9.0 \times 10^3$	$k_1 = 17.1 \times 10^6$	$k_{-1} = 1900$
$K_2 = 2.1 \times 10^5$	$k_2 = 33.2 \times 10^6$	$k_{-2} = 158$
$K_3 = 9.1 \times 10^3$	$k_3 = 4.89 \times 10^6$	$k_{-3} = 539$
$K_4 = 6.6 \times 10^5$	$k_4 = 33.0 \times 10^6$	$k_{-4} = 50.0$

Table 7.2: A standard formulation of the reaction scheme of Hb/O_2 by Gibson

	Reactants	Products	Equation
1	Hb, O_2	HbO_2	7.1
2	HbO_2, O_2	HbO_4	7.2
3	HbO_4, O_2	HbO_6	7.3
4	HbO_6, O_2	HbO_8	7.4

7.1.1 Standard Deterministic Formulation of Gibson's Model of Hemoglobin/O_2

Gibson's early sequential binding model for dioxygen's complexation to hemoglobin is depicted in standard form as written in Equations 7.1–7.4. The breakdown of the mass-action, two-component formulation is shown in Table 7.2. The data model for the standard formulation is depicted in Table 7.3. The coupled system of differential equations are written as follows:

$$\frac{d[Hb]}{dt} = -k_1[Hb][O] + k_{-1}[HbO] \tag{7.6}$$

$$\frac{d[HbO]}{dt} = k_1[Hb][O] - k_{-1}[HbO] - k_2[HbO][O] + k_{-2}[HbO_2] \tag{7.7}$$

$$\frac{d[HbO_2]}{dt} = k_2[HbO][O] - k_{-2}[HbO_2] - k_3[HbO_2][O] + k_{-3}[HbO_2] \tag{7.8}$$

$$\frac{d[HbO_3]}{dt} = k_3[HbO_2][O] - k_{-3}[HbO_3] - k_4[HbO_3][O] + k_{-4}[HbO_3] \tag{7.9}$$

$$\frac{d[HbO_3]}{dt} = k_4[HbO_3][O] - k_{-4}[HbO_4] \tag{7.10}$$

$$\frac{d[O]}{dt} = -k_1[Hb][O] - k_2[HbO][O] - k_3[Hb_2][O] - k_4[Hb_3][O] + k_{-1}[HbO] + k_{-2}[HbO_2] + k_{-3}[HbO_3] + k_{-4}[HbO_4] \tag{7.11}$$

where $[O]$ is the concentration of oxygen, $[Hb]$ is the concentration of deoxygenated

Table 7.3: A standardized formulation for Gibson's oxygenation of hemoglobin. The data elements are real valued from zero to infinity.

Data Element	Molec. Id	Value Range	Concentration in State
Y[1]	$[Hb]$	$0 - \infty$	Fully Deoxygenated Hemoglobin
Y[2]	$[HbO]$	$0 - \infty$	Singly Oxygenated Hemoglobin
Y[3]	$[HbO_2]$	$0 - \infty$	Doubly Oxygenated Hemoglobin
Y[4]	$[HbO_3]$	$0 - \infty$	Triply Oxygenated Hemoglobin
Y[5]	$[HbO_4]$	$0 - \infty$	Fully Oxygenated Hemoglobin

hemoglobin, $[HbO]$ is the concentration of hemoglobin with 1 part O bound, $[HbO_2]$ is concentration of hemoglobin with 2 parts O bound, $[HbO_3]$ is the concentration of hemoglobin with 3 parts O, bound and $[HbO_4]$ is the concentration of hemoglobin with 4 parts O bound. The rate constants for the reactions depicted in Equations 7.6–7.11 are given in Table 7.1. It should be noted that the stochastic and determin-

istic forms of the standard formulation both use only two particle classes, products and reactants.

7.1.2 Standard Stochastic Formulation of Hemoglobin/O_2 as Outlined by Gibson

$$prop_1 = sk_1 \times Hb \times O \qquad Hb + O \xrightarrow{sk_1} HbO \tag{7.12}$$

$$prop_2 = sk_2 \times HbO \times O \qquad HbO + O \xrightarrow{sk_2} HbO_2 \tag{7.13}$$

$$prop_3 = sk_3 \times HbO_2 \times O \qquad HbO_2 + O \xrightarrow{sk_3} HbO_3 \tag{7.14}$$

$$prop_4 = sk_4 \times HbO_3 \times O \qquad HbO_3 + O \xrightarrow{sk_4} HbO_4 \tag{7.15}$$

$$prop_5 = k_{-1} \times HbO \qquad HbO \xrightarrow{k_{-1}} Hb + O \tag{7.16}$$

$$prop_6 = k_{-2} \times HbO_2 \qquad HbO_2 \xrightarrow{k_{-2}} HbO + O \tag{7.17}$$

$$prop_7 = k_{-3} \times HbO_3 \qquad HbO_3 \xrightarrow{k_{-3}} HbO_2 + O \tag{7.18}$$

$$prop_8 = k_{-4} \times HbO_4 \qquad HbO_4 \xrightarrow{k_{-4}} HbO_3 + O \tag{7.19}$$

where O is the number of oxygen, Hb is the number of deoxygenated hemoglobin, HbO is the number of hemoglobin with 1 part O bound, HbO_2 is number of hemoglobin with 2 parts O bound, HbO_3 is the number of hemoglobin with 3 parts O, bound and HbO_4 is the number of hemoglobin with 4 parts O bound. The stochastic rate

constants can be treated such that each micromole in our system is represented as a particle and the volume is assumed to be 1 liter. Therefore, the second order on-rates can be taken to be equal the original rates.

Table 7.4: A standardized formulation for Gibson's oxygenation of hemoglobin. The data elements are integers valued from zero to infinity.

Data Element	Molec. Id	Value Range	Number in State
Y[1]	Hb	$0, 1, 2 \ldots$	Fully Deoxygenated Hemoglobin
Y[2]	HbO	$0, 1, 2 \ldots$	Singly Oxygenated Hemoglobin
Y[3]	HbO_2	$0, 1, 2 \ldots$	Doubly Oxygenated Hemoglobin
Y[4]	HbO_3	$0, 1, 2 \ldots$	Triply Oxygenated Hemoglobin
Y[5]	HbO_4	$0, 1, 2 \ldots$	Fully Oxygenated Hemoglobin

7.1.3 Modified Scheme of Gibson's Hemoglobin/O_2 Binding Model

The state of the overall molecular complex of hemoglobin and oxygen throughout the simulation is tracked by evaluating the respective states of the individual subunits that make up the entire hemoglobin complex (Table 7.6). The individual one-to-one reactions represent a single biochemical event. However, in order to effectively model cooperativity, it is necessary to represent the reaction rates as functions of the states of the other non-reacting subunits. For example, the rate of oxygen binding to a subunit of hemoglobin is a function of the states of the other three subunits:

$$mod\_prop_1 = f(HemSub_2U, HemSub_3U, HemSub_4U) \times HemSub_1U \times \#Oxy. \tag{7.20}$$

The canonical algorithm for modeling cooperativity during oxygen uptake by hemoglobin is expressed in terms of steadily increasing rates of the stepwise addition of oxygen. It is also possible to add an additional layer of allosteric regulation stating that there can be two characteristic structural configurations for a hemoglobin complex and that the conformational switching is a function of the extent of oxygen uptake.

The data-model of the modified, three-particle formulation of Gibson's early Hb/O_2 model is shown in Table 7.8. We will now incorporate quantal effectors into our reaction schemes.

Table 7.5: The modified formulation for hemoglobin oxygen complexation states

$t = t_0$		$t = t_0 + \Delta t_1$		$t = t_0 + \Delta t_1 + \Delta t_2$	
$HemSub_1U$	**1**	**$HemSub_1U$**	**1**	$HemSub_1U$	0
$HemSub_2U$	**1**	**$HemSub_2U$**	**1**	**$HemSub_2U$**	**1**
$HemSub_3U$	**1**	$HemSub_3U$	0	$HemSub_3U$	0
$HemSub_4U$	**1**	**$HemSub_4U$**	**1**	**$HemSub_4U$**	**1**
$HemSub_1B$	0	$HemSub_1B$	0	**$HemSub_1B$**	**1**
$HemSub_2B$	0	$HemSub_2B$	0	$HemSub_2B$	0
$HemSub_3B$	0	**$HemSub_3B$**	**1**	**$HemSub_3B$**	**1**
$HemSub_4B$	0	$HemSub_4B$	0	$HemSub_4B$	0
Oxy	**4**	**Oxy**	**3**	**Oxy**	**2**

$t = t_0 + \Delta t_1 + \Delta t_2 + \Delta t_3$		$t = t_0 + \Delta t_1 + \Delta t_2 + \Delta t_3 + \Delta t_4$	
$HemSub_1U$	0	$HemSub_1U$	0
$HemSub_2U$	**1**	$HemSub_2U$	0
$HemSub_3U$	0	$HemSub_3U$	0
$HemSub_4U$	0	$HemSub_4U$	0
$HemSub_1B$	**1**	**$HemSub_1B$**	**1**
$HemSub_2B$	0	**$HemSub_2B$**	**1**
$HemSub_3B$	**1**	**$HemSub_3B$**	**1**
$HemSub_4B$	**1**	**$HemSub_4B$**	**1**
Oxy	**1**	**Oxy**	**0**

With the inclusion of quantal effectors in our reactions, we now have a three-

particle formulation that includes reactants, products, and quantal effectors. Furthermore, the three-particle formulation of computational chemical reactions networks will only be implemented in stochastic form.

$equation$	$reaction$	
$prop_1 = f(sk_{on}) \times Hb_1U \times O$	$Hb_1U + O \xrightarrow{f(\sum Bound)} Hb_1B$	(7.21)
$prop_2 = f(sk_{on}) \times Hb_2U \times O$	$Hb_2U + O \xrightarrow{f(\sum Bound)} Hb_2B$	(7.22)
$prop_3 = f(sk_{on}) \times Hb_3U \times O$	$Hb_3U + O \xrightarrow{f(\sum Bound)} Hb_3B$	(7.23)
$prop_4 = f(sk_{on}) \times Hb_4U \times O$	$Hb_4U + O \xrightarrow{f(\sum Bound)} Hb_4B$	(7.24)
$prop_5 = f(sk_{off}) \times HemSub_1B$	$Hb_1B \xrightarrow{f(\sum Bound)} Hb_1U + O$	(7.25)
$prop_6 = f(sk_{off}) \times HemSub_2B$	$Hb_2B \xrightarrow{f(\sum Bound)} Hb_2U + O$	(7.26)
$prop_7 = f(sk_{off}) \times HemSub_3B$	$Hb_3B \xrightarrow{f(\sum Bound)} Hb_3U + O$	(7.27)
$prop_8 = f(sk_{off}) \times HemSub_4B$	$Hb_4B \xrightarrow{f(\sum Bound)} Hb_4U + O$	(7.28)

Algorithms and Source Code 2: A snippet of code from Source Code **5**. This snippet focuses on the reaction rate function that implements Gibosn's findings. The volume of the box is calculated such that $1\mu M$ of hemoglobin equals 1 hemoglobin in the system.

```
if(y[1]+y[2]+y[3]+y[4]==4):
    ssakon=17.7/(nA*vol)*1e6*1/4
    ssakoff=0
elif(y[1]+y[2]+y[3]+y[4]==3):
    ssakon=33.2/(nA*vol)*1e6*1/3
```

```
    ssakoff=1900
elif(y[1]+y[2]+y[3]+y[4]==2):
    ssakon=4.89/(nA*vol)*1e6*1/2
    ssakoff=158/2
elif(y[1]+y[2]+y[3]+y[4]==1):
    ssakon=33.0/(nA*vol)*1e6
    ssakoff=539/3
else:
    ssakon=0
    ssakoff=50/4
```

Cooperativity effects in hemoglobin are expressed in terms of the sequential binding model, where the rate of complexation of the 2nd oxygen is faster than the 1st complexation step. In turn, the 3rd complexation is greater than the 2nd, and so forth. This trend is illustrated in Equation 7.29.

$$K_1 < K_2 < K_3 < K_4 \tag{7.29}$$

In total, there are 32 discrete chemical reactions that fully account for the complexation of oxygen by the subunits of hemoglobin.

The modified data-space that is used to simplify the overall problem is given in Table 7.5. This simplified data-space is constructed in such a way that each hemoglobin subunit can exist in only one state at a specific point in time in the simulation. The second column of Table 7.5 shows the state of each subunit at a specific time where 1 = formation and 0 = no formation of the Hb-O_2 complex. Other researchers (Imai et al. [1981]) have defined the data-space of a hemoglobin-oxygen complex as all possible combinations of the states of the individual subunits, in a

manner similar to that shown in Table 7.6.

7.1.4 Computational Results

Examples of Gibson's data and multiple fits to Gibson's data are shown in Figure 7.2.4 . The deterministic two-particle formulation of Gibson's model is plotted as a blue line. The stochastic two-particle formulation is plotted in green, while the modified, stochastic three-particle formulation is plotted in cyan. Finally, a digitized example of Gibson's data is depicted in red.

We were able to model the kinetics of hemoglobin's complexation with O_2.

7.2 Thermodynamic Model

Ackers' thermodynamic model for O_2 binding to Hb can be converted to a kinetic model by employing the "principle of microscopic reversibility," as expressed in Equation 5.2. In the event that we only have knowledge of the successive equilibrium binding constants, we can invoke a suitable reaction mechanism to produce thermodynamically accurate results by choosing k_{off} and k_{on} values that satisfy "microscopic reversibility". A full-state model for the complexation of O_2 with Hb is given in Equations 7.30 – 7.61, where we make the fewest simplifying assumptions.

The stepwise and reversible complexation of oxygen by hemoglobin occurs with an apparent increasing affinity as additional oxygen molecules are bound. The four binding sites are represented by either U for unbound or B for bound to

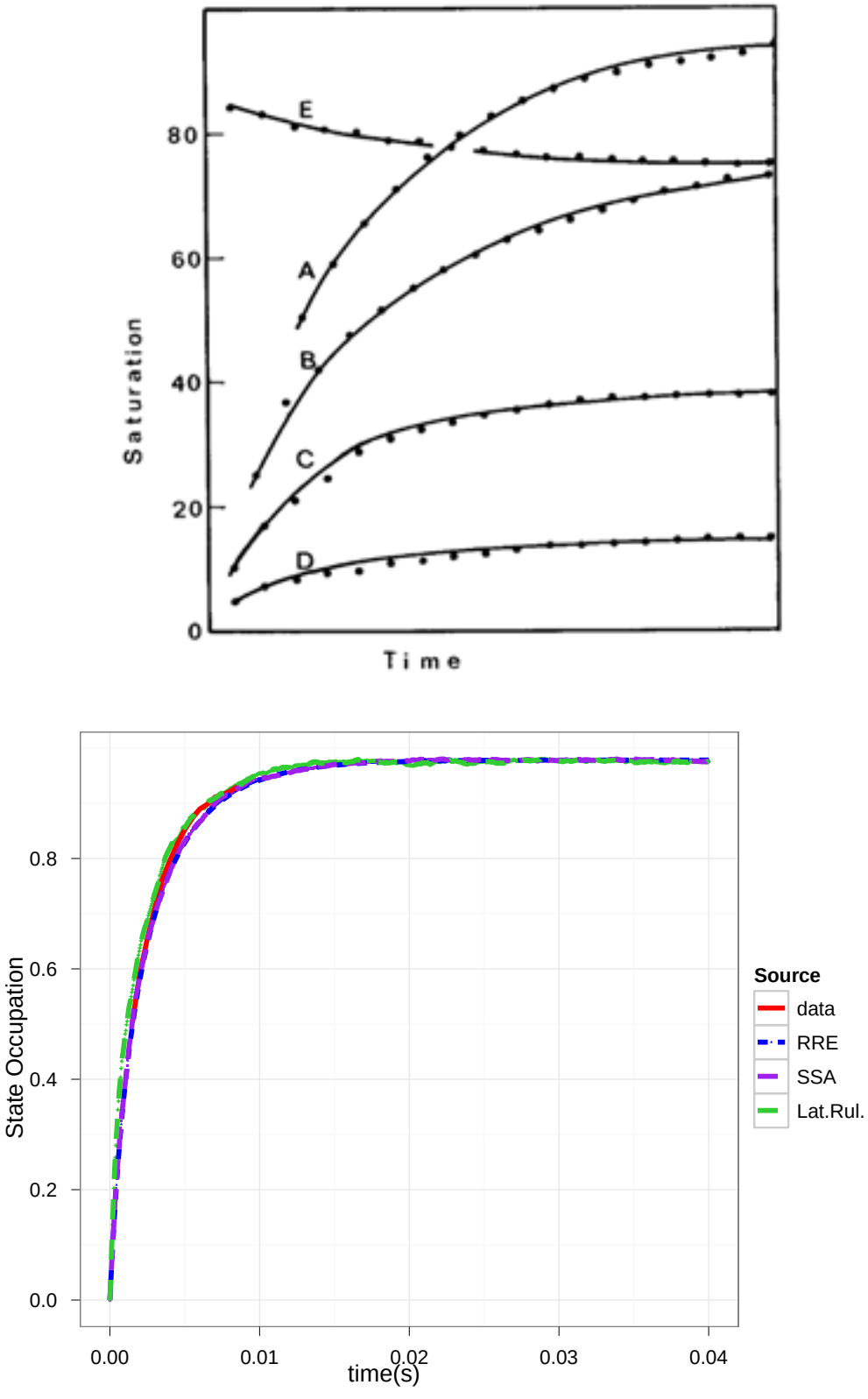


Figure 7.2: Various fits to data obtained from Gibson [1970]

oxygen (e.g., Hem_UUUU is the fully unbound form of hemoglobin. Hem_UBBB and Hem_BUBB are two examples of a hemoglobin complex with a single oxygen molecule in the bound state). In Table 7.6, the four subunits are depicted as they bind to oxygen.

$$\text{Hem\_UUUU} + \text{O}_2 \overset{1K}{\rightleftharpoons} \text{Hem\_BUUU} \tag{7.30}$$

$$\text{Hem\_UUUU} + \text{O}_2 \overset{1K}{\rightleftharpoons} \text{Hem\_UBUU} \tag{7.31}$$

$$\text{Hem\_UUUU} + \text{O}_2 \overset{1K}{\rightleftharpoons} \text{Hem\_UUBU} \tag{7.32}$$

$$\text{Hem\_UUUU} + \text{O}_2 \overset{1K}{\rightleftharpoons} \text{Hem\_UUUB} \tag{7.33}$$

$$\text{Hem\_BUUU} + \text{O}_2 \overset{9K}{\rightleftharpoons} \text{Hem\_BBUU} \tag{7.34}$$

$$\text{Hem\_BUUU} + \text{O}_2 \overset{1K}{\rightleftharpoons} \text{Hem\_BUBU} \tag{7.35}$$

$$\text{Hem\_BUUU} + \text{O}_2 \overset{1K}{\rightleftharpoons} \text{Hem\_BUUB} \tag{7.36}$$

$$\text{Hem\_UBUU} + \text{O}_2 \overset{9K}{\rightleftharpoons} \text{Hem\_BBUU} \tag{7.37}$$

$$\text{Hem\_UBUU} + \text{O}_2 \overset{1K}{\rightleftharpoons} \text{Hem\_UBBU} \tag{7.38}$$

$$\text{Hem\_UBUU} + \text{O}_2 \overset{1K}{\rightleftharpoons} \text{Hem\_UBUB} \tag{7.39}$$

$$\text{Hem\_UUBU} + \text{O}_2 \overset{1K}{\rightleftharpoons} \text{Hem\_BUBU} \tag{7.40}$$

$$\text{Hem\_UUBU} + \text{O}_2 \overset{1K}{\rightleftharpoons} \text{Hem\_UBBU} \tag{7.41}$$

$$\text{Hem\_UUBU} + \text{O}_2 \overset{9K}{\rightleftharpoons} \text{Hem\_UUBB} \tag{7.42}$$

$$\text{Hem\_UUUB} + \text{O}_2 \overset{1K}{\rightleftharpoons} \text{Hem\_BUUB} \tag{7.43}$$

$$\text{Hem\_UUUB} + \text{O}_2 \overset{1K}{\rightleftharpoons} \text{Hem\_UBUB} \tag{7.44}$$

$$\text{Hem\_UUUB} + \text{O}_2 \overset{9K}{\rightleftharpoons} \text{Hem\_UUBB} \tag{7.45}$$

$$\text{Hem\_BBUU} + \text{O}_2 \overset{9K}{\rightleftharpoons} \text{Hem\_BBBU} \tag{7.46}$$

$$\text{Hem\_BBUU} + \text{O}_2 \overset{9K}{\rightleftharpoons} \text{Hem\_BBUB} \tag{7.47}$$

$$\text{Hem\_BUBU} + \text{O}_2 \overset{90K}{\rightleftharpoons} \text{Hem\_BBBU} \tag{7.48}$$

$$\text{Hem\_BUBU} + \text{O}_2 \overset{90K}{\rightleftharpoons} \text{Hem\_BUBB} \tag{7.49}$$

$$\text{Hem\_BUUB} + \text{O}_2 \overset{90K}{\rightleftharpoons} \text{Hem\_BBUB} \tag{7.50}$$

$$\text{Hem\_BUUB} + \text{O}_2 \overset{90K}{\rightleftharpoons} \text{Hem\_BUBB} \tag{7.51}$$

$$\text{Hem\_UBBU} + \text{O}_2 \overset{90K}{\rightleftharpoons} \text{Hem\_BBBU} \tag{7.52}$$

$$\text{Hem\_UBBU} + \text{O}_2 \overset{90K}{\rightleftharpoons} \text{Hem\_UBBB} \tag{7.53}$$

$$\text{Hem\_UBUB} + \text{O}_2 \overset{90K}{\rightleftharpoons} \text{Hem\_BBUB} \tag{7.54}$$

$$\text{Hem\_UBUB} + \text{O}_2 \overset{90K}{\rightleftharpoons} \text{Hem\_UBBB} \tag{7.55}$$

$$\text{Hem\_UUBB} + \text{O}_2 \overset{9K}{\rightleftharpoons} \text{Hem\_BUBB} \tag{7.56}$$

$$\text{Hem\_UUBB} + \text{O}_2 \overset{9K}{\rightleftharpoons} \text{Hem\_UBBB} \tag{7.57}$$

$$\text{Hem\_BBBU} + \text{O}_2 \overset{550K}{\rightleftharpoons} \text{Hem\_BBBB} \tag{7.58}$$

$$\text{Hem\_BBUB} + \text{O}_2 \overset{550K}{\rightleftharpoons} \text{Hem\_BBBB} \tag{7.59}$$

$$\text{Hem\_BUBB} + \text{O}_2 \overset{550K}{\rightleftharpoons} \text{Hem\_BBBB} \tag{7.60}$$

$$\text{Hem\_UBBB} + \text{O}_2 \overset{550K}{\rightleftharpoons} \text{Hem\_BBBB} \tag{7.61}$$

7.2.1 Standard Deterministic Formulation of Ackers' Model of Hemoglobin/O_2

In standard deterministic form, 16 reaction rate equations are required for this model.

$$\frac{dA}{dt} = -k_1[A][O] + k_{-1}[B] - k_2[A][O] + k_{-2}[C] \tag{7.62}$$

$$\frac{dB}{dt} = k_1[A][O] - k_{-1}[B] - k_3[B][O] + k_{-3}[E] - k_4[B][O] + k_{-4}[F] - k_5[B][O] + k_{-5}[D] \tag{7.63}$$

$$\frac{dC}{dt} = k_2[A][O] - k_{-2}[C] - k_6[C][O] + k_{-6}[D] - k_7[C][O] + k_{-7}[F] - k_8[C][O] + k_{-8}[G] \tag{7.64}$$

$$\frac{dD}{dt} = k_5[B][O] - k_{-5}[D] + k_6[C][O] - k_{-6}[D] - k_9[D][O] + k_{-9}[H] - k_{10}[D][O] + k_{-10}[I] \tag{7.65}$$

$$\frac{dE}{dt} = k_3[B][O] - k_{-3}[E] - k_{11}[E][O] + k_{-11}[H] \tag{7.66}$$

$$\frac{dF}{dt} = k_4[B][O] - k_{-4}[F] + k_7[C][O] - k_{-7}[F] - k_{12}[F][O] + k_{-12}[H] - k_{13}[F][O] + k_{-13}[I] \tag{7.67}$$

$$\frac{dG}{dt} = k_8[C][O] - k_{-8}[G] - k_{14}[G][O] + k_{-14}[I] \tag{7.68}$$

$$\frac{dH}{dt} = k_{11}[E][O] - k_{-11}[H] + k_{12}[F][O] - k{-}12[H] - k_{15}[H][O] + k_{-15}[J] \tag{7.69}$$

$$\frac{dI}{dt} = k_{10}[D][O] - k_{-10}[I] + k_{13}[F][O] - k_{-13}[I]$$

$$- k_{16}[I][O] + k_{-16}[J] \tag{7.70}$$

$$\frac{dJ}{dt} = k_{15}[H][O] - k_{-15}[J] + k_{16}[I][O] - k_{-16}[J] \tag{7.71}$$

$$\begin{aligned}\frac{dO}{dt} =& - k_1[A][O] + k_{-1}[B] - k_2[C][O] + k_{-2}[C] \\ & - k_3[B][O] + k_{-3}[E] - k_4[B][O] + k_{-4}[F] \\ & - k_5[B][O] + k_{-5}[D] - k_6[C][O] + k_{-6}[D] \\ & - k_7[C][O] + k_{-7}[F] - k_8[C][O] + k_{-8}[G] \\ & - k_9[D][O] + k_{-9}[H] - k_{10}[D][O] + k_{-10}[I] \\ & - k_{11}[E][O] + k_{-11}[H] - k_{12}[F][O] + k_{-12}[H] \\ & - k_{13}[F][O] + k_{13}[I] - k_{14}[G][O] + k_{-14}[I] \\ & - k_{15}[H][O] + k_{-15}[J] - k_{16}[I][O] + k_{-16}[J]\end{aligned} \tag{7.72}$$

$[A]$ is equal to the concentration of fully deoxygenated Hb, $[B]$ is equal to the concentration of singly bound Hb with an oxygen on the α terminus, $[C]$ is equal to the concentration of singly bound Hb with an oxygen on the β terminus. For the doubly bound ligands, $[D]$ is equal to to the concentration of Hb with an oxygen bound to the α and β termini of the same hemoglobin dimer (recall that Hb is a dimer of dimers). $[E]$ is equal to the concentration of Hb with oxygen bound to the α termini on different dimers, $[F]$ is equal to the concentration of Hb with oxygen bound to the α and β termini of opposing dimers, $[G]$ to the concentration of Hb with oxygen being present on each β subunit. $[H]$ is a triply bound Hb with

an oxygen missing from the β subunit. $[I]$ is a triply bound Hb with an oxygen missing from the α subunit. $[J]$ is the fully oxygenated Hb macromolecule.

7.2.2 Standard Stochastic Formulation of Ackers' Model of Hemoglobin/O_2

Equation	Reaction	
$prop_1 = sk_1 \times A \times O$	$A + O \xrightarrow{sk_1} B$	(7.73)
$prop_2 = sk_2 \times A \times O$	$A + O \xrightarrow{sk_2} C$	(7.74)
$prop_3 = sk_3 \times B \times O$	$B + O \xrightarrow{sk_3} D$	(7.75)
$prop_4 = sk_4 \times B \times O$	$B + O \xrightarrow{sk_4} E$	(7.76)
$prop_5 = sk_5 \times B \times O$	$B + O \xrightarrow{sk_5} F$	(7.77)
$prop_6 = sk_6 \times C \times O$	$C + O \xrightarrow{sk_6} D$	(7.78)
$prop_7 = sk_7 \times C \times O$	$C + O \xrightarrow{sk_7} F$	(7.79)
$prop_8 = sk_8 \times C \times O$	$C + O \xrightarrow{sk_8} G$	(7.80)
$prop_9 = sk_9 \times D \times O$	$D + O \xrightarrow{sk_9} H$	(7.81)
$prop_{10} = sk_{10} \times D \times O$	$D + O \xrightarrow{sk_{10}} I$	(7.82)
$prop_{11} = sk_{11} \times E \times O$	$E + O \xrightarrow{sk_{11}} H$	(7.83)

$$prop_{12} = sk_{12} \times F \times O \qquad F + O \xrightarrow{sk_{12}} H \tag{7.84}$$

$$prop_{13} = sk_{13} \times F \times O \qquad F + O \xrightarrow{sk_{13}} I \tag{7.85}$$

$$prop_{14} = sk_{14} \times G \times O \qquad G + O \xrightarrow{sk_{14}} I \tag{7.86}$$

$$prop_{15} = sk_{15} \times H \times O \qquad H + O \xrightarrow{sk_{15}} J \tag{7.87}$$

$$prop_{16} = sk_{16} \times I \times O \qquad I + O \xrightarrow{sk_{16}} J \tag{7.88}$$

$$prop_{17} = k_{-1} \times B \qquad B \xrightarrow{k_{-1}} A + O \tag{7.89}$$

$$prop_{18} = k_{-2} \times C \qquad C \xrightarrow{k_{-2}} A + O \tag{7.90}$$

$$prop_{19} = k_{-3} \times D \qquad D \xrightarrow{k_{-3}} B + O \tag{7.91}$$

$$prop_{20} = k_{-4} \times E \qquad E \xrightarrow{k_{-4}} B + O \tag{7.92}$$

$$prop_{21} = k_{-5} \times F \qquad F \xrightarrow{k_{-5}} B + O \tag{7.93}$$

$$prop_{22} = k_{-6} \times D \qquad D \xrightarrow{k_{-6}} C + O \tag{7.94}$$

$$prop_{23} = k_{-7} \times F \qquad F \xrightarrow{k_{-7}} C + O \tag{7.95}$$

$$prop_{24} = k_{-8} \times G \qquad G \xrightarrow{k_{-8}} C + O \tag{7.96}$$

$$prop_{25} = k_{-9} \times H \qquad H \xrightarrow{k_{-9}} D + O \tag{7.97}$$

$$prop_{26} = k_{-10} \times I \qquad I \xrightarrow{k_{-10}} D + O \tag{7.98}$$

$$prop_{27} = k_{-11} \times H \qquad H \xrightarrow{k_{-11}} E + O \tag{7.99}$$

$$prop_{28} = k_{-12} \times H \qquad H \xrightarrow{k_{-12}} F + O \tag{7.100}$$

$$prop_{29} = k_{-13} \times I \qquad I \xrightarrow{k_{-13}} F + O \tag{7.101}$$

$$prop_{30} = k_{-14} \times I \qquad I \xrightarrow{k_{-14}} G + O \tag{7.102}$$

$$prop_{31} = k_{-15} \times J \qquad J \xrightarrow{k_{-15}} H + O \tag{7.103}$$

$$prop_{32} = k_{-16} \times J \qquad J \xrightarrow{k_{-16}} I + O \tag{7.104}$$

A is equal to the number of fully deoxygenated Hb, B is equal to the number of singly bound Hb with an oxygen on the α terminus, C is equal to the number of singly bound Hb with an oxygen on the β terminus. For the doubly bound ligands, D is equal to to the number of Hb with an oxygen bound to the α and β termini of the same hemoglobin dimer (recall that Hb is a dimer of dimers). E is equal to the number of Hb with oxygen bound to the α termini on different dimers, F is equal to the number of Hb with oxygen bound to the α and β termini of opposing dimers, G to the number of Hb with oxygen being present on each β subunit. H is equal to the number of triply bound Hb with an oxygen missing from the β subunit. I is equal to the number of triply bound Hb with an oxygen missing from the α subunit. J is the number of fully oxygenated Hb macromolecule.

7.2.3 Modified Stochastic Formulation of Ackers' Model of Hemoglobin/O_2

The full state model with 32 bidirectional reactions (16 when symmetry is considered) can be reduced to a modified form with 4 bidirectional reactions with

Table 7.6: The standard formulation for hemoglobin oxygen complexation states

$t = t_0$		$t = t_0 + \Delta t_1$		$t = t_0 + \Delta t_1 + \Delta t_2$	
Hem_UUUU	**1**	Hem_UUUU	0	Hem_UUUU	0
Hem_BUUU	0	Hem_BUUU	0	Hem_BUUU	0
Hem_UBUU	0	Hem_UBUU	0	Hem_UBUU	0
Hem_UUBU	0	**Hem_UUBU**	**1**	Hem_UUBU	0
Hem_UUUB	0	Hem_UUUB	0	Hem_UUUB	0
Hem_BBUU	0	Hem_BBUU	0	Hem_BBUU	0
Hem_BUBU	0	Hem_BUBU	0	**Hem_BUBU**	**1**
Hem_BUUB	0	Hem_BUUB	0	Hem_BUUB	0
Hem_UBBU	0	Hem_UBBU	0	Hem_UBBU	0
Hem_UBUB	0	Hem_UBUB	0	Hem_UBUB	0
Hem_UUBB	0	Hem_UUBB	0	Hem_UUBB	0
Hem_BBBU	0	Hem_BBBU	0	Hem_BBBU	0
Hem_BBUB	0	Hem_BBUB	0	Hem_BBUB	0
Hem_BUBB	0	Hem_BUBB	0	Hem_BUBB	0
Hem_UBBB	0	Hem_UBBB	0	Hem_UBBB	0
Hem_BBBB	0	Hem_BBBB	0	Hem_BBBB	0
Oxy	**4**	**Oxy**	**3**	**Oxy**	**2**

$t = t_0 + \Delta t_1 + \Delta t_2 + \Delta t_3$		$t = t_0 + \Delta t_1 + \Delta t_2 + \Delta t_3 + \Delta t_4$	
Hem_UUUU	0	Hem_UUUU	0
Hem_BUUU	0	Hem_BUUU	0
Hem_UBUU	0	Hem_UBUU	0
Hem_UUBU	0	Hem_UUBU	0
Hem_UUUB	0	Hem_UUUB	0
Hem_BBUU	0	Hem_BBUU	0
Hem_BUBU	0	Hem_BUBU	1
Hem_BUUB	0	Hem_BUUB	0
Hem_UBBU	0	Hem_UBBU	0
Hem_UBUB	0	Hem_UBUB	0
Hem_UUBB	0	Hem_UUBB	0
Hem_BBBU	0	Hem_BBBU	0
Hem_BBUB	0	Hem_BBUB	0
Hem_BUBB	**1**	Hem_BUBB	0
Hem_UBBB	0	Hem_UBBB	0
Hem_BBBB	0	**Hem_BBBB**	**1**
Oxy	**1**	**Oxy**	**0**

dynamic reaction rates (Equations 7.105–7.108) by simplifying in terms of symmetry. A representation of the modified formulation of the problem is shown in the following equations:

$$\mathrm{Hb}_{\alpha 1} + \mathrm{O}_2 \xrightleftharpoons{\mathrm{f(Hb}_{\beta 1}\mathrm{,Hb}_{\alpha 2}\mathrm{,Hb}_{\beta 2}\mathrm{)}} \mathrm{Hb}_{\alpha 1}\mathrm{O}_2 \tag{7.105}$$

$$\mathrm{Hb}_{\beta 1} + \mathrm{O}_2 \xrightleftharpoons{\mathrm{f(Hb}_{\alpha 1}\mathrm{,Hb}_{\alpha 2}\mathrm{,Hb}_{\beta 2}\mathrm{)}} \mathrm{Hb}_{\beta 1}\mathrm{O}_2 \tag{7.106}$$

$$\mathrm{Hb}_{\alpha 2} + \mathrm{O}_2 \xrightleftharpoons{\mathrm{f(Hb}_{\alpha 1}\mathrm{,Hb}_{\beta 1}\mathrm{,Hb}_{\beta 2}\mathrm{)}} \mathrm{Hb}_{\alpha 2}\mathrm{O}_2 \tag{7.107}$$

$$\mathrm{Hb}_{\beta 2} + \mathrm{O}_2 \xrightleftharpoons{\mathrm{f(Hb}_{\alpha 1}\mathrm{,Hb}_{\alpha 2}\mathrm{,Hb}_{\beta 1}\mathrm{)}} \mathrm{Hb}_{\beta 2}\mathrm{O}_2 \tag{7.108}$$

where $Hb_{\alpha 1}$ is the alpha subunit on the first dimer, $Hb_{\alpha 2}$ is the alpha subunit on the second dimer, $Hb_{\beta 1}$ is the beta subunit on the first dimer, $Hb_{\beta 2}$ is the beta subunit on the second dimer, and O_2 is dioxygen. When there is no space and "+" between the hemoglobin subunit and dioxygen that means that they are complexed. The "f(*)" in the rate means that this rate is a function of the parameters provided in place of "*".

Here we provide the modified Gillespie equations on the left along with the

specific reaction trajectory that they represent on the right.

equation	reaction	
$prop_1 = fkon_1 \times HemSub_1U \times O$	$Hb_{\alpha 1} + O \longrightarrow Hb_{\alpha 1}O$	(7.109)
$prop_2 = fkon_2 \times HemSub_2U \times O$	$Hb_{\beta 1} + O \longrightarrow Hb_{\beta 1}O$	(7.110)
$prop_3 = fkon_3 \times HemSub_3U \times O$	$Hb_{\alpha 2} + O \longrightarrow Hb_{\alpha 2}O$	(7.111)
$prop_4 = fkon_4 \times HemSub_4U \times O$	$Hb_{\beta 2} + O \longrightarrow Hb_{\beta 2}O$	(7.112)
$prop_5 = koff1 \times HemSub_1B$	$Hb_{\alpha 1}O \longrightarrow Hb_{\alpha 1} + O$	(7.113)
$prop_6 = koff2 \times HemSub_2B$	$Hb_{\beta 1}O \longrightarrow Hb_{\beta 1} + O$	(7.114)
$prop_7 = koff3 \times HemSub_3B$	$Hb_{\alpha 2}O \longrightarrow Hb_{\alpha 2} + O$	(7.115)
$prop_8 = koff4 \times HemSub_4B$	$Hb_{\beta 2}O \longrightarrow Hb_{\beta 2} + O$	(7.116)

Algorithms and Source Code 3: A snippet of code from Source Code **8**. The following code implements Acker's cooperative scheme of Hb/O_2^{2+} complexation.

```
if sum(y[1:5])==4:
    """Ackers Hb ~ 10"""
    kon1=kon
    kon2=kon
    kon3=kon
    kon4=kon
    koff1 = koff
    koff2 = koff
    koff3 = koff
    koff4 = koff
elif sum(y[1:5])==3:
    if y[5]==1:
        """ one alpha is bound to an oxygen """
        kon1 = 0
        kon2 = mult1*kon
        kon3 = kon
        kon4 = kon
        koff1= koff
        koff2 = 0
        koff3 = 0
```

```
        koff4 = 0
    if y[6]==1:
        """ one beta is bound to an oxygen """
        kon1 = mult1*kon
        kon2 = 0
        kon3 = kon
        kon4 = kon
        koff1 = koff
        koff2 = koff
        koff3 = koff
        koff4 = koff
    if y[7]==1:
        """ one alpha is bound to an oxygen """
        kon1 = kon
        kon2 = kon
        kon3 = 0
        kon4 = mult1*kon
        koff1 = koff
        koff2 = koff
        koff3 = koff
        koff4 = koff
    if y[8]==1:
        """ one beta is bound to an oxygen """
        kon1 = kon
        kon2 = kon
        kon3 = mult1*kon
        kon4 = 0
        koff1 = koff
        koff2 = koff
        koff3 = koff
        koff4 = koff
elif sum(y[1:5])==2:
    if y[1]==1 and y[2]==1:
        """  """
        kon1 = mult1*kon
        kon2 = mult1*kon
        kon3 = mult1*kon
        kon4 = mult1*kon
        koff1 = koff
        koff2 = koff
        koff3 = koff
        koff4 = koff
    if y[1]==1 and y[3]==1:
        """  """
        kon1 = mult2*kon
        kon2 = mult2*kon
        kon3 = mult2*kon
```

```
            kon4 = mult2*kon
            koff1 = koff
            koff2 = koff
            koff3 = koff
            koff4 = koff
        if y[1]==1 and y[4]==1:
            """  """
            kon1 = mult2*kon
            kon2 = mult2*kon
            kon3 = mult2*kon
            kon4 = mult2*kon
            koff1 = koff
            koff2 = koff
            koff3 = koff
            koff4 = koff
        if y[2]==1 and y[3]==1:
            """  """
            kon1 = mult2*kon
            kon2 = mult2*kon
            kon3 = mult2*kon
            kon4 = mult2*kon
            koff1 = koff
            koff2 = koff
            koff3 = koff
            koff4 = koff
        if y[2]==1 and y[4]==1:
            """  """
            kon1 = mult2*kon
            kon2 = mult2*kon
            kon3 = mult2*kon
            kon4 = mult2*kon
            koff1 = koff
            koff2 = koff
            koff3 = koff
            koff4 = koff
        if y[3]==1 and y[4]==1:
            """  """
            kon1 = mult1*kon
            kon2 = mult1*kon
            kon3 = mult1*kon
            kon4 = mult1*kon
            koff1 = koff
            koff2 = koff
            koff3 = koff
            koff4 = koff
    elif sum(y[1:5])==1:
        if y[1]==1:
```

```
        """  """
        kon1 = mult3*kon
        kon2 = 0
        kon3 = 0
        kon4 = 0
        koff1 = koff
        koff2 = koff
        koff3 = koff
        koff4 = koff
    if y[2]==1:
        """  """
        kon1 = 0
        kon2 = mult3*kon
        kon3 = 0
        kon4 = 0
        koff1 = koff
        koff2 = koff
        koff3 = koff
        koff4 = koff
    if y[3]==1:
        """  """
        kon1 = 0
        kon2 = 0
        kon3 = mult3*kon
        kon4 = 0
        koff1 = koff
        koff2 = koff
        koff3 = koff
        koff4 = koff
    if y[4]==1:
        """  """
        kon1 = 0
        kon2 = 0
        kon3 = 0
        kon4 = mult3*kon
        koff1 = koff
        koff2 = koff
        koff3 = koff
        koff4 = koff
```

7.2.4 Computational Size or Summary of Our Models

In the standard representation, the largest number of data elements required to represent the number of states of a macromolecule is a^n. The value of a represents the number of different states that exist for an individual subunit—e.g., for hemoglobin, a equals two, corresponding to bound and unbound. The number of subunits equals n. The largest number of reactions that are needed to represent the dynamic activity of a macromolecule is given by

$$r \times n \times a^{(n-1)} \tag{7.117}$$

where r equals the number of reactions a subunit can undergo, e.g., for hemoglobin/O_2, $r = 2$.

In our modified formulation, a single macromolecule requires $a \times n$ Boolean data elements (1 byte for true or false), and $r \times n$ data elements are required to account for the postulated reaction mechanism. For example, the computational complexity for the oxygen + hemoglobin system is compared in Table 7.7 to a system that the author has used to represent the cooperative binding of calcium (Ca^{++}) with calmodulin (CaM) and subsequently with calcium calmodulin kinase II (CaMKII), so that r= 104, a = 20, and n = 12. The latter case has proven to be computationally intractable.

In order to address the issue of complexity first, a simple kinetic model of sequential binding of O_2 by Hb, which is constrained by the data of Gibson, was

	Hemoglobin	Ca^{++}/CaM/CaMKII
a	2	20
n	4	12
r	1/2	52/104
a^n	16	4.10×10^{15}
$r \times n \times a^{(n-1)}$	32/64	$1.27 \times 10^{17}/2.55 \times 10^{17}$
$a \times n$	8	240
$r \times n$	4/8	624/1248

Table 7.7: This table provides a numerical comparison of the relative complexity in the problem formulations for two protein systems exhibiting kinetic and thermodynamic cooperativity. The numbers separated by a slash (/) are for either the bidirectional or unidirectional case, in that order.

developed. Four reactions are used in the standard formulation, while in the modified formulation, eight separate reactions (i.e., four reversible reactions) were represented (Table 7.2).

7.2.5 Computational Results

The Nedler-Mead simplex method is used to fit the reaction mechanism proposed by Ackers and Holt [2006] to a subset of the data of Ackers et. al. [1976], as depicted in Figure 7.4. The data representation of the modified reaction scheme is depicted in Table 7.9. The data from the deterministic and stochastic fits are shown in Figure 7.5. Initial equilibrium ratios, K, for both the deterministic simulations were chosen such that the starting prediction loosely approximated the observed Hb/O_2 binding curves. Stochastic simulations had stochastic equilibrium ratios that were consistent with deterministic results (0–20% greater). The results, which are also summarized in Table 7.10, show the cooperative relationship proposed by Ackers is not constant. The reason for this is also provided by Ackers, in that

Table 7.8: A modified formulation of the reaction scheme of Hb/O_2 by Gibson

	Reactants	Products	Quantal Effectors	Equation
1	$\mathrm{HbSub}_1, \mathrm{O}_2$	$\mathrm{HbSub}_1\mathrm{O}_2$	$\mathrm{HbSub}_2, \mathrm{HbSub}_3, \mathrm{HbSub}_4, +/-\mathrm{O}_2$	7.105-7.108
2	$\mathrm{HbSub}_2, \mathrm{O}_2$	$\mathrm{HbSub}_2\mathrm{O}_2$	$\mathrm{HbSub}_1, \mathrm{HbSub}_3, \mathrm{HbSub}_4, +/-\mathrm{O}_2$	7.105-7.108
3	$\mathrm{HbSub}_3, \mathrm{O}_2$	$\mathrm{HbSub}_3\mathrm{O}_2$	$\mathrm{HbSub}_1, \mathrm{HbSub}_2, \mathrm{HbSub}_4, +/-\mathrm{O}_2$	7.105-7.108
4	$\mathrm{HbSub}_4, \mathrm{O}_2$	$\mathrm{HbSub}_4\mathrm{O}_2$	$\mathrm{HbSub}_1, \mathrm{HbSub}_2, \mathrm{HbSub}_3, +/-\mathrm{O}_2$	7.105-7.108

Table 7.9: A modified formulation of Ackers' Hb/O_2

	Reactants	Products	Quantal Effecters	Equation
1	$\mathrm{Hb}_{\alpha 1}, \mathrm{O}_2$	$\mathrm{Hb}_{\alpha 1}\mathrm{O}_2$	$\mathrm{Hb}_{\beta 1}, \mathrm{Hb}_{\alpha 2}, \mathrm{Hb}_{\beta 2}, +/-\mathrm{O}_2$	7.30, 7.37, 7.40, 7.43, 7.52, 7.54, 7.56, 7.61
2	$\mathrm{Hb}_{\beta 1}, \mathrm{O}_2$	$\mathrm{Hb}_{\beta 1}\mathrm{O}_2$	$\mathrm{Hb}_{\alpha 1}, \mathrm{Hb}_{\alpha 2}, \mathrm{Hb}_{\beta 2}, +/-\mathrm{O}_2$	7.31, 7.34, 7.41, 7.44, 7.48, 7.50, 7.57, 7.60
3	$\mathrm{Hb}_{\alpha 2}, \mathrm{O}_2$	$\mathrm{Hb}_{\alpha 2}\mathrm{O}_2$	$\mathrm{Hb}_{\alpha 1}, \mathrm{Hb}_{\beta 1}, \mathrm{Hb}_{\beta 2}, +/-\mathrm{O}_2$	7.32, 7.35, 7.38, 7.45, 7.46, 7.51, 7.55, 7.59
4	$\mathrm{Hb}_{\beta 2}, \mathrm{O}_2$	$\mathrm{Hb}_{\beta 2}\mathrm{O}_2$	$\mathrm{Hb}_{\alpha 1}, \mathrm{Hb}_{\beta 1}, \mathrm{Hb}_{\alpha 2}, +/-\mathrm{O}_2$	7.33, 7.36, 7.39, 7.42, 7.47, 7.49, 7.53, 7.58

he stated as early as 1976 that hemoglobin is actually a dimer of dimers and that at low concentration it exhibits a more dimer-like behavior than tetrameric-like, prototypical, character.

While the kinetic results of the kinetic model of Gibson (see Figure 7.2.4) show consistency between the various models, the results of the thermodynamic system of Ackers (see Figure 7.5) show a strong divergence in behavior. The divergence seems to occur when dioxygen is limiting with respect to number rather than concentration. This is discussed further in the next section and the following chapter.

Table 7.10: Deterministic fits of the mechanism of Ackers and Holt [2005] to the data of Mills et al. [1976]

	40nM	0.27μM	5.4μM	38μM
$K(M^{-1})$	1.846 $\times 10^5$	5.90 $\times 10^4$	8.97 $\times 10^3$	7.35 $\times 10^3$
$Mult_1$	0.8587	0.8795	20.525	9.217
$Mult_2$	0.6218	0.6002	54.732	80.860
$Mult_3$	11.9955	64.0806	83.635	518.5384

7.3 Discussion

At high concentrations of hemoglobin, in small volumes for single-lattice models of hemoglobin, both the standard and modified stochastic formulations deviate from the experimental data. This seems to be a quantal effect that is apparently a non-ergodic in nature. It is important to note that the experiments were performed at higher volumes and therefore it is possible that they are unfit to be considered as an analogy for what would happen when observing a single molecule of "hemoglobin" in a box.

It is possible that the critical threshold for quantal behavior seems to be the point from experimental data that occurs when the volume is decreased to the point that there are fewer ligand molecules than there are cooperative subunits (four O_2 for hemoglobin). This is not an artifact, but the point at which the system shows a quantization of states that is apparently non-ergodic. That the deterministic treatment cannot accurately describe the quantal behavior of the system is reminiscent of the particle-in-a-box model that has been used to describe matter's wave-packet duality.

The stochastic model fits the apparent non-ergodicity, but the deterministic fails

to fit it. The system becomes apparently non-ergodic at low volumes because cooperativity cannot be fully realized when there are more cooperative subunits than available ligands. We must break down this model further to determine what is happening.

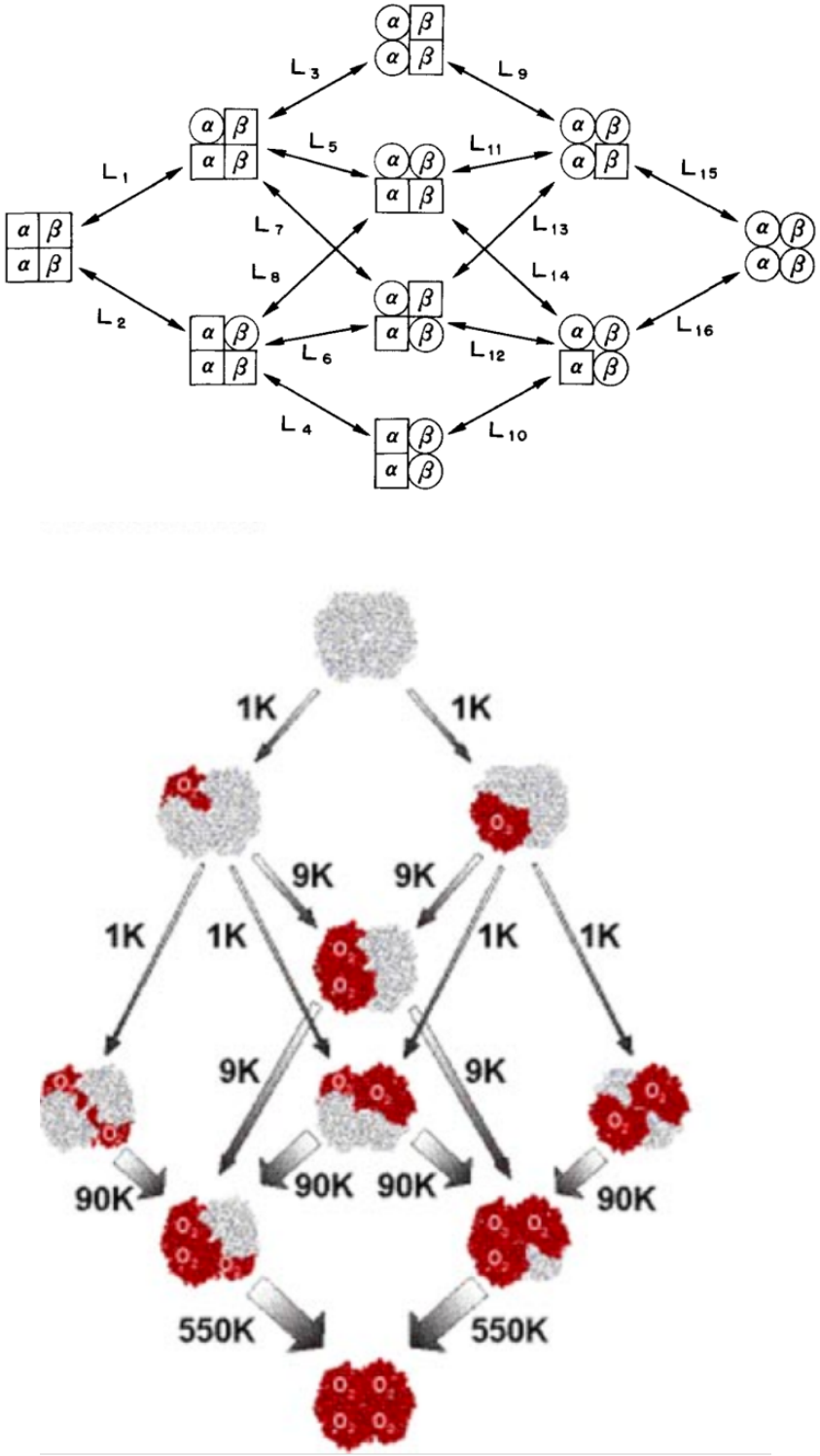


Figure 7.3: A) A schematic diagram of the possible modes of interaction of hemoglobin with oxygen, according to the model of Adair and interpreted by Imai [1981]. B) Ackers' detailed model of hemoglobin thermodynamics [2006].

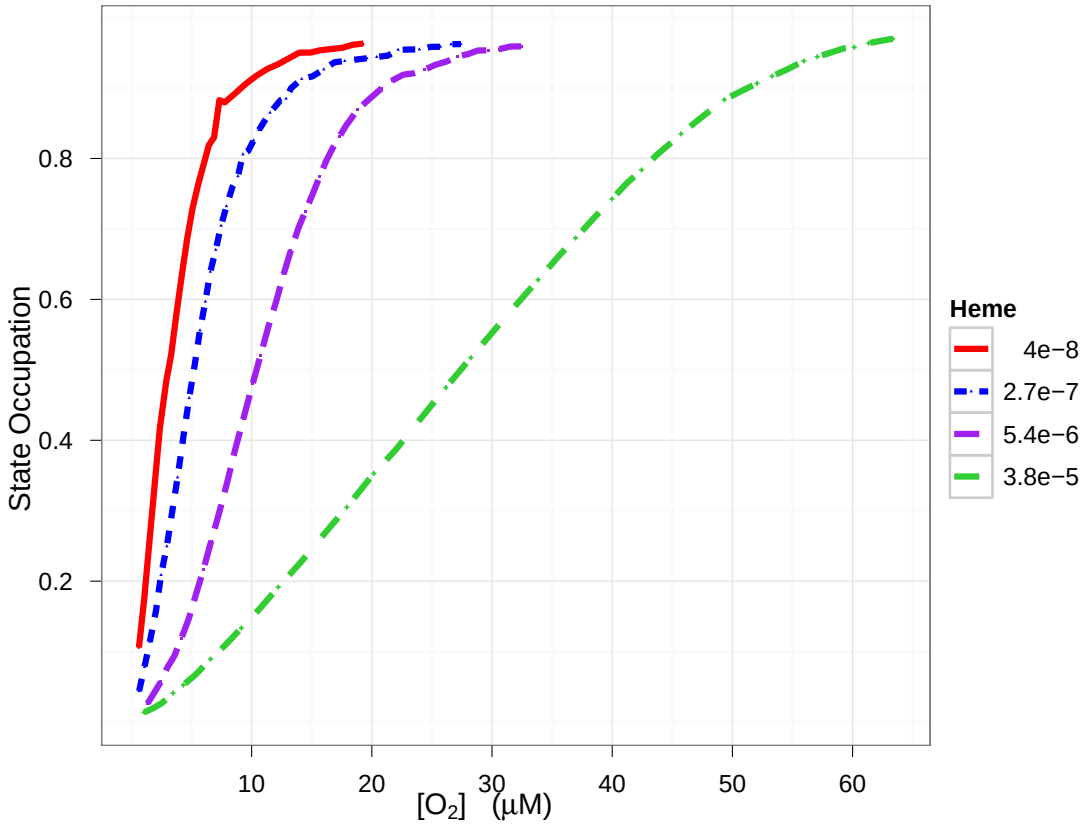
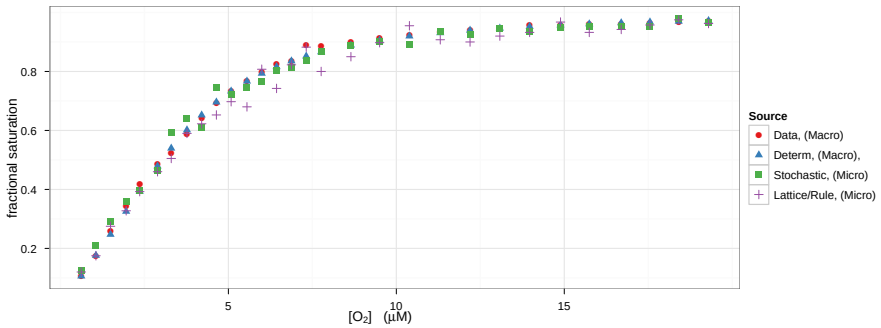
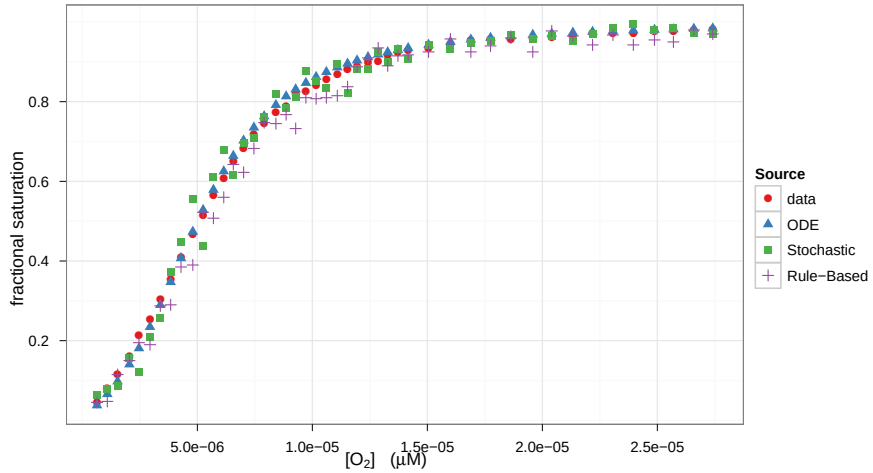


Figure 7.4: Percentage of hemoglobin bound vs. the total concentration of oxygen. The actual oxygen is modified in this plot as the original data did not account for the total oxygen in the system.

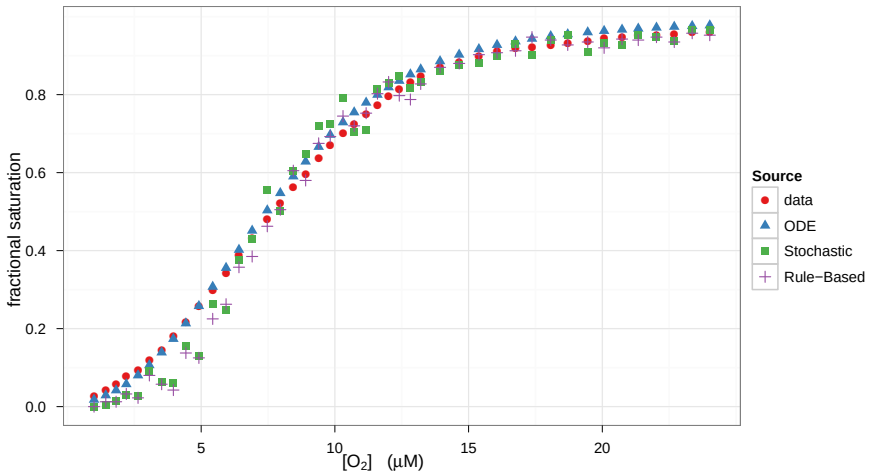


(a) ***[Heme]=40***nM

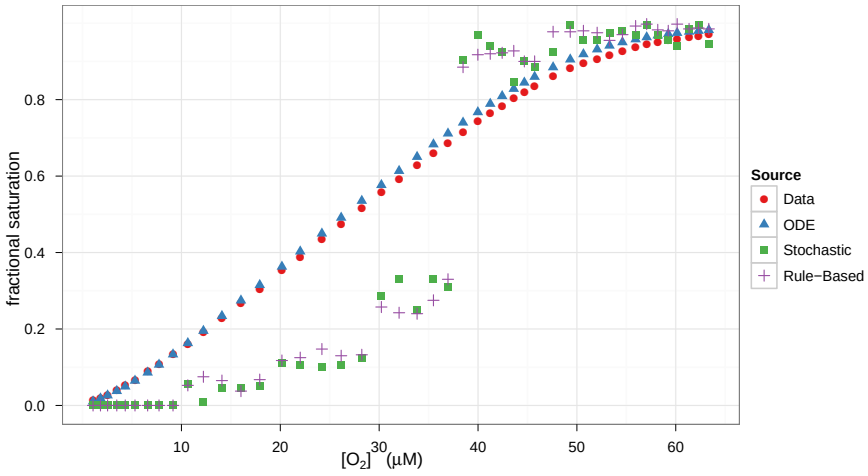


(b) ***[Heme]=0.27***μM

Figure 7.5: see next page



(c) ***[Heme]=5.4***μM



(d) ***[Heme]=38***μM

Figure 7.5: (cont.) Data fitting of the conceptual model of Ackers and Holt (ref insert 10) to the data of Mills et al. [1976]. The original data is in red and the computer-generated deterministic fits are in blue.

Chapter 8

Case III: A Simple Cooperative Dimer and Implications for Calcium Calmodulin Kinase II

In the previous chapter, we saw an example of hemoglobin exhibiting non-ergodic like behavior. By limiting the oxygen count in a system containing a single hemoglobin to a non-saturating number, we were able to significantly lower levels of oxygenation of hemoglobin. This is an exciting result, as there are cooperative processes that occur in volumes that regularly contain a ligand count that is less that the number that is sufficient to activate a complete set of quantal effectors contained within a macromolecule

In this chapter, we will build the simplest "toy" protein, with two binding sites that act as symmetric quantal effectors. The goal of this exercise is to determine what is necessary and sufficient to create non-ensemble-like behavior in biochemical systems. We will point out the limit at which a researcher could model a system with stochastic versus reaction-rate equations. Finally, we will then describe how a dendritic spine might fulfill the requirements for non-ergodic-like behavior in a

biological system.

8.1 The Properties of a Highly Cooperative Protein with Two Binding Sites

Our test system will use the following reaction scheme,

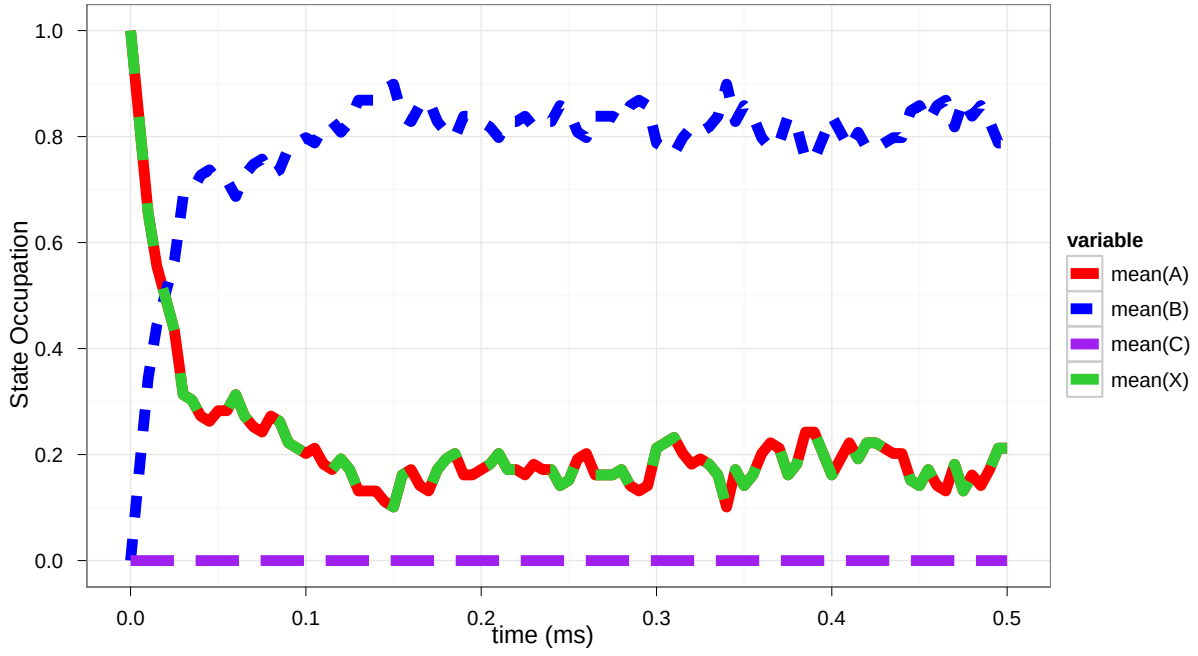
$$A + X \underset{}{\overset{K_{D1}}{\rightleftharpoons}} B \tag{8.1}$$

$$B + X \underset{}{\overset{K_{D2}}{\rightleftharpoons}} C \tag{8.2}$$

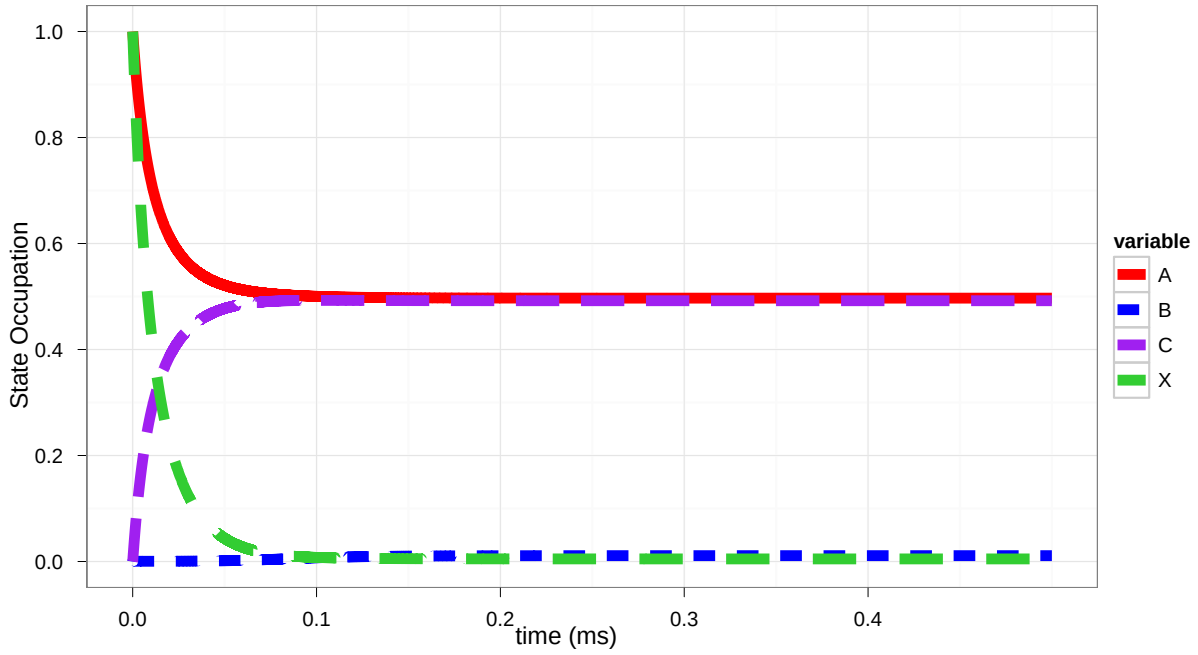
where A is an unbound protein, B is a singly bound protein, C is a doubly bound protein, and X is a ligand that may bind to A and B. Another constraint of our system is highly cooperative, $K_{D1} >> K_{D2}$.

8.2 When #A = 1 and #X = 1

In Figure 8.1, we see that the stochastic and deterministic results (Figures 8.1(a) and 8.1(b), respectively) differ from one another. Which result is correct? By observation, in a system containing A and X can only undergo the reaction depicted in Equation **??**. The reaction depicted in Equation **??** should be impossible, and therefore no C should be present in our system. So our stochastic simulation is correct, while the deterministic result is explicitly incorrect in that it predicts a completely different composition of our system kinetically and at equilibrium.



(a) ***Stochastic Simulation***



(b) ***Deterministic Simulation***

Figure 8.1: A comparison of deterministic and stochastic models of a system outlined in Equations 8.1 and 8.2. Initial conditions are #A=1 and #X=1.

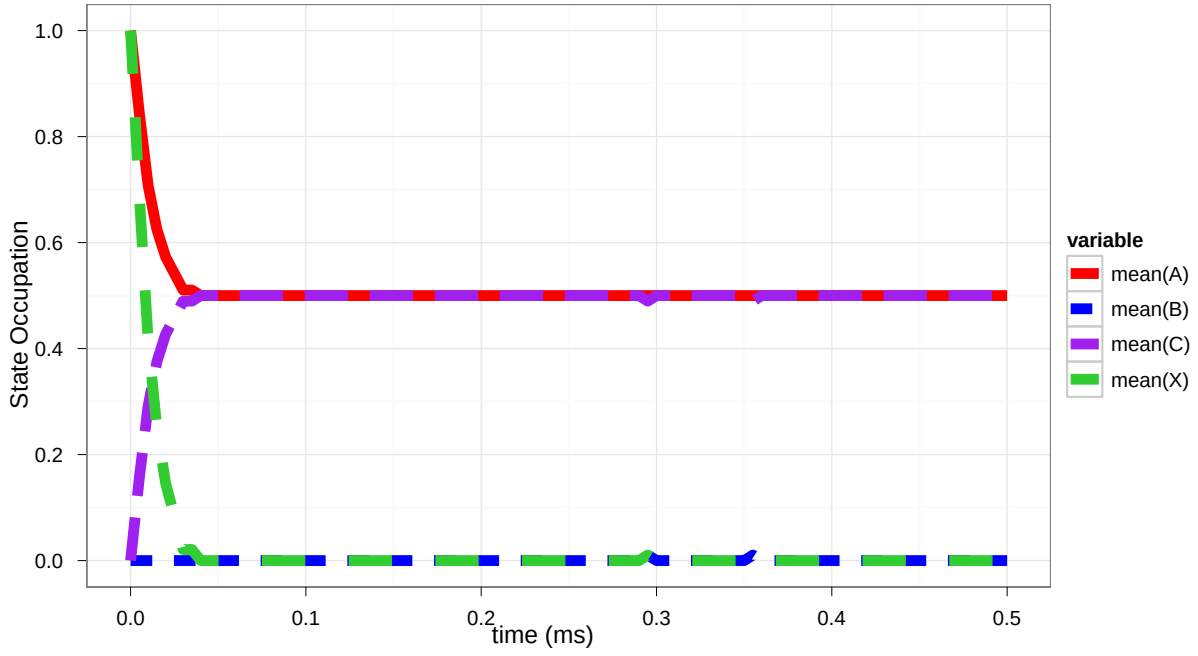
While minor discrepancies have been observed between stochastic and deterministic systems, this may be the first case where we can see that the stochastic result is a significantly more realistic representation of the likely physical reality than the deterministic result.

8.3 When #A = 2 and #X = 2 and the Idea of Inaccessible States

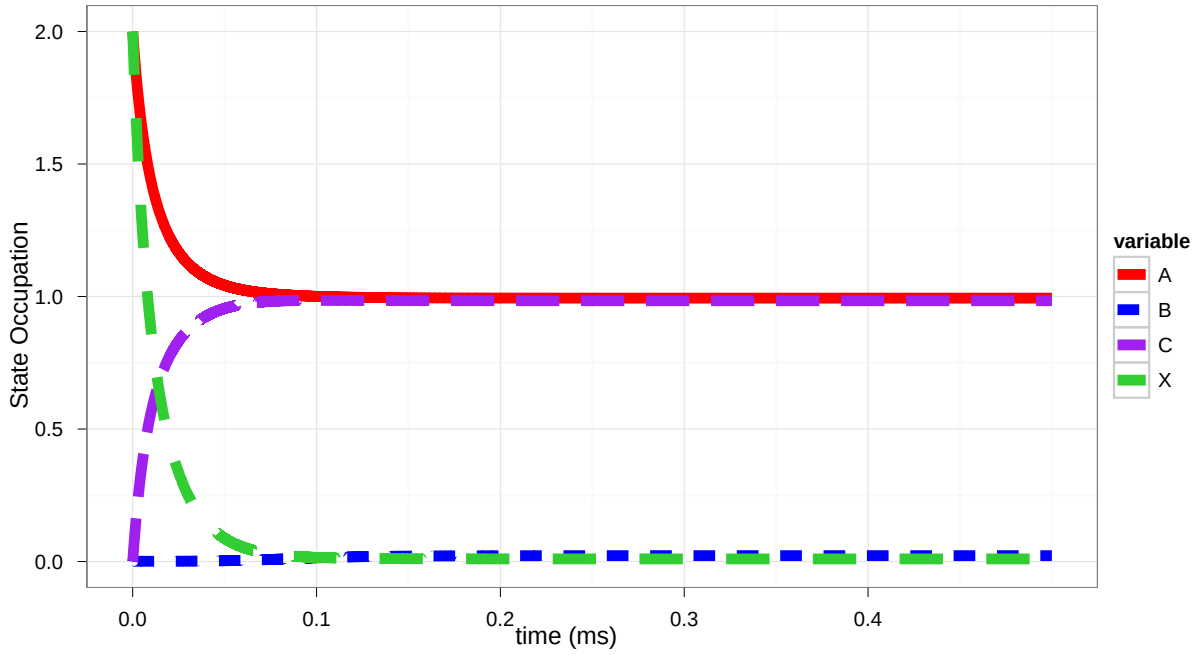
Figure 8.2 shows that our stochastic and deterministic systems perform similarly when the system represented in Figure 8.1 is doubled in size. Based on this apparent volume effect, we notice that the switch occurs again when a cooperative macromolecule may explore its full state space. The limit of the behavior seems to be the cutoff where some states that could occur in a classical formulation are restricted due to limited resources. This finding implies that we might be able to recover macroscopic behavior in the Ackers' model if the concentration was large enough.

8.4 Revisiting the Ackers' Model

In Figure 8.3, we increase the volume of the box of hemoglobin. With the first increase in volume (B), we observe the quantal behavior shifts to the left, as does the point at which four dioxygen are in the system. There is also a lesser discrepancy in the stochastic formulation at the point before there are eight dioxygen



(a) ***Stochastic Simulation***



(b) ***Deterministic Simulation***

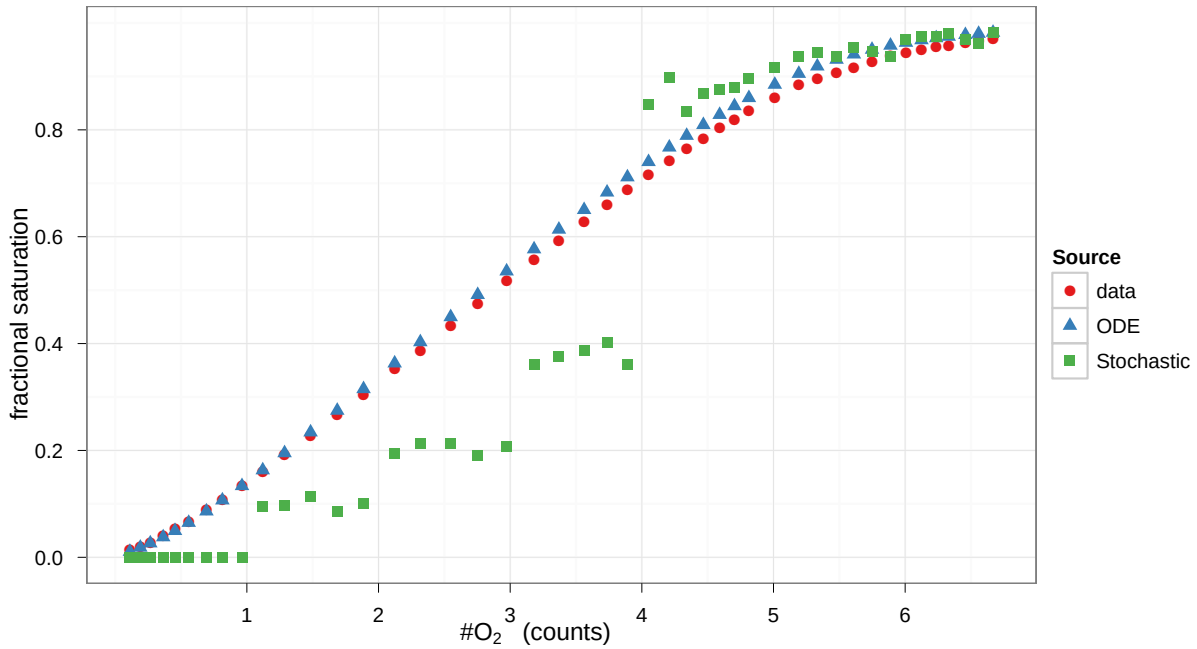
Figure 8.2: A comparison of deterministic and stochastic models of a system outlined in Equations 8.1 and 8.2. Initial conditions are #A=2 and #X=2.

molecules in the system. When the system size is doubled again (C), the non-ergodic-like behavior disappears into the noise.

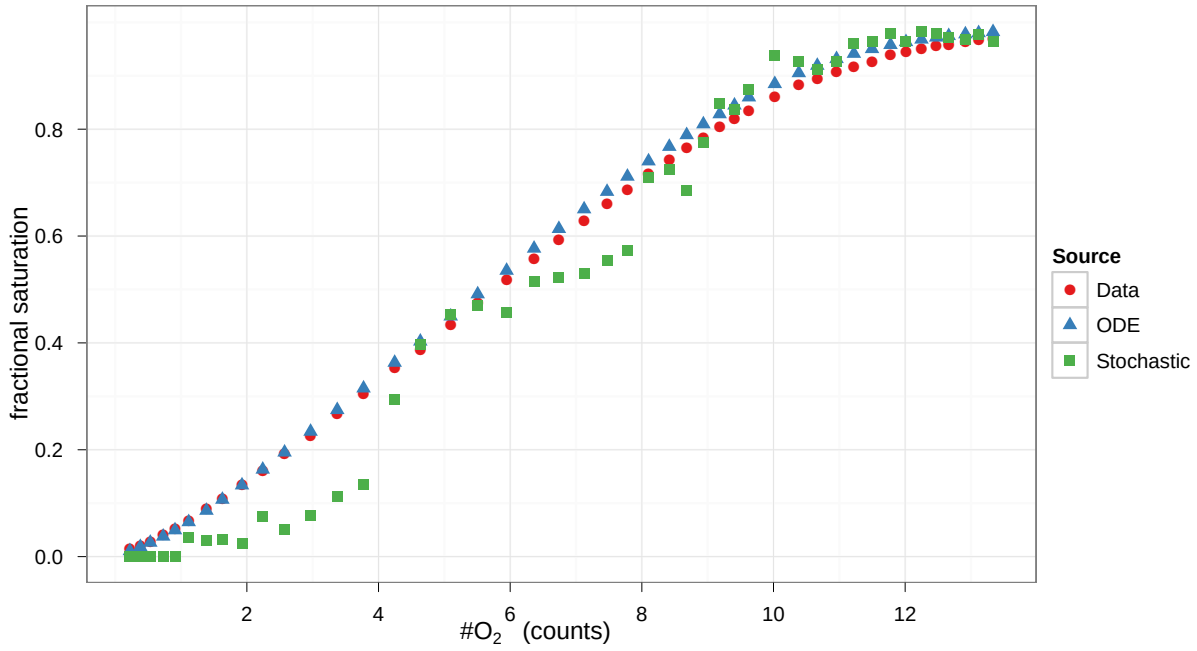
This is an exciting result as it shows that a cooperative system might allow us to resolve features of the quantal nature of macromolecular dynamics. What is desired is a more cooperative system that is activated by a ligand that is easier to control than dioxygen. Ca^{2+}/CaM/CaMKII is an obvious choice for such a system and it will be discussed in a later section of this chapter.

8.5 Ergodic Breaking or Incomparable Ensembles?

Experimental biologists gain insight into biological systems by studying biological systems in varying conditions and then inferring the behavior of the biological system based upon microscopic observations. By definition, biochemists often treat these ensembles as being comparable by directly comparing them. Furthermore modelers often expect stochastic and deterministic results to converge. For example, in 2005 Lok stated that Moleculizer, a stochastic rule-based modeling system, and BioNetGen (version 1), a deterministic rule-based modeling system should give comparable results. As I have shown in this chapter and the previous one, stochastic and deterministic results do not necessarily converge. Ironically Lok and Brent did not give their code Moleculizer enough credit, as their method should be more accurate at small volumes than classical methods that are more similar to BioNetGen (v.1) [2005].

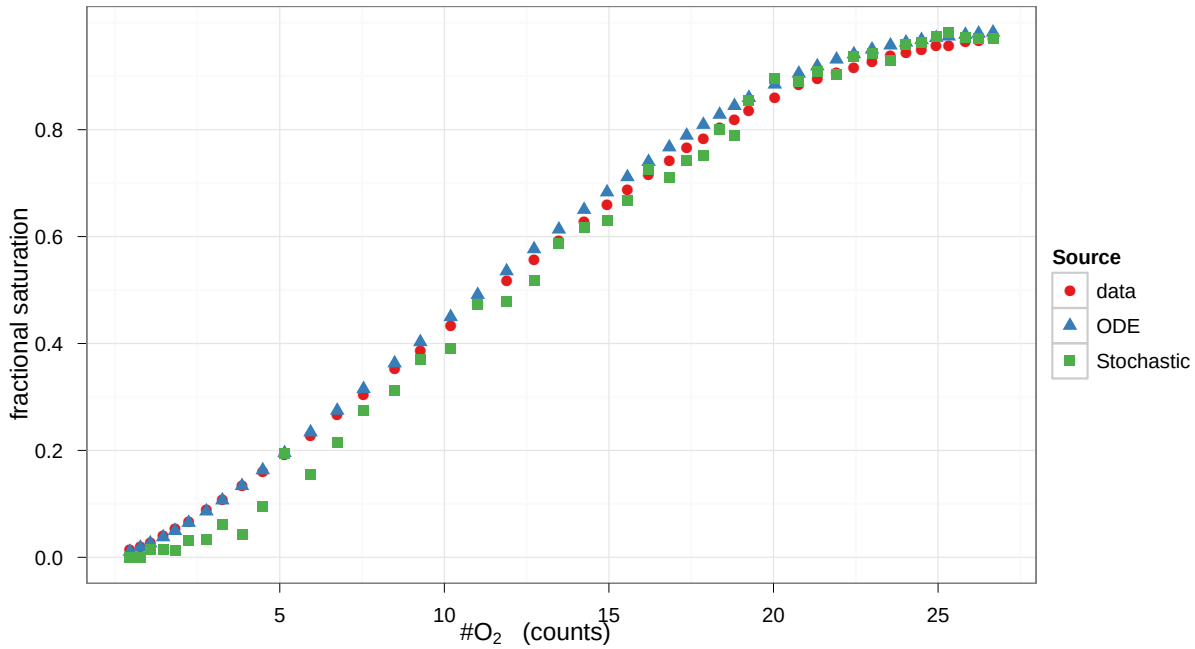


(a) ***1 Hb*** @ $38\mu M$ ***[Heme]***



(b) ***2 Hb*** @ $38\mu M$ ***[Heme]***

Figure 8.3: see next page



(c) ***4 Hb*** $@\ 38\mu M$ ***[Heme]***

Figure 8.3: A comparison of deterministic and stochastic models of a system outlined in Equations 8.1 and 8.2. Initial conditions are #A=2 and #X=2.

8.6 How Does This Relate to Calcium Signaling in the Dendritic Spine?

The cooperative relationship in Ca^{2+}/CaM/CaMKII signaling in the dendritic spine is actually a perfect domain to consider the quantal behavior of a macromolecule. While CaMKII can bind to 12 CaM molecules, that can in total bind to 48 ions of Ca^{2+}, it has a high particle requirementin order for it to explore its full state space.

Furthermore, the dendritic spine is very interesting in that it is very small and can be diffusionally restricted from the rest of the cell. This means that once it

reaches a basal level of free Ca^{2+}, it can control further entry to some degree. As we mentioned earlier, the resting Ca^{2+} concentration for a cell would result in a volume such as the dendritic spine having approximately four free calcium ions. A dendritic spine without CaMKII activity already present would be able to utilize Ca^{2+}/CaM to function as a switch beyond what can be described by classical/deterministic methods. The mechanism by which this is done when there is Ca^{2+}/CaM present is not directly apparent.

Currently, neuroscientists who are interested in learning and memory might analyze the phosphorylation of CaMKII under varying concentrations but at very nonphysical volumes. Doing so has consistently led to the understanding that CaMKII can become activated at very low levels of Ca^{2+}.

This has led to confusion as to how small amounts of Ca^{2+} can lead to long-term depression when experiments predict kinase activity. Based on these findings, a neuroscientist might consider taking steps toward understanding CaMKII activity under more physiologically realistic volumes.

It is understood that, at rest, a dendritic spine can have insufficient Ca^{2+} to saturate a CaMKII macromolecule bound to CaM. This means that CaMKII has zero probability of entering some of its state space even at high concentrations of Ca^{2+} that are equivalent to low numbers of Ca^{2+} in a small volume. In such a case the discrete system would be below the threshold for macroscopic behavior. This strongly points to the possibility of quantal effects being exploited at the synapse as a regular and ubiquitous feature of synaptic plasticity.

Chapter 9

Conclusions

In this study, we compare and contrast the capability of deterministic, standard stochastic, and modified stochastic models to accurately reproduce observed kinetic and thermodynamic data. Our modified stochastic representation appears to give us a simple method to tease out otherwise complex relationships between molecular states. Again our modified method is analogous to a lattice statistical model that employs that employs rules or reaction rate equations to replace *reaction rate constants*.

An additional complication is found in attempting to deduce mechanistic relationships such as cooperative influences from macroscopic data. This falls into the general classification of an inverse problem. An inverse-problem approach involves the prediction of structurally dependent microscopic mechanisms from observed macroscopic dynamics.

However, in many inverse problems, the ability to deduce mechanistic relationships may be ill-posed, because the dynamics of a single system may be reproduced with alternative plausible mechanisms. Pauling [1935] has previously demonstrated that many possible solutions are consistent with observed data of

the Hb + O_2 system. The deterministic and the more compact stochastic approach that we applied to the Ackers and Holt model and associated data provide alternative tools that can be applied to explore similar cooperative issues in biochemical reactions.

In the first application, the pH-dependent protonation of the zwitterion glycine is modeled, and the solutions are consistent. In the first part of the second case, the kinetic model of the sequential complexation of four dioxygen molecules by hemoglobin is modeled in an effort to account for the apparent cooperative relationship, or the enhanced binding affinity with respect to the successive uptake of oxygen, that is observed experimentally between the four different subunits of the hemoglobin molecule. In this kinetic model, all models considered showed consistent results.

Conversely, our thermodynamic models of Hb/O_2 complexation according to Ackers showed our first evidence of quantal behavior in both the standard and the modified stochastic formulations. These quantal effects lead to ergodic breaking that is apparently, due to some phenomena inherently present in cooperativity. Similar to the idea of a particle in a box, our simulation predicts that we might be observing the quantization of energy levels of Hb as O_2 is dialyzed into our system.

To our knowledge, inconsistencies in the average behavior of the stochastic versus the deterministic version of the same biochemical system have not been previously reported. The direct significance is that stochastic simulations of nearly

equivalent ratios of macromolecule binding sites to their ligands are more biologically realistic than deterministic models of the same systems. Rule-based modeling approaches are beneficial in that they allow us to directly confront the non-ideal nature of cooperative chemical reactions. We can directly extend these findings to the Ca^{++}/CaM/CaMKII system and see that with its high degree of possible cooperativity and its apparent switch-like/bistable behavior, it is possible that LTP uses quantal effects as a common component of activity-dependent-plasticity.

Finally, as we produce models that are increasingly more precise, we may be able to reverse engineer more accurate parameters for reactive force fields, an essential component of reactive molecular dynamics packages such as ReaxFF (van Duin et al. [2001]). As more rule-based modeling approaches are implemented to deal with the combinatorial complexity of many interacting components in a complex chemical reaction, we must keep in mind that these models may not give us much insight into the physicochemical properties that underlie the chemical reactions that we observe.

Appendices

Example Source Code

Case I: Glycine Protonation

Algorithms and Source Code 4: Glycine protonation represented as a system of ordinary differential equations, the stochastic simulation algorithm, and "rule-based" techniques

```
#!/usr/bin/env python2.6
# -*- coding: utf-8 -*-
"""
Glycine Negative cooperativity
example from "Binding and Linkage" 1990, page 59
"""
__docformat__ = "restructuredtext en"
#TODO hi
import numpy as np
import numpy.random as npr
import scipy as sp
import scipy.integrate as spi
import pylab as pl
def glyssa(tend,steps,trials,H):
    timepoints=np.arange(steps+1)*(tend/steps)
    datapoints=np.zeros(steps+1)
    for x in xrange(int(trials)):
        vol=1e-10
        nA=6.023e23
        prop = sp.zeros(4)
        y=sp.zeros(5)
        t=0
        cnt=0
        Ka=5.0e9/(nA*vol)
        Kb=2.0e4/(nA*vol)
        cab=1/100.0
        koff=1
```

```
        y[0]=H*nA*vol
        y[1]=1
        y[2]=1
        y[3]=0
        y[4]=0
##        print y[0]
        while tend>t:

            prop[0]=Ka*koff*cab**y[4] * y[0] * y[1]
            prop[1]=Kb*koff*cab**y[3] * y[0] * y[2]
            prop[2]=koff * y[3]
            prop[3]=koff * y[4]
            cprop=sp.cumsum(prop)
            r1=npr.random()
            tau = 1/cprop[-1]*sp.log(1/r1)
            t=tau+t
            while t>=cnt*timepoints[1] and cnt<steps+1:
                Y= (y[3]+y[4])/2.0
##                print y
                datapoints[cnt]+=Y
                cnt+=1
            if (t < tend):
                r2=npr.random()
                if   r2<cprop[0]/cprop[-1]:
                    y[0]-=1
                    y[1]-=1
                    y[3]+=1
                elif r2<cprop[1]/cprop[-1]:
                    y[0]-=1
                    y[2]-=1
                    y[4]+=1
                elif r2<cprop[2]/cprop[-1]:
                    y[0]+=1
                    y[1]+=1
                    y[3]-=1
                elif r2<cprop[3]/cprop[-1]:
                    y[0]+=1
                    y[2]+=1
                    y[4]-=1
            else:
                break
#    print datapoints/int(trials)
    return [timepoints,datapoints/int(trials)]

def glyssaNOT(tend,steps,trials,H):
    timepoints=np.arange(steps+1)*(tend/steps)
    datapoints=np.zeros(steps+1)
```

```
    for x in xrange(int(trials)):
        vol=1e-10
        nA=6.023e23
        prop = sp.zeros(4)
        y=sp.zeros(5)
        t=0
        cnt=0
        Ka=5.0e9/(nA*vol)
        Kb=2.0e4/(nA*vol)
        cab=1/100.0
        koff=1

        y[0]=H*nA*vol
        y[1]=1
        y[2]=1
        y[3]=0
        y[4]=0
##        print y[0]
        while tend>t:

            prop[0]=Ka/koff * y[0] * y[1]
            prop[1]=Kb/koff * y[0] * y[2]
            prop[2]=koff * y[3]
            prop[3]=koff * y[4]
            cprop=sp.cumsum(prop)
            r1=npr.random()
            tau = 1/cprop[-1]*sp.log(1/r1)
            t=tau+t
            while t>=cnt*timepoints[1] and cnt<steps+1:
                Y= (y[3]+y[4])/2.0
##                print y
                datapoints[cnt]+=Y
                cnt+=1
            if (t < tend):
                r2=npr.random()
                if   r2<cprop[0]/cprop[-1]:
                    y[0]-=1
                    y[1]-=1
                    y[3]+=1
                elif r2<cprop[1]/cprop[-1]:
                    y[0]-=1
                    y[2]-=1
                    y[4]+=1
                elif r2<cprop[2]/cprop[-1]:
                    y[0]+=1
                    y[1]+=1
                    y[3]-=1
```

```
                elif r2<cprop[3]/cprop[-1]:
                    y[0]+=1
                    y[2]+=1
                    y[4]-=1
            else:
                break
#    print datapoints/int(trials)
    return [timepoints,datapoints/int(trials)]
def glyrre(y,t):
    H=y[0]
    A=y[1]
    B=y[2]
    C=y[3]
    D=y[4]
    Ka=5.0e9
    Kb=2.0e4
    koff=0.999
    cab=1/100.0
    yprime=pl.zeros(5)
    yprime[0]=  -(Ka/koff*A*H)-(Kb/koff*A*H)-(Ka/koff*cab*C*H)-(
       Kb/koff*cab*B*H)+koff*(B+C+D+D)
    yprime[1]=  -(Ka/koff*A*H)-(Kb/koff*A*H)+(koff*B)+(koff*C)
    yprime[2]=  (Ka/koff*H*A)+koff*D-(koff*B)-Kb/koff*cab*H*B
    yprime[3]=  (Kb/koff*H*A)+koff*D-(koff*C)-Ka/koff*cab*H*C
    yprime[4]=  Kb/koff*cab*H*B+Ka/koff*cab*H*C-2.0*koff*D
    return yprime

if __name__ == '__main__':
    rreconc=[]
    rrevalue=[]
    mult = 2.0
    for expo in range(int(12*mult)):
        startH   = 1.0*10.0**(-expo/mult)
        startGly = 1.0e-13
        yzero=pl.array([startH,startGly,0,0,0])
    #    times=pl.linspace(0.0,0.5,1e4,endpoint=True)
        times=sp.arange(0,2.0,.005)
        test=spi.odeint(glyrre,yzero,times)
        print expo/3., ((test[-1,2]+test[-1,3])*.5+test[-1,4])/(
           startGly)
        rreconc.append(expo/mult)
        rrevalue.append(((test[-1,2]+test[-1,3])*.5+test[-1,4])/(
           startGly))

    testlist=[1e-13,1e-12,1e-11,1/(5.0*10**10),1/(1.5*10**10),1e-
       10,1/(4.5*10**9),1/(3.8*10**9),1/(2.0*10**9),1/(1.5*10**9)
       ,
```

```
            1e-9,1e-8,1e-7,1e-6,1e-5,
            1e-4,1e-3,1/600.0,1/500.0,1/200.0,1/150.0,1e-
               2,1/60.0,1/50.0,1/20.0,1/15.0,1e-1,1e-0]
numlist=[]
datalist=[]
for number in testlist:
    tempt,tempdata=glyssa(.99,9,3000,number)
    print -sp.log10(number), number, tempdata[-1]
    numlist.append(-sp.log10(number))
    datalist.append(tempdata[-1])

testlistNOT=[1e-13,1e-12,1e-11,1/(5.0*10**10),1/(1.5*10**10)
   ,1e-10,
        1/(4.5*10**9),1/(3.8*10**9),1/(2.0*10**9),1/(1.5*10**
           9),
        1e-9,1e-8,1e-7,1e-6,1e-5,1/(3.8*10**4),1/(2.0*10**4),
        1e-4,1/(3.8*10**3),1/(2.0*10**3),1e-3,1e-2,1e-1,1e-0]
numlistNOT=[]
datalistNOT=[]
for number in testlistNOT:
    tempt,tempdata=glyssaNOT(.99,9,3000,number)
    #print -sp.log10(number), number, tempdata[-1]
    numlistNOT.append(-sp.log10(number))
    datalistNOT.append(tempdata[-1])
pl.plot(rreconc,rrevalue,"g",label="Negative Cooperativity (
   ODE's)",ms=4,lw=2)
pl.yticks((0, 0.5, 1), ('0', r'$\frac{1}{2}$', '1'), color =
   'k', size = 18)
pl.xticks((2,4,6,8,10,12), ('2','4','6','8','10','12'), color
    = 'k', size = 18)

pl.plot(numlist,datalist,label="Negative Cooperativity (
   Stochastic Sim)",ms=4,lw=2)
pl.plot(numlistNOT,datalistNOT,"r",label="Without Charge
   Interference",ms=4,lw=2)
for (a, b) in zip(rreconc,rrevalue ):
    print a,b
for (a, b) in zip(numlist,datalist ):
    print a,b
for (a, b) in zip(numlistNOT,datalistNOT ):
    print a,b


pl.legend(loc="best")
pl.xlabel("pH",size=18)
pl.ylabel("fractional saturation",size=18)
pl.show()
```

Case II: Gibson's Oxygen Binding to Hemoglobin

Some relevant math:

$$\frac{41.5\mu M\ heme}{L} \times \frac{1Hb}{4Heme} \times \frac{1M}{10^6\mu M} \times \frac{6.023 \times 10^{23} Molecs}{1M} = \frac{6.249 \times 10^{18} Molecs}{L}$$

$$\frac{1}{\frac{6.249 \times 10^{18}}{L}} = \frac{1.6 \times 10^{-19} L}{1Hb}$$

$$\frac{124\mu M\ O_2}{L} \times \frac{1M}{10^6\mu M} \times \frac{6.023 \times 10^{23} Molec.s}{1\ M} \times \frac{1.6 \times 10^{-19} L}{1} = 1156416.$$

Algorithms and Source Code 5: Gibson's kinetic model of Hb/O_2, applied to his own data as reported in [1970]

```
#!/usr/bin/env python2.6
# -*- coding: utf-8 -*-
"""
Gibson 1970

This will be the standard to compare to for Gibson 1970
"""
__docformat__ = "restructuredtext en"
"""
:Parameters:

    - kon1=17.7
    - kon2=33.2
    - kon3=4.89
    - kon4=33.0
    - koff1=1900.
    - koff2=158.
    - koff3=539.
    - koff4=50.
"""
import numpy as np
import numpy.random as npr
```

```
import scipy as sp
import scipy.integrate as spi
import pylab as pl
def myssa(tend,steps,trials):
    timepoints=np.arange(steps+1)*(tend/steps)
    datapoints=np.zeros(steps+1)
    for x in xrange(int(trials)):
        prop = sp.zeros(8)
        y=sp.zeros(9)
        t=0
        cnt=0
        vol=1.6e-19
        nA=6.023e23
        uM=nA*10**-6
        Oxy=11
        if npr.random()<0.9:
            Oxy=Oxy+1
        Hb=int(1)
        y[0]=Oxy
        y[1]=1
        y[2]=1
        y[3]=1
        y[4]=1
        while tend>t:
            if(y[1]+y[2]+y[3]+y[4]==4):
                ssakon=17.7/(nA*vol)*1e6*1/4
                ssakoff=0
            elif(y[1]+y[2]+y[3]+y[4]==3):
                ssakon=33.2/(nA*vol)*1e6*1/3
                ssakoff=1900
            elif(y[1]+y[2]+y[3]+y[4]==2):
                ssakon=4.89/(nA*vol)*1e6*1/2
                ssakoff=158/2
            elif(y[1]+y[2]+y[3]+y[4]==1):
                ssakon=33.0/(nA*vol)*1e6
                ssakoff=539/3
            else:
                ssakon=0
                ssakoff=50/4
            prop[0]=ssakon * y[0] * y[1]
            prop[1]=ssakon * y[0] * y[2]
            prop[2]=ssakon * y[0] * y[3]
            prop[3]=ssakon * y[0] * y[4]
            prop[4]=ssakoff * y[5]
            prop[5]=ssakoff * y[6]
            prop[6]=ssakoff * y[7]
            prop[7]=ssakoff * y[8]
```

```
            cprop=sp.cumsum(prop)
            r1=npr.random()
            tau = 1/cprop[-1]*sp.log(1/r1)
            t=tau+t
            while t>=cnt*timepoints[1] and cnt<steps+1:
                Y= (y[5]+y[6]+y[7]+y[8])/(4)
                datapoints[cnt]+=Y
                cnt+=1
            if (t < tend):
                r2=npr.random()
                if   r2<cprop[0]/cprop[-1]:
                    y[0]-=1
                    y[1]-=1
                    y[5]+=1
                elif r2<cprop[1]/cprop[-1]:
                    y[0]-=1
                    y[2]-=1
                    y[6]+=1
                elif r2<cprop[2]/cprop[-1]:
                    y[0]-=1
                    y[3]-=1
                    y[7]+=1
                elif r2<cprop[3]/cprop[-1]:
                    y[0]-=1
                    y[4]-=1
                    y[8]+=1
                elif r2<cprop[4]/cprop[-1]:
                    y[0]+=1
                    y[1]+=1
                    y[5]-=1
                elif r2<cprop[5]/cprop[-1]:
                    y[0]+=1
                    y[2]+=1
                    y[6]-=1
                elif r2<cprop[6]/cprop[-1]:
                    y[0]+=1
                    y[3]+=1
                    y[7]-=1
                elif r2<=cprop[7]/cprop[-1]:
                    y[0]+=1
                    y[4]+=1
                    y[8]-=1
            else:
                break
    return [timepoints,datapoints/int(trials)]
```

```
117
def rre_hemosub1(y,t,kon1,kon2,kon3,kon4,koff1,koff2,koff3,koff4)
    :
    yprime = sp.zeros(6)
    yprime[0]= -kon1 * y[0] * y[1] + koff1 * y[2]-kon2 * y[0] * y
        [2] +\
                 koff2 * y[3] -kon3 * y[0] * y[3] + koff3 * y[4] -
                     \
122              kon4 * y[0] * y[4] + koff4 * y[5]
    yprime[1]= -kon1 * y[0] * y[1] + koff1 * y[2]
    yprime[2]= -kon2 * y[0] * y[2] + koff2 * y[3] + kon1 * y[0] *
        y[1] - koff1 * y[2]
    yprime[3]= -kon3 * y[0] * y[3] + koff3 * y[4] + kon2 * y[0] *
        y[2] - koff2 * y[3]
    yprime[4]= -kon4 * y[0] * y[4] + koff4 * y[5] + kon3 * y[0] *
        y[3] - koff3 * y[4]
127 yprime[5]=                                          kon4 * y[0] *
        y[4] - koff4 * y[5]
    return yprime


def ssa(tend,steps,trials):
132 timepoints=np.arange(steps+1)*(tend/steps)
    datapoints=np.zeros(steps+1)
    for x in xrange(int(trials)):
        y = sp.zeros(6)
        prop = sp.zeros(8)
137     y=sp.zeros(6)

        t=0
        cnt=0
        vol=1.6e-19
142


        nA=6.023e23
        uM=nA*10**-6
        ssakon1=17.7*1e6/(nA*vol)
147     ssakon2=33.2 *1e6/(nA*vol)
        ssakon3=4.89 *1e6/(nA*vol)
        ssakon4=33.0 *1e6/(nA*vol)

        ssakoff1=1900.
152     ssakoff2=158.
        ssakoff3=539.
        ssakoff4=50.
        Oxy=int(11)
        if npr.random()<0.9:
```

```
            Oxy = Oxy +1
        Hb=int(1)
        y[0]=Oxy
        y[1]=Hb
        while tend>t:
            prop[0]=ssakon1 * y[0] * y[1]
            prop[1]=ssakon2 * y[0] * y[2]
            prop[2]=ssakon3 * y[0] * y[3]
            prop[3]=ssakon4 * y[0] * y[4]
            prop[4]=ssakoff1 * y[2]
            prop[5]=ssakoff2 * y[3]
            prop[6]=ssakoff3 * y[4]
            prop[7]=ssakoff4 * y[5]
            cprop=sp.cumsum(prop)
            r1=npr.random()
            tau = 1/cprop[-1]*sp.log(1/r1)
            t=tau+t
            while t>=cnt*timepoints[1] and cnt<steps+1:
                Y= (y[5]+y[4]*.75+y[3]*.5+y[2]*.25)/(Hb)
                datapoints[cnt]+=Y
                #print t, cnt*timepoints[1] ,Y,y,Oxy,Hb
                cnt+=1
            if (t < tend):
                r2=npr.random()
                if   r2<cprop[0]/cprop[-1]:
                    y[0]-=1
                    y[1]-=1
                    y[2]+=1
                elif r2<cprop[1]/cprop[-1]:
                    y[0]-=1
                    y[2]-=1
                    y[3]+=1
                elif r2<cprop[2]/cprop[-1]:
                    y[0]-=1
                    y[3]-=1
                    y[4]+=1
                elif r2<cprop[3]/cprop[-1]:
                    y[0]-=1
                    y[4]-=1
                    y[5]+=1
                elif r2<cprop[4]/cprop[-1]:
                    y[0]+=1
                    y[1]+=1
                    y[2]-=1
                elif r2<cprop[5]/cprop[-1]:
                    y[0]+=1
                    y[2]+=1
```

```
                    y[3]-=1
                elif r2<cprop[6]/cprop[-1]:
                    y[0]+=1
                    y[3]+=1
                    y[4]-=1
                elif r2<=cprop[7]/cprop[-1]:
                    y[0]+=1
                    y[4]+=1
                    y[5]-=1
            else:
                break
    print ssakon1,ssakon2
    return [timepoints,datapoints/int(trials)]


def fig1aGibson1970():
    a=sp.array([[0.0, 0.502279056001],
    [0.00049173089564, 0.590841492954],
    [0.00099581398037, 0.654261224638],
    [0.00149921986175, 0.710287464286],
    [0.00200221942111, 0.761877608713],
    [0.00251838381359, 0.797197238313],
    [0.00303414188405, 0.828080772692],
    [0.00352004883089, 0.853059178138],
    [0.0040202041362,  0.873596656015],
    [0.00450529843902, 0.889702871019],
    [0.00501902489943, 0.898405929291],
    [0.00553315768186, 0.911545082784],
    [0.00604647782027, 0.915812045835],
    [0.00651596935793, 0.921572204336],
    [0.00701504113788, 0.930280094956]])
    return a


if __name__ == '__main__':
    """This is the main docstring"""
    data=fig1aGibson1970()
    t,y=ssa(0.04,900,200.)
    myt,myy=myssa(0.04,900,1000.)
    print myy
    kon1  =   17.7
    kon2  =   33.2
    kon3  =    4.89
    kon4  =   33.0
    koff1 = 1900.0
    koff2 =  158.0
```

```
    koff3 =  539.0
252 koff4 =   50.0
    Hb    =   41.5/4
    Oxy   =  124
    yzero =   pl.array([Oxy,Hb,0,0,0,0])
    times = pl.linspace(0,.04,1000,endpoint=True)
257 SingleSub = sp.integrate.odeint(rre_hemosub1, yzero, times,
       args=(kon1,kon2,kon3,kon4,koff1,koff2,koff3,koff4,))
    data=fig1aGibson1970()

    pl.plot(times, (Oxy-SingleSub[:,0])/(Hb*4), label="RRE
       Calculated Fig1 A") # plot theta vs t
    pl.plot(data[:,0]+.0015,data[:,1],'r+',label="Digitized Fig1
       A",ms=10)
262 pl.plot(t,y,'gx',label="Standard Gillespie",ms=4)
    pl.plot(myt,myy,'cx',label="My Rule Based Method",ms=4)

    pl.legend(loc="best")
    pl.xlabel("time")
267 pl.ylabel("fractional saturation")
    pl.show()
```

Case IIB: Ackers' Thermodynamic Model of Hemoglobin/O_2^{2+}

Algorithms and Source Code 6: A deterministic model of Hb/O_2 dynamics as presented by Ackers KinpySrinivasan

```
#!/usr/bin/python
import numpy as np
import numpy.random as npr
import scipy as sp
import scipy.integrate as spi
import scipy.optimize as spo
import pylab as pl
import sys

from scipy import *
import scipy.integrate as itg

def  ackersrre(karr, Oxy):
    """ Function doc """
    kon = karr[0]
    mult1=karr[1]
    mult2=karr[2]
    mult3=karr[3]
    Oxy = float(Oxy[0])
    t = arange(0, 3, 0.005)
#    Oxy = 3.8e-4
    '''
    ## Reaction ##

    #Some second order reaction
    A + O <-> B
    A + O <-> C
    B + O <-> E
    B + O <-> D
    B + O <-> F
    C + O <-> D
    C + O <-> F
    C + O <-> G
    D + O <-> H
    D + O <-> I
    E + O <-> H
    F + O <-> H
    F + O <-> I
    G + O <-> I
    H + O <-> J
    I + O <-> J
```

```
    ## Mapping ##

    A   0        -1*v_0(y[0], y[2], y[1]) -1*v_1(y[0], y[3], y
       [1])
    O   1        -1*v_0(y[0], y[2], y[1]) -1*v_1(y[0], y[3], y
       [1]) -1*v_2(y[2], y[4], y[1]) -1*v_3(y[2], y[5], y[1]) -1*
       v_4(y[2], y[1], y[6]) -1*v_5(y[3], y[5], y[1]) -1*v_6(y
       [3], y[1], y[6]) -1*v_7(y[3], y[7], y[1]) -1*v_8(y[8], y
       [5], y[1]) -1*v_9(y[9], y[5], y[1]) -1*v_10(y[8], y[4], y
       [1]) -1*v_11(y[8], y[1], y[6]) -1*v_12(y[9], y[1], y[6])
       -1*v_13(y[9], y[1], y[7]) -1*v_14(y[8], y[10], y[1]) -1*
       v_15(y[9], y[10], y[1])
    B   2        +1*v_0(y[0], y[2], y[1]) -1*v_2(y[2], y[4], y
       [1]) -1*v_3(y[2], y[5], y[1]) -1*v_4(y[2], y[1], y[6])
    C   3        +1*v_1(y[0], y[3], y[1]) -1*v_5(y[3], y[5], y
       [1]) -1*v_6(y[3], y[1], y[6]) -1*v_7(y[3], y[7], y[1])
    E   4        +1*v_2(y[2], y[4], y[1]) -1*v_10(y[8], y[4], y
       [1])
    D   5        +1*v_3(y[2], y[5], y[1]) +1*v_5(y[3], y[5], y
       [1]) -1*v_8(y[8], y[5], y[1]) -1*v_9(y[9], y[5], y[1])
    F   6        +1*v_4(y[2], y[1], y[6]) +1*v_6(y[3], y[1], y
       [6]) -1*v_11(y[8], y[1], y[6]) -1*v_12(y[9], y[1], y[6])
    G   7        +1*v_7(y[3], y[7], y[1]) -1*v_13(y[9], y[1], y
       [7])
    H   8        +1*v_8(y[8], y[5], y[1]) +1*v_10(y[8], y[4], y
       [1]) +1*v_11(y[8], y[1], y[6]) -1*v_14(y[8], y[10], y[1])
    I   9        +1*v_9(y[9], y[5], y[1]) +1*v_12(y[9], y[1], y
       [6]) +1*v_13(y[9], y[1], y[7]) -1*v_15(y[9], y[10], y[1])
    J   10       +1*v_14(y[8], y[10], y[1]) +1*v_15(y[9], y[10],
       y[1])
    '''

    dy = lambda y, t: array([\
     -1*v_0(y[0], y[2], y[1]) -1*v_1(y[0], y[3], y[1]),\
     -1*v_0(y[0], y[2], y[1]) -1*v_1(y[0], y[3], y[1]) -1*v_2(y[2
        ], y[4], y[1]) -1*v_3(y[2], y[5], y[1]) -1*v_4(y[2], y[1]
        , y[6]) -1*v_5(y[3], y[5], y[1]) -1*v_6(y[3], y[1], y[6])
         -1*v_7(y[3], y[7], y[1]) -1*v_8(y[8], y[5], y[1]) -1*v_9
        (y[9], y[5], y[1]) -1*v_10(y[8], y[4], y[1]) -1*v_11(y[8]
        , y[1], y[6]) -1*v_12(y[9], y[1], y[6]) -1*v_13(y[9], y[1
        ], y[7]) -1*v_14(y[8], y[10], y[1]) -1*v_15(y[9], y[10],
        y[1]),\
     +1*v_0(y[0], y[2], y[1]) -1*v_2(y[2], y[4], y[1]) -1*v_3(y[2
        ], y[5], y[1]) -1*v_4(y[2], y[1], y[6]),\
```

```
     +1*v_1(y[0], y[3], y[1]) -1*v_5(y[3], y[5], y[1]) -1*v_6(y[3
        ], y[1], y[6]) -1*v_7(y[3], y[7], y[1]),\
     +1*v_2(y[2], y[4], y[1]) -1*v_10(y[8], y[4], y[1]),\
     +1*v_3(y[2], y[5], y[1]) +1*v_5(y[3], y[5], y[1]) -1*v_8(y[8
        ], y[5], y[1]) -1*v_9(y[9], y[5], y[1]),\
     +1*v_4(y[2], y[1], y[6]) +1*v_6(y[3], y[1], y[6]) -1*v_11(y[
        8], y[1], y[6]) -1*v_12(y[9], y[1], y[6]),\
67   +1*v_7(y[3], y[7], y[1]) -1*v_13(y[9], y[1], y[7]),\
     +1*v_8(y[8], y[5], y[1]) +1*v_10(y[8], y[4], y[1]) +1*v_11(y
        [8], y[1], y[6]) -1*v_14(y[8], y[10], y[1]),\
     +1*v_9(y[9], y[5], y[1]) +1*v_12(y[9], y[1], y[6]) +1*v_13(y
        [9], y[1], y[7]) -1*v_15(y[9], y[10], y[1]),\
     +1*v_14(y[8], y[10], y[1]) +1*v_15(y[9], y[10], y[1])\
    ])
72
    #Initial concentrations:
    y0 = array([\
    #A
    4e-8/4,\
77  #O
    Oxy,\
    #B
    0.0,\
    #C
82  0.0,\
    #E
    0.0,\
    #D
    0.0,\
87  #F
    0.0,\
    #G
    0.0,\
    #H
92  0.0,\
    #I
    0.0,\
    #J
    0.0,\
97  ])



    #A + O <-> B
102 v_0 = lambda A, B, O : k0 * A**1 * O**1 - k0r * B**1
    k0 = kon *2
    k0r = 11.0
```

```
    #A + O <-> C
v_1 = lambda A, C, O : k1 * A**1 * O**1 - k1r * C**1
k1 = kon*2
k1r = 11.0

    #B + O <-> E
v_2 = lambda B, E, O : k2 * B**1 * O**1 - k2r * E**1
k2 = kon
k2r = 11.0

    #B + O <-> D
v_3 = lambda B, D, O : k3 * B**1 * O**1 - k3r * D**1
k3 = kon * mult1
k3r = 11.0

    #B + O <-> F
v_4 = lambda B, O, F : k4 * B**1 * O**1 - k4r * F**1
k4 = kon
k4r = 11.0

    #C + O <-> D
v_5 = lambda C, D, O : k5 * C**1 * O**1 - k5r * D**1
k5 = kon *mult1
k5r = 11.0

    #C + O <-> F
v_6 = lambda C, O, F : k6 * C**1 * O**1 - k6r * F**1
k6 = kon
k6r = 11.0

    #C + O <-> G
v_7 = lambda C, G, O : k7 * C**1 * O**1 - k7r * G**1
k7 = kon
k7r = 11.

    #D + O <-> H
v_8 = lambda H, D, O : k8 * D**1 * O**1 - k8r * H**1
k8 = kon*mult1
k8r = 11.0

    #D + O <-> I
v_9 = lambda I, D, O : k9 * D**1 * O**1 - k9r * I**1
k9 = kon*mult1
k9r = 11.0

    #E + O <-> H
```

```
152     v_10 = lambda H, E, O : k10 * E**1 * O**1 - k10r * H**1
        k10 = kon * mult2*2
        k10r = 11.0

        #F + O <-> H
157     v_11 = lambda H, O, F : k11 * F**1 * O**1 - k11r * H**1
        k11 = kon*mult2
        k11r = 11.0

        #F + O <-> I
162     v_12 = lambda I, O, F : k12 * F**1 * O**1 - k12r * I**1
        k12 = kon * mult2
        k12r = 11.0

        #G + O <-> I
167     v_13 = lambda I, O, G : k13 * G**1 * O**1 - k13r * I**1
        k13 = kon *2 * mult2
        k13r = 11.0

        #H + O <-> J
172     v_14 = lambda H, J, O : k14 * H**1 * O**1 - k14r * J**1
        k14 = kon*mult3
        k14r = 11.0

        #I + O <-> J
177     v_15 = lambda I, J, O : k15 * I**1 * O**1 - k15r * J**1
        k15 = kon *mult3
        k15r = 11.0


182     Y = itg.odeint(dy, y0, t)
        #print Y[-1,:], Oxy, (Oxy - Y[-1,1]),(Oxy - Y[-1,1]) / 3.8e-5
        return (Oxy - Y[-1,1]) / 4e-8

    def getData (fName):
187     """ Function doc """
        fp = pl.loadtxt(fName)

        return (fp[:,1],fp[:,0])

192



    def plot_Ackers(x,v):
        data = []
197     for val in x:
            data.append(ackersrre(v,[val,val]))
```

```
    return data
def resid_Ackers(v,x,y):
    err =0
    for cnt in range(len(x)):
        err+=abs(y[cnt]-ackersrre(v,[x[cnt],x[cnt]]))
        #print y[cnt], ackersrre(v,[x[cnt],x[cnt]]),x[cnt]
    print "Error", err,v
    return err


def main(inFile, kon, mult1, mult2, mult3):
    """Function Main Doc"""
    #Sprint #(inFile)
    [x,y ] =getData(inFile)
    y=y/max(y)*0.9699144577
    #dataPoints= fp.readlines()
    #pl.plot(x,y)
    #pl.show()
    #y=y/max(y)
    e=lambda v, x,y:(((ackersrre(v,x)-y)**2)**.5)
   # print x
    p0 = array([kon, mult1, mult2 ,mult3])
#     [a,b] = spo.curve_fit(ackersrre,x,y)
 #    plsq = spo.leastsq(e, p0, args=(y, x), maxfev=2000)
    v = spo.fmin(resid_Ackers,p0,args=(x,y))
    #v= spo.anneal(resid_Ackers,p0,args=(x,y))
    print v

    res= resid_Ackers(p0,x,y)

if __name__=='__main__':
    dataFile = sys.argv[1]
    kon = float(sys.argv[2])
    mult1 = float(sys.argv[3])
    mult2 = float(sys.argv[4])
    mult3 = float(sys.argv[5])
    main(dataFile,kon,mult1,mult2, mult3)
```

Algorithms and Source Code 7: A standard stochastic model of Hb/O_2 dynamics as presented by Ackers

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import numpy as np
import numpy.random as npr
import scipy as sp
import scipy.integrate as spi
import scipy.optimize as spo
import pylab as pl
import sys, copy


size = 1.




def hemossa_alt(tend,steps,trials,O2,kon_orig,mult1,mult2,mult3):
#    timepoints=np.arange(steps+1)*(tend/steps)
    datapoints=np.zeros(trials)
    timepoints=np.arange(steps+1)#*(float(tend)/float(steps))
    #print "\n\n\nfloat \t ",len(tend), steps
    timepoints=timepoints*(float(tend)/float(steps))
    for x in xrange(int(trials)):
        #print "Oxygen", O2
        vol=1e-10
        nA=6.023e23
        prop = sp.zeros(32)
        y=sp.zeros(11)
        t=0
        cnt=0
        # size = 10
        kon  = kon_orig/size
        kon1 = kon * mult1
        kon2 = kon * mult2
        kon3 = kon * mult3

        koff=11
        two= 2.

        k0 = kon *two
        k0r = 11.0
        k1 = kon*two
        k1r = 11.0
        k2 = kon *two
```

```
        k2r = 11.0
        k3 = kon * mult1
        k3r = 11.0
        k4 = kon
        k4r = 11.0
        k5 = kon *mult1
        k5r = 11.0
        k6 = kon
        k6r = 11.0
        k7 = kon

        k7r = 11.*two
        k8 = kon*mult1
        k8r = 11.0
        k9 = kon*mult1
        k9r = 11.0
        k10 = kon * mult2*two
        k10r = 11.0
        k11 = kon*mult2
        k11r = 11.0
        k12 = kon * mult2
        k12r = 11.0
        k13 = kon *two * mult2
        k13r = 11.0
        k14 = kon*mult3
        k14r = 11.0*two
        k15 = kon *mult3
        k15r = 11.0*two

        y[0]=1 * size
        y[1]=0
        y[2]=0
        y[3]=0
        y[4]=0
        y[5]=0
        y[6]=0
        y[7]=0
        y[8]=0
        y[9]=0
        y[10]=O2

        while tend>t:
            #print "y= ",y,t,
            prop[0] = k0 * y[0]**1 * y[10]**1     #y[1]+=1; y
                [0]-=1; y[10]-=1;
            prop[1] = k0r * y[1]**1               #y[1]-=1; y
                [0]+=1; y[10]+=1;
```

```
    prop[2] = k1 * y[0]**1 * y[10]**1    #y[2]+=1; y
        [0]-=1; y[10]-=1;
    prop[3] = k1r * y[2]**1              #y[2]-=1; y
        [0]+=1; y[10]+=1;
    prop[4] = k2 * y[1]**1 * y[10]**1    #y[4]+=1; y
        [1]-=1; y[10]-=1;
    prop[5] = k2r * y[4]**1              #y[4]-=1; y
        [1]+=1; y[10]+=1;
    prop[6] = k3 * y[1]**1 * y[10]**1    #y[3]+=1; y
        [1]-=1; y[10]-=1;
    prop[7] = k3r * y[3]**1              #y[3]-=1; y
        [1]+=1; y[10]+=1;
    prop[8] = k4 * y[1]**1 * y[10]**1    #y[5]+=1; y
        [1]-=1; y[10]-=1;
    prop[9] = k4r * y[5]**1              #y[5]-=1; y
        [1]+=1; y[10]+=1;
    prop[10] = k5 * y[2]**1 * y[10]**1   #y[3]+=1; y
        [2]-=1; y[10]-=1;
    prop[11] = k5r * y[3]**1             #y[3]-=1; y
        [2]+=1; y[10]+=1;
    prop[12] = k6 * y[2]**1 * y[10]**1   #y[5]+=1; y
        [2]-=1; y[10]-=1;
    prop[13] = k6r * y[5]**1             #y[5]-=1; y
        [2]+=1; y[10]+=1;
    prop[14] = k7 * y[2]**1 * y[10]**1   #y[6]+=1; y
        [2]-=1; y[10]-=1;
    prop[15] = k7r * y[6]**1             #y[6]-=1; y
        [2]+=1; y[10]+=1;
    prop[16] = k8 * y[3]**1 * y[10]**1   #y[7]+=1; y
        [3]-=1; y[10]-=1;
    prop[17] = k8r * y[7]**1             #y[7]-=1; y
        [3]+=1; y[10]+=1;
    prop[18] = k9 * y[3]**1 * y[10]**1   #y[8]+=1; y
        [3]-=1; y[10]-=1;
    prop[19] = k9r * y[8]**1             #y[8]-=1; y
        [3]+=1; y[10]+=1;
    prop[20] = k10 * y[4]**1 * y[10]**1  #y[7]+=1; y
        [4]-=1; y[10]-=1;
    prop[21] = k10r * y[7]**1            #y[7]-=1; y
        [4]+=1; y[10]+=1;
    prop[22] = k11 * y[5]**1 * y[10]**1  #y[7]+=1; y
        [5]-=1; y[10]-=1;
    prop[23] = k11r * y[7]**1            #y[7]-=1; y
        [5]+=1; y[10]+=1;
    prop[24] = k12 * y[5]**1 * y[10]**1  #y[8]+=1; y
        [5]-=1; y[10]-=1;
    prop[25] = k12r * y[8]**1            #y[8]-=1; y
```

```
                [5]+=1; y[10]+=1;
            prop[26] = k13 * y[6]**1 * y[10]**1        #y[8]+=1; y
                [6]-=1; y[10]-=1;
            prop[27] = k13r * y[8]**1                  #y[8]-=1; y
                [6]+=1; y[10]+=1;
            prop[28] = k14 * y[7]**1 * y[10]**1        #y[9]+=1; y
                [7]-=1; y[10]-=1;
            prop[29] = k14r * y[9]**1                  #y[9]-=1; y
                [7]+=1; y[10]+=1;
            prop[30] = k15 * y[8]**1 * y[10]**1        #y[9]+=1; y
                [8]-=1; y[10]-=1;
            prop[31] = k15r * y[9]**1                  #y[9]-=1; y
                [8]+=1; y[10]+=1;



            cprop=sp.cumsum(prop)
            r1=npr.random()
            tau = 1/cprop[-1]*sp.log(1/r1)
            #print "time info",tau, cprop[-1],r1
            t=tau+t #imepoints[cnt]
            #print t,tau
            while t>=timepoints[cnt] and cnt<steps:
                Y= (y[5]+y[6]+y[7]+y[8])/(4.0)
                #print timepoints[cnt],
                for cntb in range(len(y)):#,timepoints
                    #print y[cntb],
                    pass
                #print "\n",
                cnt+=1
            if (t < tend):
                r2=npr.random()
                if   r2<=cprop[0]/cprop[-1]:
#                    print "0"
                    y[1] +=1
                    y[0] -=1
                    y[10]-=1
                elif   r2<=cprop[1]/cprop[-1]:
#                    print "1"
                    y[1] -=1
                    y[0] +=1
                    y[10]+=1
                elif r2<=cprop[2]/cprop[-1]:
#                    print "2"
                    y[2]+=1
                    y[0]-=1
                    y[10]-=1;
```

```
155                 elif r2<=cprop[3]/cprop[-1]:
#                       print "3"
                        y[2]-=1
                        y[0]+=1
                        y[10]+=1
160                 elif r2<=cprop[4]/cprop[-1]:
#                       print "4"
                        y[4]+=1
                        y[1]-=1
                        y[10]-=1
165                 elif r2<=cprop[5]/cprop[-1]:
#                       print "5"
                        y[4]-=1
                        y[1]+=1
                        y[10]+=1
170                 elif r2<=cprop[6]/cprop[-1]:
#                       print "6"
                        y[3]+=1
                        y[1]-=1
                        y[10]-=1
175                 elif r2<=cprop[7]/cprop[-1]:
#                       print "7"
                        y[3]-=1
                        y[1]+=1
                        y[10]+=1
180                 elif r2<=cprop[8]/cprop[-1]:
#                       print "8"
                        y[5]+=1
                        y[1]-=1
                        y[10]-=1
185                 elif r2<=cprop[9]/cprop[-1]:
#                       print "9"
                        y[5]-=1
                        y[1]+=1
                        y[10]+=1
190                 elif r2<=cprop[10]/cprop[-1]:
#                       print "10"
                        y[3]+=1
                        y[2]-=1
                        y[10]-=1
195                 elif r2<=cprop[11]/cprop[-1]:
#                       print "11"
                        y[3]-=1
                        y[2]+=1
                        y[10]+=1
200                 elif r2<=cprop[12]/cprop[-1]:
#                       print "12"
```

```
                    y[5]+=1
                    y[2]-=1
                    y[10]-=1
205             elif r2<=cprop[13]/cprop[-1]:
#                    print "13"
                    y[5]-=1
                    y[2]+=1
                    y[10]+=1
210             elif r2<=cprop[14]/cprop[-1]:
#                    print "14"
                    y[6]+=1
                    y[2]-=1
                    y[10]-=1
215             elif r2<=cprop[15]/cprop[-1]:
#                    print "15"
                    y[6]-=1
                    y[2]+=1
                    y[10]+=1
220             elif r2<=cprop[16]/cprop[-1]:
#                    print "16"
                    y[7]+=1
                    y[3]-=1
                    y[10]-=1
225             elif r2<=cprop[17]/cprop[-1]:
#                    print "17"
                    y[7]-=1
                    y[3]+=1
                    y[10]+=1
230             elif r2<=cprop[18]/cprop[-1]:
#                    print "18"
                    y[8]+=1
                    y[3]-=1
                    y[10]-=1
235             elif r2<=cprop[19]/cprop[-1]:
#                    print "19"
                    y[8]-=1
                    y[3]+=1
                    y[10]+=1
240             elif r2<=cprop[20]/cprop[-1]:
#                    print "20"
                    y[7]+=1
                    y[4]-=1
                    y[10]-=1
245             elif r2<=cprop[21]/cprop[-1]:
#                    print "21"
                    y[7]-=1
                    y[4]+=1
```

```
                    y[10]+=1
250                 elif r2<=cprop[22]/cprop[-1]:
#                       print "22"
                    y[7]+=1
                    y[5]-=1
                    y[10]-=1
255                 elif r2<=cprop[23]/cprop[-1]:
#                       print "23"
                    y[7]-=1
                    y[5]+=1
                    y[10]+=1
260                 elif r2<=cprop[24]/cprop[-1]:
#                       print "24"
                    y[8]+=1
                    y[5]-=1
                    y[10]-=1
265                 elif r2<=cprop[25]/cprop[-1]:
#                       print "25"
                    y[8]-=1
                    y[5]+=1
                    y[10]+=1
270                 elif r2<=cprop[26]/cprop[-1]:
#                       print "26"
                    y[8]+=1
                    y[6]-=1
                    y[10]-=1
275                 elif r2<=cprop[27]/cprop[-1]:
#                       print "27"
                    y[8]-=1
                    y[6]+=1
                    y[10]+=1
280                 elif r2<=cprop[28]/cprop[-1]:
#                       print "28"
                    y[9]+=1
                    y[7]-=1
                    y[10]-=1
285                 elif r2<=cprop[29]/cprop[-1]:
#                       print "29"
                    y[9]-=1
                    y[7]+=1
                    y[10]+=1
290                 elif r2<=cprop[30]/cprop[-1]:
#                       print "30"
                    y[9]+=1
                    y[8]-=1
                    y[10]-=1
295                 elif r2<=cprop[31]/cprop[-1]:
```

```
#                    print "31"
                    y[9]-=1
                    y[8]+=1
                    y[10]+=1
            else:
                datapoints[x]=(O2-y[10])/(size*4.)

#                print datapoints,O2,y[10],size*4
#    print "data",datapoints,O2,y
    return [timepoints,datapoints]



def plot_ssa_ackers(v,x,Hb):
    retval=[]
    normx=sp.copy(x)
    vol = 1/(Hb*6.023e23)
    for amt in normx:
        startO2=float(round(float(amt)/(Hb*4)  *size))
        startO2 = round(amt*6.023e23*vol*size)
        #print "startO2,Hb, amt,vol",startO2,Hb, amt,vol
        retval.append(ackersssa(v,startO2))
 #   print x
 #   print retval


    return(retval)

def ackersssa(v,O):
    t=15
    n=250
    test = hemossa_alt(t,20,n,O,v[0],v[1],v[2],v[3])
    values=test[1]
    #print "values",
    #print values
    #print "ret",
    #print sp.mean(values)
    return sp.mean(values)

def stochKon(rate,concHeme):
    """ Function doc """
    nA=6.023e23
    molecPerLiter=concHeme*nA*4.
    literPerMolec=1/molecPerLiter
    stochRate=rate/(literPerMolec*nA)/4.
    print "**********rate liter kon", rate,literPerMolec,
       stochRate
```

```
    return stochRate


if __name__=="__main__":
    argv=sys.argv
    print argv
    try:
        read_data=np.loadtxt(sys.argv[1])
    except IOError:
        print "Error!!!  Could not open file: %s" %argv[1]
        sys.exit(-1)
    xall = read_data[:,1]
    yall = read_data[:,0]
#     xall= sp.array(xall*yall[yall.argmax()])

#
    yall2=sp.array(yall)
    #print xall
    #print argv[2],1/float(argv[6])
    # sys.exit(0)
    # yall=sp.array(yall)
    # e=lambda v, x, y,Hb:(((fitackersB(v,x,Hb)-y)**2)**.5).sum()
    try:
        kon   = float(argv[2])
        mult1 = float(argv[3])
        mult2 = float(argv[4])
        mult3 = float(argv[5])

        Hb    = float(argv[6])/4

    except ValueError,msg:
        print "Error reading the arguments:\n\tkon= %s\n\
           tmultiplyer1= %s\n\tmultiplyer2 =%s\n\tmultiplyer3 = %
           s" % (argv[2], argv[3], argv[4], argv[5])
        print "Error message: ", msg
        sys.exit(-1)



    kon = stochKon(kon,Hb)
    print "kon",  kon
    v0=[kon ,   mult1 ,  mult2  ,  mult3]
    #yall2=sp.array(yall/yall[yall.argmax()])
    ytoo = plot_ssa_ackers(v0,xall,Hb)
    ytoo = sp.array(ytoo)
    print "kon = ", kon
```

```
    for (x,y,z) in zip(xall, yall, ytoo):
        print x,y,z
    pl.plot(xall,yall2,"r+",xall,ytoo)
390
    pl.savefig(argv[1].split(".")[0]+"-fitnorm-
       randforrealSTOCH_more-"+str(kon)+".png")
    pl.show()
```

Algorithms and Source Code 8: A modified stochastic model of Hb/O_2 dynamics as presented by Ackers

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import numpy as np
import numpy.random as npr
import scipy as sp
import scipy.integrate as spi
import scipy.optimize as spo
import pylab as pl
import sys

def ackersrre(y,t,karray):

    koff=11
    kon=karray[0]
    kon1=karray[0]*karray[1]
    kon2=karray[0]*karray[2]
    kon3=karray[0]*karray[3]

    yprime = sp.zeros(11)
    yprime[0] = -2*kon * y[0] * y[10] + koff * y[1] -2*kon * y[0]
        * y[10] + koff * y[2]

    yprime[1] = +2*kon * y[0] * y[10] - koff * y[1] -kon1 * y[1]
       * y[10]\
              + koff * y[3] -kon * y[1] * y[10] + koff * y[4]\
               -kon * y[1] * y[10] + koff * y[5]

    yprime[2] = +2*kon * y[0] * y[10] - koff * y[2] -kon1 * y[2]
       * y[10]\
              + koff * y[3]-kon * y[2] * y[10] + koff * y[5]\
              -kon * y[2] * y[10] + koff * y[6]
    yprime[3] = +kon1 * y[1] * y[10] -koff * y[3]+kon1 * y[2] * y
       [10]\
              -koff * y[3]-kon1 * y[3] * y[10] + koff * y[7]\
              -kon1 * y[3] * y[10] + koff * y[8]
    yprime[4] = +kon * y[1] * y[10] - koff * y[4]-kon2 * y[4] * y
       [10]\
              + koff *  y[7]
    yprime[5] = +kon * y[1] * y[10] - koff * y[5]+kon * y[2] * y[
       10]\
              - koff * y[5]-kon2 * y[5] * y[10] + koff * y[7] \
              - kon2 * y[5] * y[10] + koff * y[8]
    yprime[6] = +kon * y[2] * y[10] -koff * y[6]-kon2 * y[6] * y[
       10]\
              + koff * y[8]
```

```
    yprime[7] = +kon1 * y[3] * y[10] - koff * y[7]+kon2 * y[4] *
       y[10]\
                - koff * y[7]+kon2 * y[5] * y[10] - koff * y[7]\
                -kon3 * y[7] * y[10] + koff * y[9]*2
    yprime[8] = +kon1 * y[3] * y[10] - koff * y[8]+kon2 * y[5] *
       y[10]\
                - koff * y[8]+kon2 * y[6] * y[10] - koff * y[8]\
                -kon3 * y[8] * y[10] + koff * y[9]*2
    yprime[9] = + kon3 * y[7] * y[10] - koff * y[9]*2+kon3 * y[8]
        * y[10]\
                - koff * y[9]*2
    yprime[10] = -2*kon * y[0] * y[10] + koff *y[1]-2*kon * y[0]
       * y[10]\
                + koff *y[2]-kon * y[1] * y[10] + koff *y[4]-kon
                   *y[1]* y[10]\
                + koff *y[5]-kon1 * y[1] * y[10] + koff *y[3]-
                   kon1*y[2]*y[10]\
                + koff *y[3]-kon * y[2] * y[10] + koff * y[5]-kon
                   *y[2]*y[10]\
                + koff * y[6]-kon1 * y[3] * y[10] + koff * y[7]\
                -kon1 * y[3] * y[10] + koff * y[8]-kon2 * y[4] *
                   y[10]\
                 + koff * y[7]-kon2 * y[5] * y[10] + koff * y[7]
                    \
                -kon2 * y[5] * y[10] + koff * y[8]-kon2 * y[6] *
                   y[10]\
                 + koff * y[8]-kon3 * y[7] * y[10] + 2*koff * y[9
                    ]\
                -kon3 * y[8] * y[10] + 2*koff * y[9]
    return yprime

def hemossa(tend,steps,trials,O2,kon,mult1,mult2,mult3):
    timepoints=np.arange(steps+1)*(tend/steps)
    datapoints=np.zeros(steps+1)
    for x in xrange(int(trials)):
        vol=1e-10
        nA=6.023e23
        prop = sp.zeros(8)
        y=sp.zeros(9)
        t=0
        cnt=0

        koff=11

        y[0]=O2
        y[1]=1
        y[2]=1
```

```
        y[3]=1
        y[4]=1
        y[5]=0
        y[6]=0
        y[7]=0
        y[8]=0

        while tend>t:
            """Determine the Specific Ackers
            "hemoglobin type that the complex represents"""
            # kon=5e2/(nA*vol)
            # mult1=0.9
            #  mult2=1.5
            #  mult3=550.0
            #
            koff=11
            if sum(y[1:5])==4:
                """Ackers Hb ~ 10"""
                kon1=kon
                kon2=kon
                kon3=kon
                kon4=kon
                koff1 = koff
                koff2 = koff
                koff3 = koff
                koff4 = koff
            elif sum(y[1:5])==3:
                if y[5]==1:
                    """ one alpha is bound to an oxygen """
                    kon1 = 0
                    kon2 = mult1*kon
                    kon3 = kon
                    kon4 = kon
                    koff1= 0
                    koff2 = 0
                    koff3 = 0
                    koff4 = 0
                if y[6]==1:
                    """ one alpha is bound to an oxygen """
                    kon1 = mult1*kon
                    kon2 = 0
                    kon3 = kon
                    kon4 = kon
                    koff1 = koff
                    koff2 = koff
                    koff3 = koff
                    koff4 = koff
```

```
                if y[7]==1:
                    """ one alpha is bound to an oxygen """
                    kon1 = kon
                    kon2 = kon
                    kon3 = 0
                    kon4 = mult1*kon
                    koff1 = koff
                    koff2 = koff
                    koff3 = koff
                    koff4 = koff
                if y[8]==1:
                    """ one alpha is bound to an oxygen """
                    kon1 = kon
                    kon2 = kon
                    kon3 = mult1*kon
                    kon4 = 0
                    koff1 = koff
                    koff2 = koff
                    koff3 = koff
                    koff4 = koff
            elif sum(y[1:5])==2:
                if y[1]==1 and y[2]==1:
                    """ one beta is bound to an oxygen """
                    kon1 = mult1*kon
                    kon2 = mult1*kon
                    kon3 = mult1*kon
                    kon4 = mult1*kon
                    koff1 = koff
                    koff2 = koff
                    koff3 = koff
                    koff4 = koff
                if y[1]==1 and y[3]==1:
                    """ one beta is bound to an oxygen """
                    kon1 = mult2*kon
                    kon2 = mult2*kon
                    kon3 = mult2*kon
                    kon4 = mult2*kon
                    koff1 = koff
                    koff2 = koff
                    koff3 = koff
                    koff4 = koff
                if y[1]==1 and y[4]==1:
                    """ one beta is bound to an oxygen """
                    kon1 = mult2*kon
                    kon2 = mult2*kon
                    kon3 = mult2*kon
                    kon4 = mult2*kon
```

```
                    koff1 = koff
                    koff2 = koff
                    koff3 = koff
                    koff4 = koff
            if y[2]==1 and y[3]==1:
                """ one beta is bound to an oxygen """
                kon1 = mult2*kon
                kon2 = mult2*kon
                kon3 = mult2*kon
                kon4 = mult2*kon
                koff1 = koff
                koff2 = koff
                koff3 = koff
                koff4 = koff
            if y[2]==1 and y[4]==1:
                """ one beta is bound to an oxygen """
                kon1 = mult2*kon
                kon2 = mult2*kon
                kon3 = mult2*kon
                kon4 = mult2*kon
                koff1 = koff
                koff2 = koff
                koff3 = koff
                koff4 = koff
            if y[3]==1 and y[4]==1:
                """ one beta is bound to an oxygen """
                kon1 = mult1*kon
                kon2 = mult1*kon
                kon3 = mult1*kon
                kon4 = mult1*kon
                koff1 = koff
                koff2 = koff
                koff3 = koff
                koff4 = koff
        elif sum(y[1:5])==1:
            if y[1]==1:
                """ one beta is bound to an oxygen """
                kon1 = mult3*kon
                kon2 = 0
                kon3 = 0
                kon4 = 0
                koff1 = koff
                koff2 = koff
                koff3 = koff
                koff4 = koff
            if y[2]==1:
                """ one beta is bound to an oxygen """
```

```
                    kon1 = 0
                    kon2 = mult3*kon
                    kon3 = 0
                    kon4 = 0
                    koff1 = koff
                    koff2 = koff
                    koff3 = koff
                    koff4 = koff
                if y[3]==1:
                    """ one beta is bound to an oxygen """
                    kon1 = 0
                    kon2 = 0
                    kon3 = mult3*kon
                    kon4 = 0
                    koff1 = koff
                    koff2 = koff
                    koff3 = koff
                    koff4 = koff
                if y[4]==1:
                    """ one beta is bound to an oxygen """
                    kon1 = 0
                    kon2 = 0
                    kon3 = 0
                    kon4 = mult3*kon
                    koff1 = koff
                    koff2 = koff
                    koff3 = koff
                    koff4 = koff
            prop[0]=kon1 * y[0] * y[1]
            prop[1]=kon2 * y[0] * y[2]
            prop[2]=kon3 * y[0] * y[3]
            prop[3]=kon4 * y[0] * y[4]
            prop[4]=koff * y[5]
            prop[5]=koff * y[6]
            prop[6]=koff * y[7]
            prop[7]=koff * y[8]
            cprop=sp.cumsum(prop)
            r1=npr.random()
            tau = 1/cprop[-1]*sp.log(1/r1)
            t=tau+t
            while t>=cnt*timepoints[1] and cnt<steps+1:
                Y= (y[5]+y[6]+y[7]+y[8])/4.0
                datapoints[cnt]+=Y
                cnt+=1
            if (t < tend):
                r2=npr.random()
                if   r2<=cprop[0]/cprop[-1]:
```

```
                    y[0]-=1
                    y[1]-=1
                    y[5]+=1
                elif r2<=cprop[1]/cprop[-1]:
                    y[0]-=1
                    y[2]-=1
                    y[6]+=1
                elif r2<=cprop[2]/cprop[-1]:
                    y[0]-=1
                    y[3]-=1
                    y[7]+=1
                elif r2<=cprop[3]/cprop[-1]:
                    y[0]-=1
                    y[4]-=1
                    y[8]+=1
                elif   r2<=cprop[4]/cprop[-1]:
                    y[0]+=1
                    y[1]+=1
                    y[5]-=1
                elif r2<=cprop[5]/cprop[-1]:
                    y[0]+=1
                    y[2]+=1
                    y[6]-=1
                elif r2<=cprop[6]/cprop[-1]:
                    y[0]+=1
                    y[3]+=1
                    y[7]-=1
                elif r2<=cprop[7]/cprop[-1]:
                    y[0]+=1
                    y[4]+=1
                    y[8]-=1
            else:
                break
    return [timepoints,datapoints/int(trials)]

def fitackersB(v,x,Hb):
    startHemo = Hb
    retval=[]
    print v
    for amt in x:
        #print amt
        yzero=pl.array([startHemo,0,0,0,0,0,0,0,0,0,float(amt)])

        #    times=pl.linspace(0.0,0.5,1e4,endpoint=True)
        times=sp.arange(0,5.0,.005)
        ray=v
        test=spi.odeint(ackersrre,yzero,times,args=(ray,))
```

```
        retval.append((amt-test[-1,10])/(startHemo*4))
#    print retval
    return sp.array(retval)
def plot_ssa_ackers(v,x,Hb):
    retval=[]
    normx=sp.copy(x)
    for amt in normx:
        startO2=round(float(amt)/Hb)
        print "looky",startO2,Hb, amt
        retval.append(ackersssa(v,startO2))
    return(retval)

def ackersssa(v,O):
    test = hemossa(6.0,4,200,O,v[0],v[1],v[2],v[3])

    values=test[1]
    print values
    ret=sp.copy(values[-1])
    print ret
    return ret
def stochKon(rate,concHeme):
    """ Function doc """
    nA=6.023e23
    molecPerLiter=concHeme*nA*4.
    literPerMolec=1/molecPerLiter
    stochRate=rate/(literPerMolec*nA)/4.
    print "**********rate liter kon", rate,literPerMolec,
       stochRate
    return stochRate

if __name__=="__main__":
#    print fitackersB([1.0, 9., 90., 550.],30)
    argv=sys.argv
    print argv
    try:
        read_data=pl.mlab.load(argv[1])
    except IOError:
        print "Error!!!  Could not open file: %s" %argv[1]
        sys.exit(-1)
    xall = read_data[:,1]

    yall = read_data[:,0]
    xall=sp.array(xall)

    yall2=sp.array(yall)
    # yall=sp.array(yall)
    e=lambda v, x, y,Hb:(((fitackersB(v,x,Hb)-y)**2)**.5).sum()
```

```
    try:
        kon    = float(argv[2])
        mult1  = float(argv[3])
        mult2  = float(argv[4])
        mult3  = float(argv[5])
        Hb     = float(argv[6])/4.
        # Hb     = xall[yall.argmax()]/4

    except ValueError,msg:
        print "Error reading the arguments:\n\tkon= %s\n\
           tmultiplyer1= %s\n\tmultiplyer2 =%s\n\tmultiplyer3 = %
           s" % (argv[2], argv[3], argv[4], argv[5])
        print "Error message: ", msg
        sys.exit(-1)
    kon = stochKon(kon,Hb)
    v0=[kon ,   mult1 ,  mult2  ,  mult3]

#    v= spo.fmin(e,v0,args=(xall,yall,Hb), ftol=1.9, xtol=1.9,
   full_output=1, maxiter=100000,maxfun=100000)
    #v= spo.fmin(e,v0,args=(xall,yall,Hb),  maxiter=100000,maxfun
       =100000)
    #print v
    #print e(v,xall,yall)
    #print yall,
    #ynew = fitackersB(v0,xall,Hb)
    #pl.plot(xall,yall2,"r+",xall,ynew)
    ytoo = plot_ssa_ackers(v0,xall,Hb)
    pl.plot(xall,yall2,"r+",xall,ytoo)

    pl.savefig(argv[1].split(".")[0]+"-fitnorm-randforrealSTOCH-"
       +str(kon)+".png")
    for x,y,z in zip (xall,yall2,ytoo):
        print x, y, z
```

Part II

Evolutionary Algorithms for the Optimization of Force-Fields in Computational Chemistry

Chapter 10

Motivation to Use Evolutionary Algorithms to Optimize *in Silico* Code

The laboratory of William A. Goddard has a long history of developing computational chemistry methods. Many of these methods, if not all of them, have relied on a combination of parameter sweeps and analytic methods, such as the conjugate gradient method, to perform parameter optimization. While the analytic methods will quickly find a minimal value in parameter-space, there is no guarantee that it will be the global minima.

We define *fitness landscape* as a multi-dimensional space, where there is an axis for each parameter and an additional axis to represent the performance/error of those parameters on a specific task and approach. Our task is the calculation of information regarding a set of chemical elements, and the approach is not only the method, but also the formulation of our evaluation of error. Parameter sweeps sample parameter-space on a grid and are, therefore, not bound by gradient methods. A major drawback to parameter sweeps is that they vary individual parameters one at a time. Thus, it may be possible to miss the global minimum, depending

on the pathway of the sweeps. Also, intuition is necessary when performing parameter sweeps; as a result, it is a somewhat biased approach.

Many of the systems that we work with have tortuous parameter-spaces, and we would like to transverse them in an intelligent, semi-automated, and thorough fashion. Population-based optimization techniques, such as the class of optimization techniques called evolutionary algorithms, provide a method to optimize our parameters in a more global fashion and to transverse parameter-space moving toward better solutions.

Over the past three years, we have begun applying evolutionary algorithms with great success to the task of optimizing our systems. Today, members of the Goddard lab have applied the codes and schemes of implementation outlined in the subsequent part of my thesis for the optimization of charge equilibration (QEq) (Rappé and Goddard [1991]), electronic force-fields (EFF) (Su and Goddard [2009]), and reactive force-fields (ReaxFF) (van Duin et al. [2001]).

Chapter 11

Introduction to Evolutionary Algorithms

More than 40 years ago, evolutionary algorithms (EAs) were shown to perform well on optimization tasks (Rechenberg [1965], Fogel et al. [1966], and Holland [1962]; for review see Jong [2007]). In the past 15 years EAs have been applied to various computational chemistry problems, including structure prediction, quantitative structure-activity relationships (QSAR), and chemometrics (Clark [2000]).

EAs are a class of meta-heuristic methods whose foundation is loosely based on the phenomena of genetic transmission of information. Parameters can be thought of as "genes" containing various values, or "alleles". A vector of parameters, or genes, is, therefore, a candidate solution that is analogous to a "chromosome". Each set of chromosomes is a "population", and each population after the first is derived in some manner from previous populations. Populations have an order of evolution that is analogous to "generations". As a reference point, if each population derives no information from previous generations, that is a simple Monte Carlo or random search.

11.1 A Brief History of Evolutionary Algorithm Methodologies

Holland [1962] popularized genetic algorithms, a form of evolutionary algorithms, in the United States. At about the same time, Fogel [1966] was developing evolutionary programming in Southern California, and Rechenberg [1965] was developing evolutionary strategies in Germany. Somewhat later, Koza [1991] made genetic programming popular.

Easily differentiating between these methods as they exist in 2011 is something of a fool's errand. We can discuss the features of genetic algorithms and explain the choices that we took in the optimization of QEq. The set of general features of evolutionary algorithms includes the chromosome, a population, one generation and the process in which the population in a generation leads to the next through some operation of simulated evolution.

11.2 The Chromosome

In the early days of genetic algorithms, the chromosome was represented as a vector of ones and zeros, or by binary encoding. A simple chromosome containing two genes is presented in Table 11.2.

Real-valued encoding is another type of encoding where the alleles, or the values of the genes, are contained within a single element in the chromosome. We show the real-valued example of a chromosome with four genes in Table 11.2.

chromosome Ident.	gene 1				gene 2			
I.	1	0	1	0	1	1	1	0

Table 11.1: A binary chromosome. All elements of the chromosome have a value of zero or one. The notion of a gene is derived from multiple elements of this chromosome in the same fashion that a gene on a chromosome is a construct of a string of nucleic acids.

chromosome Ident	gene 1	gene2	gene3	gene4
I.	10	5.3	-3000.1	11.9

Table 11.2: A real-valued chromosome. The values of the genes are contained within a single element in the chromosome. The notion of a gene is derived from multiple elements of this chromosome in the same fashion that a gene on a chromosome is a construct of a string of nucleic acids.

11.3 The Population and Generations

In evolutionary algorithms, populations have been implemented in a variety of ways. If a chromosome can be thought of as a vector of numbers, a population can be thought of as a two-dimensional array, or a vector of vectors. The population in evolutionary algorithms has varied in size from one into the hundreds, if not greater. The idea of parents and children relate to the idea of a population, but have been implemented in different ways. This will be discussed further when we talk about updating methods. Generations are snapshots of a population at some point in an optimizations trajectory.

11.4 Evolving a Population

There are two methods by which a population of chromosomes evolve. The simplest form is mutation, where a value is altered with some probability that is

the mutation frequency. In real-valued chromosomes, there is also the concept of mutation severity, the degree by which a value can be changed. Additionally, there are many methods of mutation. Values can be mutated in a bounded, uniform manner, or by using some other function to vary the parameter, such as a Gaussian function.

11.5 Updating, Crossover

We do not employ crossover in this work, but it is the sharing or exchange of chromosomal information in order to create a child.

Chapter 12

Evolutionary Algorithm Optimization of Charge Equilibration

One of our long-term goals in the Goddard laboratory has been to develop fast methods to describe atomic charge at quantum-mechanical-level accuracy. In addition to improving on the structure of fast algorithms, we have also begun to search for more accurate parameters to represent atomic electronegativity and hardness. In this endeavor, we would like to develop our physical intuition, as well as better models. In this chapter, we describe a general method employing EAs that we have applied to parameter optimization of our Gaussian-based charge equilibration (gQEq) method (Rappé and Goddard [1991], Sefcik et al. [2002]). We also present our QEq parameters for LAMMPS / QEq models of biological systems (see figure 12.1). We use PGAPack[1996] for the underlying Evolutionary algorithms.

12.1 Optimization Constraints

We develop our methods with the following guidelines in mind:

1. The atomic radii should be scaled in a uniform manner, if at all.

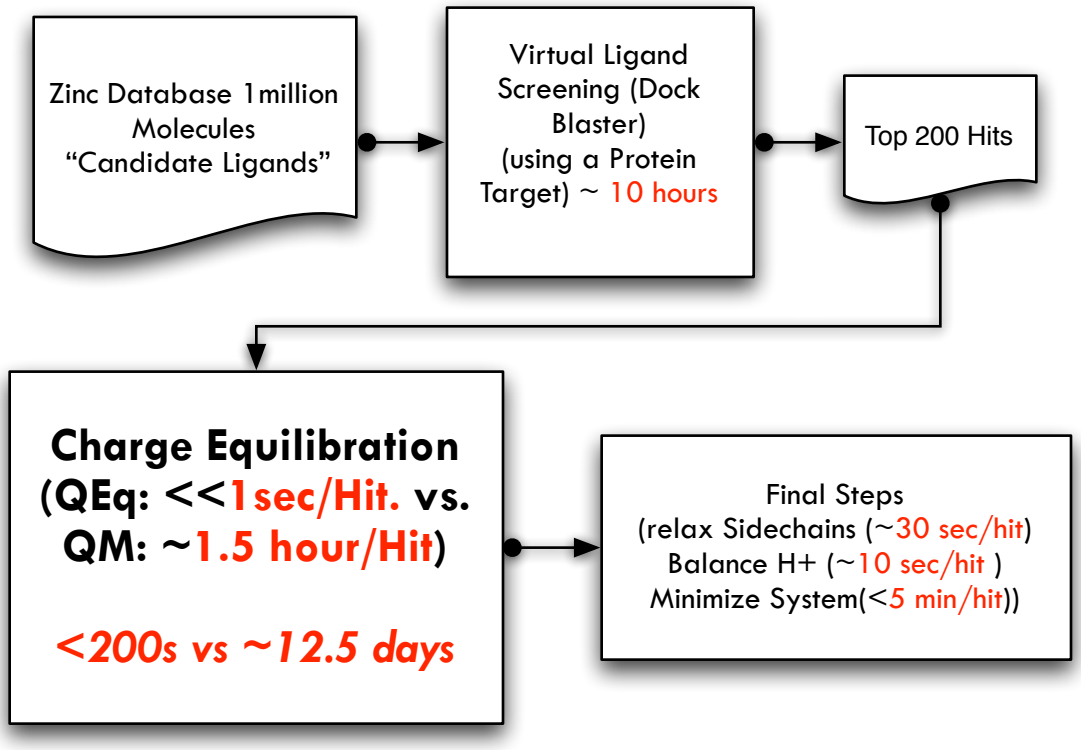


Figure 12.1: Gameplan

2. The electronegativity and hardness values may change individually.

3. A caveat of guideline (2) is that we would like to maintain certain orderings (i.e., $X_F > X_{Cl} > X_{Br}$) without explicitly forcing that order, if possible.

4. We would like to adapt the rate of mutation of any given gene (i.e., electronegativity or hardness) as a function of the diversity of our set of "alleles" for that gene such that we can narrow our search as we approach more optimal values.

We achieve guideline (1) by assigning a single gene to scale the radii uniformly. Guideline (2) is achieved by assigning normalized representations to individual electronegativities and hardnesses, representing genes. These normalized representations are scaling parameters (Table 12.1) for the individual parameters (Table 12.2). For example, the gene representing the hardness of c (X_F) might have an allele value of 0.9. During our evaluation of fitness, this is translated to (0.9×10.874) or 9.7866, since our starting value for X_F is 10.874 (see Table 12.1). Therefore, each of our chromosomes contains two genes per atom, representing a scaling of electronegativity X and hardness J, or $n \times 2$ genes where n is the number of atoms described in our parameter set. Guidelines (3) and (4) will be discussed subsequently.

Chromosome ID	I	II	III	IV	V	VI	VII	VIII	IV
1	1.0	1.25	1	1	1	1	0.9	1	1
2	0.9	0.8	1.2	1.5	0.7	0.99	0.7	1.1	0.85
3	1.01	1.0	1	1	1	1	0.9	1	1.31

Table 12.1: An example of our population of real-valued chromosomes whose value represents a scaling rather than a specific value

12.2 Initialization

We encode our genes with scaling factors that will operate on our previously reported parameters (Rappé and Goddard [1991]). We initially allow our population of chromosomes to deviate from the original values by up to +/- 10%.

In row 1 of Table 12.1, we present an example chromosome, a parameter set, which uses the initial parameters for all elements, except for parameters II and VII that are mutated to having values of 125% and 90% of their initial values, respectively. Our set of initial values is reprinted in Table 12.2 for the convenience of the reader.

12.3 Evaluation

We evaluate the performance of our chromosomes as candidate solutions by applying a metric of fitness. We define our metric of fitness to be the standard deviation of the difference between QEq calculated charges and actual charges, derived either experimentally or from quantum mechanical (QM) Mulliken charges

Element	X	J eV	R0
Li	3.006	4.772	1.557
C	5.343	10.126	0.759
N	6.899	11.760	0.715
O	8.741	13.364	0.669
F	10.874	14.948	0.706
Na	2.843	4.592	2.085
Si	4.168	6.974	1.176
P	5.463	8.000	1.102
S	6.928	8.972	1.047
Cl	8.564	9.892	0.994
K	2.421	3.84	2.586
Br	7.790	8.850	1.141
Rb	2.331	3.692	2.770
I	6.822	7.524	1.333
Cs	2.183	8.972	2.984
H	4.5280	13.8904	0.371

Table 12.2: Data from Rappé & Goddard [1991]

(Equation 12.1).

$$\sqrt{\frac{\sum_A \Delta q_A^2}{\#_{atoms}}} \tag{12.1}$$

where A is a given atom, Δq_A^2 is the squared difference in charge between the calculated charge for atom A and the experimental or QM calculated value, and $\#_{atoms}$ is the number of atoms in all of the structures that we are considering.

12.4 Evolving Our Population

Using the fitness metric as a guideline, the next set of candidate solutions is generated by culling poor performers and replacing them with "mutated" versions of better candidate solutions. Mutation and asexual reproduction are achieved by

stochastically varying chromosome values. We keep the top 20% of our population each generation and asexually reproduce the remaining 80% from the top 20%. We practice "elitism" in that the very best performer is allowed to pass on an unaltered copy of itself.

The chromosomes are converted to parameter files for QEq. These parameter files are used to calculate charges on the same set of molecules previously reported (Rappé and Goddard [1991]). Next, we apply our metric of fitness to determine the performance of each chromosome of the new generation.

12.5 Adaptive Mutation Severity

Our mutation is implemented in a nonstandard form. We wish to focus our mutations around current values, so we generate random values from a normal distribution that uses the current value as a mean for the distribution. Furthermore, we wish for our severity of mutation to adapt with the simulation, meaning that as we are more certain of our range of values, we want our mutations to sample from a tighter distribution. To do this, we define a subset of the population, the top 10% of performers, as our control set. We evaluate the standard deviation of a population for each value of our optimization set and use those values as the basis of our variances. This allows for our optimization to account for not only the current value of individual parameters, but also their relative performance.

12.6 Mutation Frequency

For our inorganic set of molecules, we use a mutation frequency of 100%. For our organic set of molecules, we use a mutation frequency of 25%. Using a mutation rate of 100% for the organic set of molecules delays its convergence.

12.7 Training Sets

We iterate through our training phase many times. We initially train on inorganic molecules from a test set from Rappé and Goddard and extend with data taken from the National Institute of Standards and Technology (NIST) Computational Chemistry Comparison and Benchmark Database (CCCBDB) (Table 12.3). There are three parameters that QEq uses to describe each atom: electronegativity, hardness, and atomic radius.

We train this set in two phases: initially, we optimize only the atomic radii, and then we optimize the individual electronegativites and hardnesses for Cs, H, K, Na, Rb, and Li. We make the initial assumption that the atomic radii will likely be different when using Gaussian functionals, rather than Slater functionals, and that they should scale linearly. With this assumption, we perform a simple one-dimensional parameter sweep to determine the scaling factor that yields the minimal error between the QEq calculation and experimentally derived charges. We perform this sweep on our initial set of diatomic molecules found in Table 12.3 and find that the Gaussian radii should be approximately 1.46 times that of the Slater

	H	Cs	K	Na	Rb	Li	Cl	F	I
Br	HBr	BrCs	BrK	NaBr	RbBr	LiBr	BrCl	BrF	ICl
Cl	HCl	CsCl	KCl	NaCl	RbCl	LiCl		ClF	
F	HF	CsF	KF	NaF	RbF	LiF			
I	HI	CsI	KI	NaI	RbI	LiI			
Li	HLi								

Table 12.3: Initial training cases

radii, as seen in Figure 13.1.

Our second set of molecules consists of organic molecules that are representative of a diverse set of ligands of interest to drug designers. While the first set serves as a proof-of-principle that we can optimize against experimental values of charge, this set consists totally of calculated Mulliken charges. We use the initial set of molecules to provide a general starting point for the optimization of our second set of molecules.

12.8 Final Considerations

We test the performance of our optimization against the set of molecules from "A Directory of Useful Decoys" (Huang, Shoichet et al.2006) that are commonly used in the benchmarking of ligand docking workflows. We use the original QEq found in Lingraf as a point-of-performance comparison. We use our original metric of performance, the sample standard deviation from Equation 12.1, as well as the

mean absolute error, displayed in Equation 12.2:

$$\frac{1}{n}\sum_{i=1}^{n}|e_i| \tag{12.2}$$

where e_i is the difference in charge between quantum mechanical calculated values and QEq calculated values, *i* is the atom ID for each molecule, and *n* is the number of atoms in each molecule.

Chapter 13

Results

13.1 Our Performance

The performance of our EA on the optimization of parameters for the inorganic set of molecules is depicted in Figure 13.2. To limit our state space, we hold the electronegativity of fluorine constant. The electronegativity of fluorine acts as a reference point that gently constrains our optimization in parameter space. With this one value held constant, optimization runs converge in several hundred generations; otherwise, they do not converge after thousands of generations. The reason for this is that the values of electronegativity and hardness are relative. This is an important consideration for further global optimization of similar systems.

To further constrain our search space, we add the following set of constraints for electronegativity: $X_F > X_{Cl} > X_{Br} > X_I$, $X_{Li} > X_{Na} > X_K > X_{Rb} > X_{Cs}$, $X_F > X_O$, and $X_{Cl} > X_S$. We constrain hardness with the matching relationships.

The performance of the optimization of the inorganic molecules can be seen in Figure 13.2. According to our performance against our metric of fitness, we optimize our system well by 150 generations. Taking a deeper look, we can analyze

Element	Lingraf QEq X, eV	Inorganic gQEq X, eV	Organic gQEq X, eV
Li	3.006	2.607	-
C	5.343	-	6.293
N	6.899	-	7.933
O	8.741	-	9.571
F	10.874	10.874	10.874
Na	2.843	2.523	-
S	6.928	-	5.690
Cl	8.564	9.530	6.193
K	2.421	1.806	-
Br	7.790	8.768	5.705
Rb	2.331	2.067	-
I	6.822	8.210	4.432
Cs	2.183	1.859	-
H	4.5280	7.764	5.193

Table 13.1: Original parameters and our optimized values for electronegativity

the evolution of individual parameters, illustrated in Figures 13.3(a) and 13.3(b). There is noticeable change in some of our parameters as far out as 400 generations, specifically X_{Na} and J_{Na}.

Following this initial stage, we apply the optimization task to the set of organic molecules, most of which are known ligands. The candidate solutions perform well over time, as shown in Figures 13.4 and 13.5. The systems converged within 150 generations to a stable performance with respect to the metric of fitness. The parameters evolve as displayed in Figures 13.6(a) and 13.6(b). Looking at the evolution of parameters as depicted in Figures 13.6(a) and 13.6(b), We see that the evolution of X_I and J_F are late to stabilize. It is unclear if J_F has completely stabilized. Increasing the representation of halogens in our training set should increase the rate of convergence of these molecules.

Element	Lingraf QEq J, eV	Inorganic gQEq J, eV	Organic gQEq J, eV
Li	4.772	2.607	-
C	10.126	-	8.750
N	11.760	-	11.283
O	13.364	-	14.807
F	14.948	13.972	20.574
Na	4.592	2.924	-
S	8.972	-	6.142
Cl	9.892	12.260	10.139
K	3.84	2.795	-
Br	8.850	11.330	6.736
Rb	3.692	2.346	-
I	7.524	10.705	7.600
Cs	8.972	2.882	-
H	13.8904	11.103	18.292

Table 13.2: Original parameters and our optimized values for hardness.

We tested out evolutionary-algorithm-optimized parameters on a novel set of ligands, the "Directory of Useful Ligands". We show our performance on two metrics — standard deviation and mean absolute error — in Figures 13.7 and 13.8, respectively. We outperformed the previous version of QEq on all members of the set.

13.2 Discussion

Our optimization yields favorable results in that we accurately represent atomic charge. However, a single element yields divergent values between data sets. While this is consistent with the idea that atom-types exhibit divergent chemical behavior as a function of their state of binding, it results in our need to either define atom types within our parameter sets or to build parameter sets that are specific

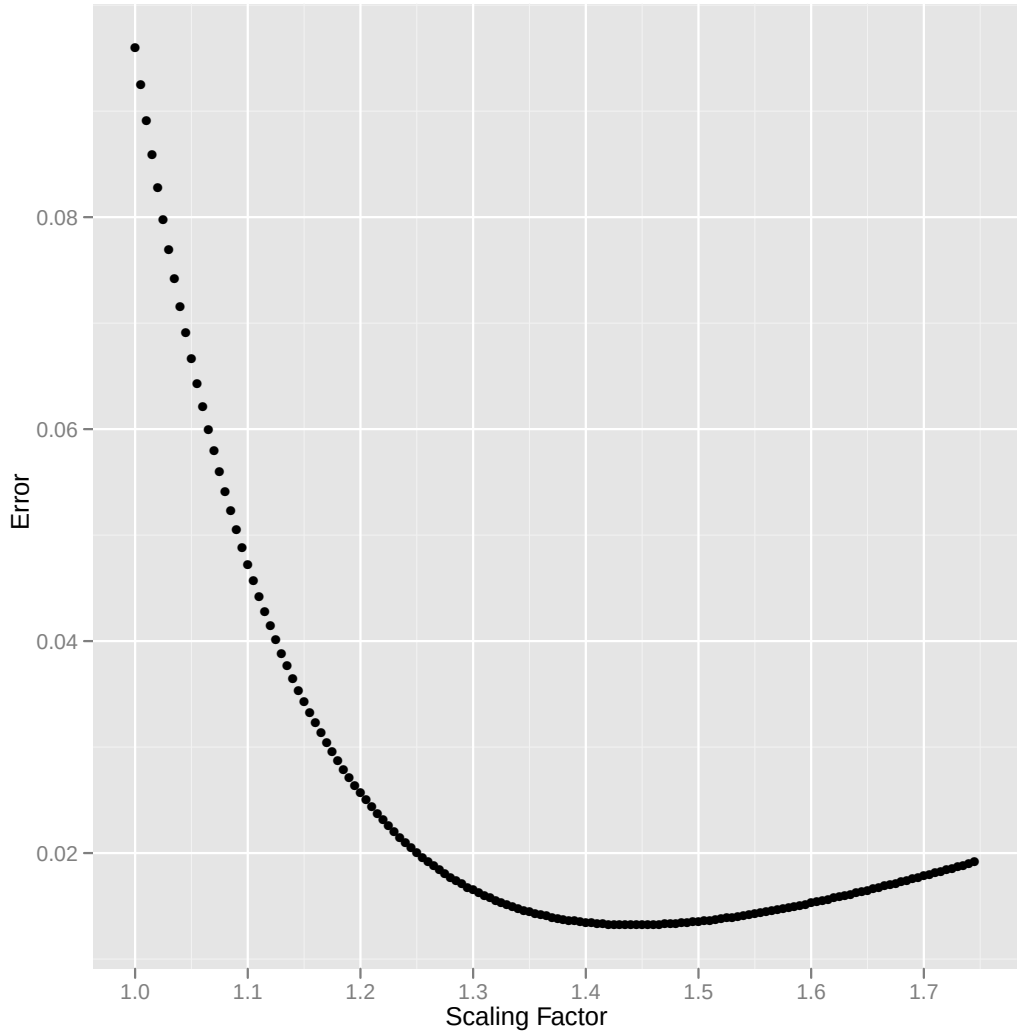


Figure 13.1: Determination of an initial scaling factor for atomic radii. The scaling factor at minimum error is approximately 1.46.

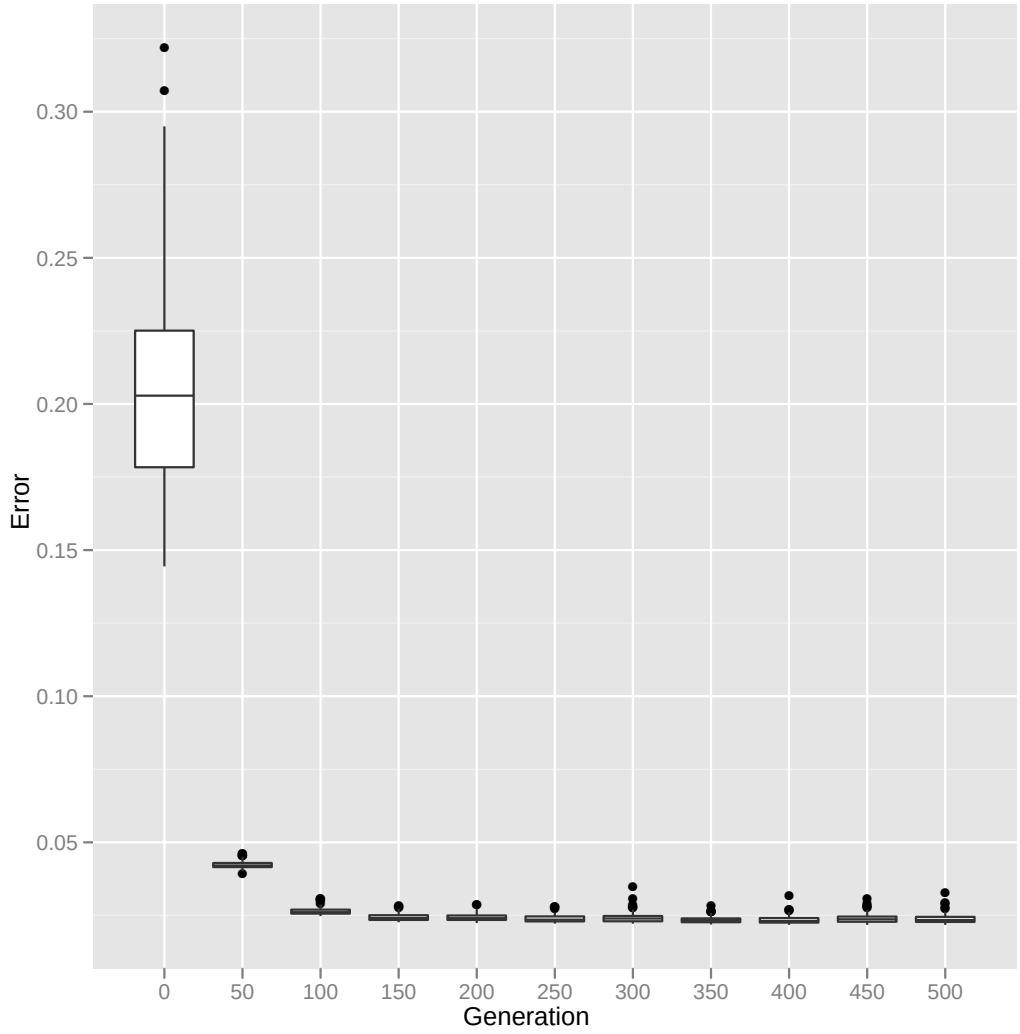
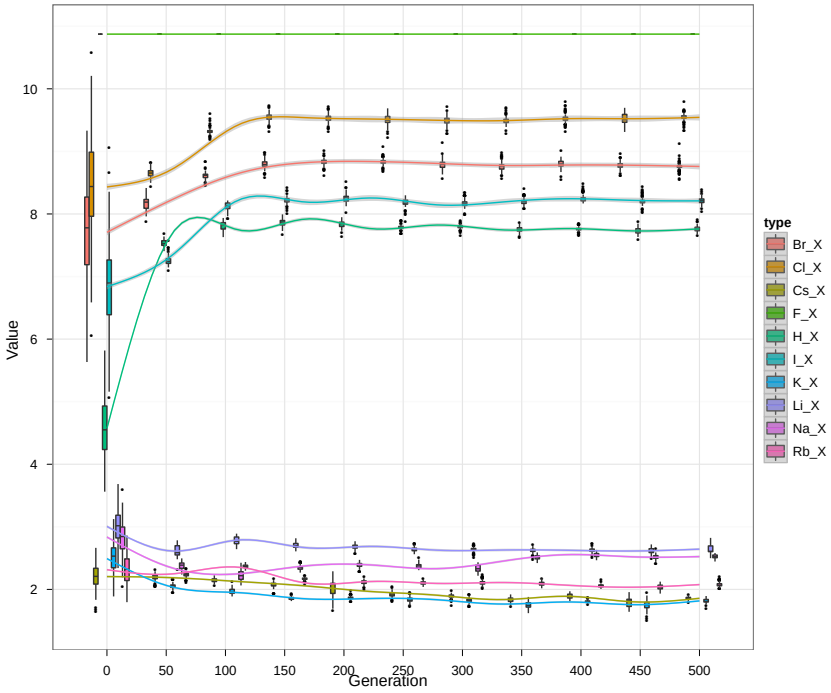
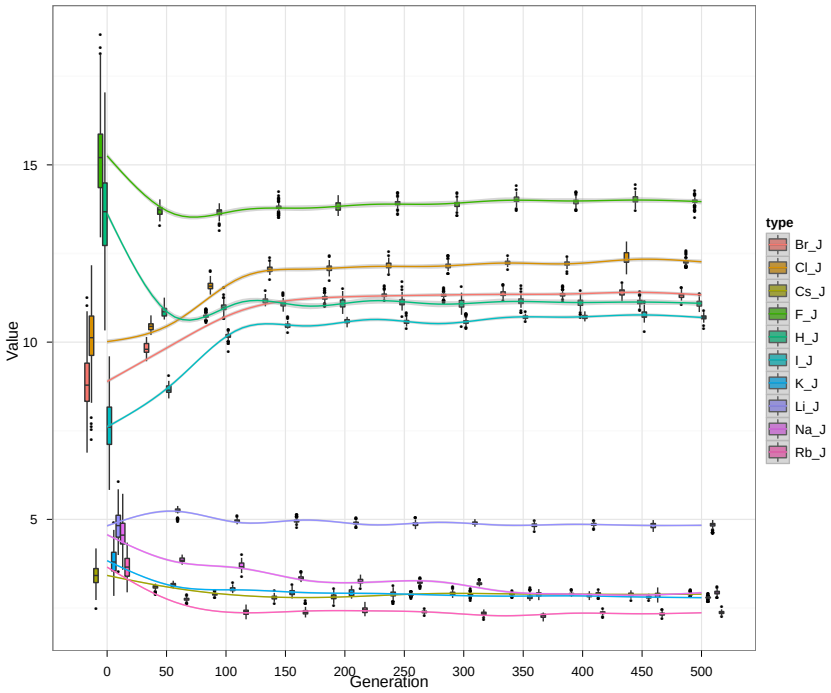


Figure 13.2: Boxplots of population performance as a function of generation for our initial set of inorganic molecules.



(a) ***Electronegativity***



(b) ***Hardness***

Figure 13.3: Optimization of our inorganic set of molecules.

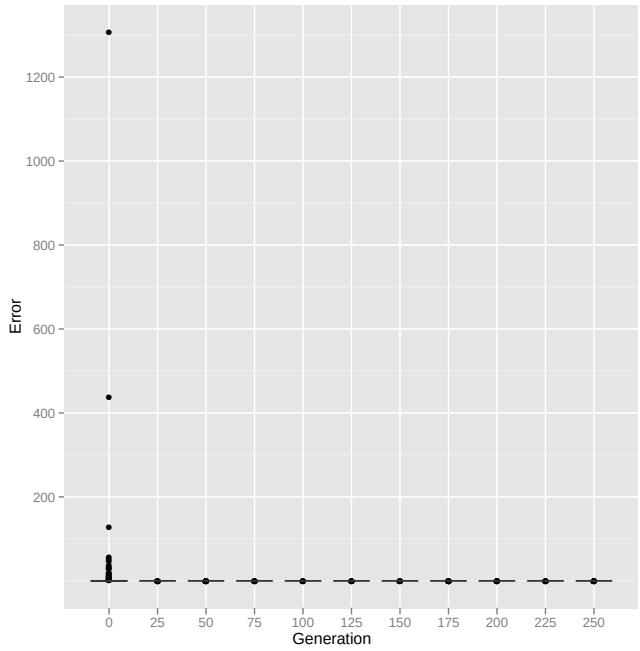


Figure 13.4: Boxplots of population performance as a function of generation for our set of organic molecules

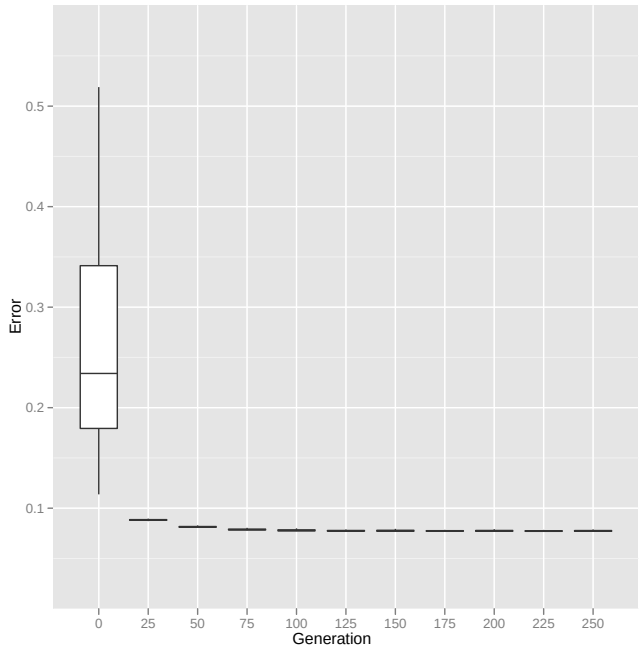
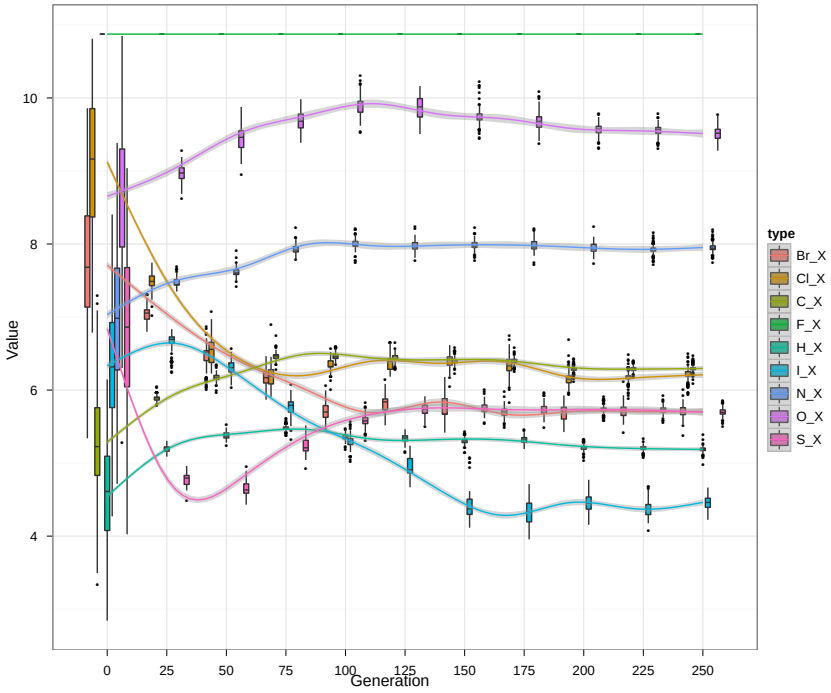
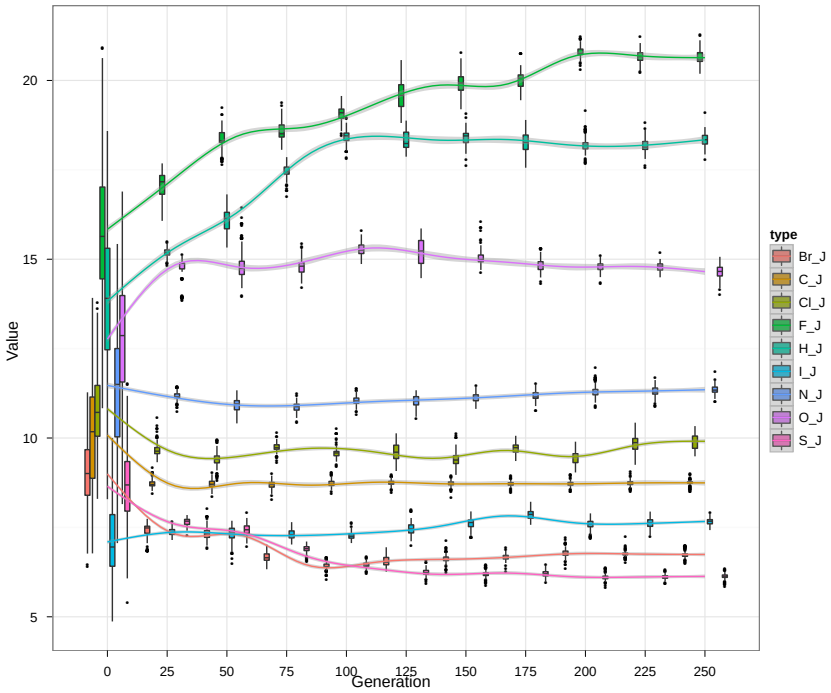


Figure 13.5: Boxplots of population performance as a function of generation for our set of organic molecules. This plot reproduces the data shown in Figure 13.4 with outliers excluded



(a) ***Electronegativity***



(b) ***Hardness***

Figure 13.6: Optimization of our organic set of molecules

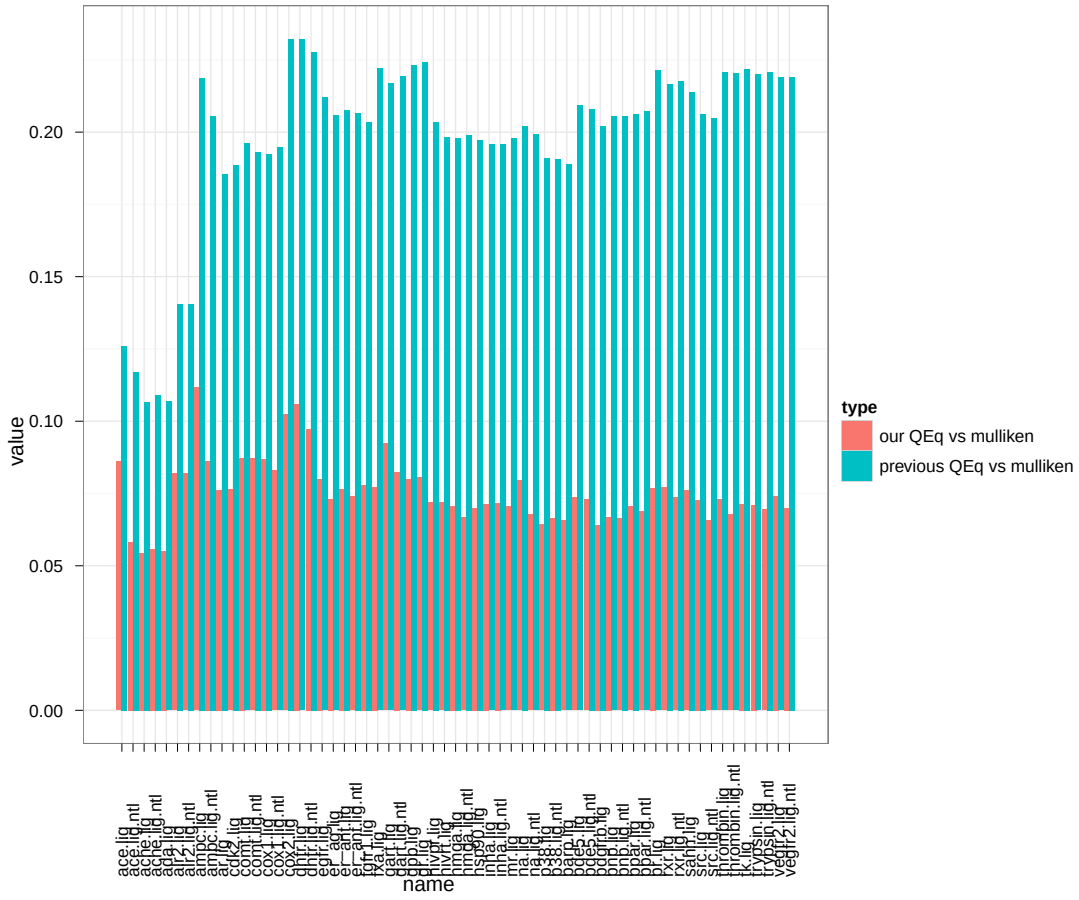


Figure 13.7: Standard deviations of EA-derived gQEq charges and quantum-mechanical-derived charges (blue) vs. standard deviations of charges from the previous model gQEq and quantum-mechanical-derived charges (red)

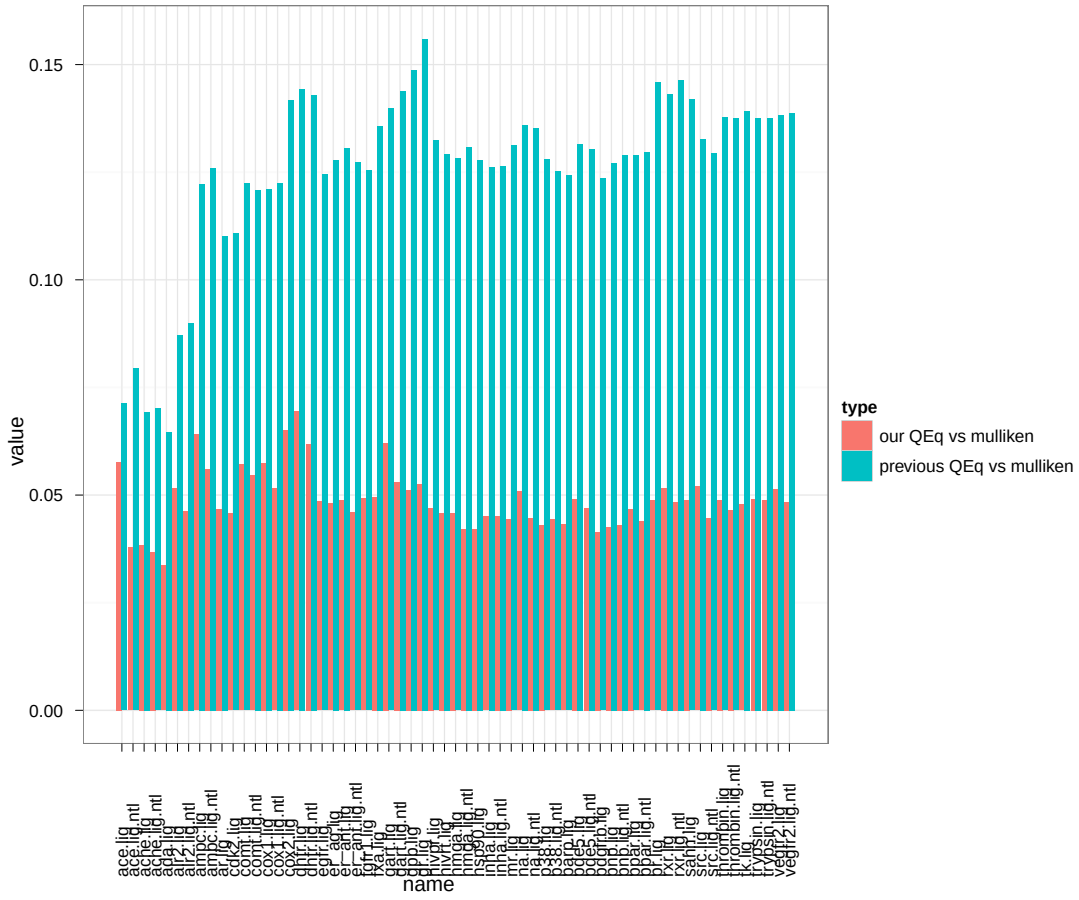


Figure 13.8: Mean absolute errors of EA-derived QEq charges and quantum-mechanical-derived charges (blue) vs. mean absolute errors of charges from the previous model QEq and quantum-mechanical-derived charges (red)

to given molecular populations. Iczkowski and Margrave [1961] remind us that that Mulliken "pointed out the importance of using the ionization potentials corresponding to the appropriate valence state of the atom in a molecule [1934]."

Another issue that we have not confronted is that of recombination in the evolutionary algorithm. We perform well, but recombination may improve our optimization. Alternatively, it may add an additional level of complexity to our optimization without improving its performance. In fact, it might be favorable to develop our mutation such that our chromosomes move toward the best performer in a scheme similar to particle swarm optimization (PSO). This would require minor additions to our data model, but could prove to be a fruitful endeavor. The EA community has been increasing the use of PSO recently, likely due to the effectiveness of optimization and the ease of use.

13.3 Conclusions

The Goddard laboratory uses QEq to estimate atomic charges in ligand-docking simulations, used to predict potential drug candidates. Using EAs, we have improved our ability to model atomic charges, thereby increasing our chances of discovering a successful drug candidate.

Additionally, the developed code using EAs, as described in this thesis, has proven useful at optimizing other Goddard systems, including EFF and ReaxFF. The code was developed in such a way that it can easily be adapted and applied to additional systems by simply updating the parameters.

Appendices

Example Source Code

Evolutionary Algorithms in C

Algorithms and Source Code 9: C code for an Evolutionary Algorithm to optimize QEq

```
#include <pgapack.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <math.h>

#define max(A,B)        ((A) > (B) ? (A):(B))
#define min(A,B)        ((A) > (B) ? (B):(A))

double *mut_rates;
double *curr_mins;

int MyMutation (PGAContext * ctx, int p, int pop, double mr, int
   state);
void MyPrint (PGAContext * ctx, FILE * fp, int pop);
void tmp (PGAContext * ctx, int pop);
double MyEvaluate (PGAContext * ctx, int p, int pop);
void run (int argc, char **argv, int *para);
void parseParam (int len, int *para);

int
main (int argc, char **argv)
{
  MPI_Init (&argc, &argv);

  int my_array[] = { 1, 23, 17, 4, -5, 100 };
  parseParam (3, my_array);
  run (argc, argv, my_array);
```

```
  return (0);
}

/*
   ************************************************************************

 *    Initialize all members to the starting position as defined
    by the      *
 *   file "ffopt".
                                                                      *
 ************************************************************************
    */

int
MyMutation (PGAContext * ctx, int p, int pop, double mr, int
   state)
{
  int stringlen, i, count = 0;
  double k, updated, randnum, noise, avg, prev, curr, high, low,
     mnval =
    0.01, mx = .99, mxval = 3.0, delta;
  stringlen = PGAGetStringLength (ctx);
  // stringlen = 18;

  double ref[29], norms[29], actual[29];
  for (count = 0; count < 29; count++)
    {
      norms[count] = 1.0;
    }
  ref[1] = 10.87400;
  ref[2] = 8.56400;
  ref[3] = 7.79000;
  ref[4] = 6.82200;
  ref[5] = 4.52800;
  ref[6] = 3.00600;
  ref[7] = 2.84300;
  ref[8] = 2.42100;
  ref[9] = 2.33100;
  ref[10] = 2.18300;
  ref[11] = 5.34300;
  ref[12] = 6.89900;
  ref[13] = 8.74100;
  ref[14] = 6.92800;
  ref[15] = 14.94800;
  ref[16] = 9.89200;
  ref[17] = 8.85000;
```

```
  ref[18] = 7.52400;
  ref[19] = 13.89040;
  ref[20] = 4.77200;
  ref[21] = 4.59200;
  ref[22] = 3.84000;
  ref[23] = 3.69200;
  ref[24] = 3.42200;
  ref[25] = 10.12600;
  ref[26] = 11.76000;
  ref[27] = 13.36400;
  ref[28] = 8.97200;
  int gen = PGAGetGAIterValue (ctx);
  for (i = 2; i < stringlen; i++)
    {
      //if( i!=5 && i!=19){
      if (1)
        {
          //         if((i<11 ) ||(i>=15 && i<=24)){
          if ((i > 24) || (i >= 11 && i <= 14) || (i <= 5)
              || (i > 14 && i <= 19))
            {
              if (PGARandomFlip (ctx, mr))
                {
                  if (state == 1)
                    {
                      avg = PGAGetRealAllele (ctx, p, pop, i);
                    }
                  else
                    {
                      avg = 1;
                    }

                  mut_rates[i] = max (mut_rates[i], mnval);
                  k = PGARandomGaussian (ctx, avg, mut_rates[i]);

                  norms[i] = k;

                  //k = min(mxval, k);
                  // k = max(mnval,k);
                  PGASetRealAllele (ctx, p, pop, i, k);
                  printf ("mutation info %i %i %i %i %f %f %f %f
                     %f\n", gen,
                          p, i, pop, low, high, avg, k, mut_rates
                             [i]);
                  count++;
                }
            }
```

```
        }
    }
  for (count = 0; count < 29; count++)
    {
      actual[count] = norms[count] * ref[count];
      printf ("count %i %f\n", count, actual[count]);
      if (actual[count] < 0)
        {
          count = 0;
        }

    }
  if ((actual[1] < actual[2]) || (actual[2] < actual[3])
      || (actual[3] < actual[4]) || (actual[6] < actual[7])
      || (actual[7] < actual[8]) || (actual[8] < actual[9])
      || (actual[9] < actual[10]) || (actual[15] < actual[16])
      || (actual[16] < actual[17]) || (actual[17] < actual[18])
      || (actual[20] < actual[21]) || (actual[21] < actual[22])
      || (actual[22] < actual[23]) || (actual[23] < actual[24])
      || (actual[1] < actual[13]) || (actual[15] < actual[27])
      || (actual[2] < actual[14]) || (actual[16] < actual[28]))
    {
      //i=2;
      count = 0;
    }

  return (count);
}

void
parseParam (int len, int *para)
{
  printf ("len = %i\n", len);
  printf ("%i %i %i\n", para[0], para[1], para[2]);
  para[0] = 100;
  printf ("%i %i %i\n", para[0], para[1], para[2]);
  FILE *ifile, *ft, *fs, *fs2;
  char ch;

  ifile = fopen ("wagGA.par", "r");
  if (ifile == NULL)
    {
      puts
        ("Cannot find wagGA.par.  Please make sure that it is in
           the current working directory.");
      exit (-1);
    }
```

```
}

void
run (int argc, char **argv, int *para)
{
  printf ("hi %i\n", para[0]);
  int population_size = 150;
  //exit(0);
  int total_num_generations = 250;
  PGAContext *ctx;
  int i, j, n, m1, m2, popsize, numreplace;
  double probcross;

  ctx = PGACreate (&argc, argv, PGA_DATATYPE_REAL, 29,
     PGA_MINIMIZE);
  PGASetMutationType (ctx, PGA_MUTATION_GAUSSIAN);
  double *low, *high;
  int stringlen;
  stringlen = PGAGetStringLength (ctx);
  mut_rates = (double *) malloc (stringlen * sizeof (double));
  curr_mins = (double *) malloc (stringlen * sizeof (double));
  low = (double *) malloc (stringlen * sizeof (double));
  high = (double *) malloc (stringlen * sizeof (double));
  for (i = 0; i < stringlen; i++)
    {
      low[i] = 1.;
      high[i] = 1.;
      mut_rates[i] = .15;
    }
  low[0] = 1.;
  high[0] = 1.;
  PGASetRealInitRange (ctx, low, high);
  double rate_init = 1.0;
  double rate_main = .25;
  PGASetPrintFrequencyValue (ctx, 1);
  int num = population_size;
//      PGASetStoppingRuleType(ctx, PGA_STOP_NOCHANGE);
//      PGASetMaxNoChangeValue(ctx, 25);
  PGASetMaxGAIterValue (ctx, total_num_generations);
  PGASetPopSize (ctx, num);
  PGASetNoDuplicatesFlag (ctx, PGA_TRUE);
  PGASetSelectType (ctx, PGA_SELECT_PROPORTIONAL);

  PGASetStoppingRuleType (ctx, PGA_STOP_MAXITER);
  //PGASetMutationRealValue(ctx, 0);
  int check, rank, cnt, idx, floor;
  PGASetMutationRealValue (ctx, 0.0);
```

```
207   PGASetUp (ctx);
      rank = PGAGetRank (ctx, MPI_COMM_WORLD);
      for (cnt = 1; cnt < population_size; cnt++)
        {
          check = 0;
212       while (check == 0)
            {
              check = MyMutation (ctx, cnt, PGA_OLDPOP, rate_init, 0)
                 ;
              printf ("%i\n", check);
            }
217     }
      PGAEvaluate (ctx, PGA_OLDPOP, MyEvaluate, MPI_COMM_WORLD);
      // rank = PGAGetRank(ctx, NULL);
      // PGAEvaluate(ctx, PGA_OLDPOP, MyEvaluate, NULL);
      if (rank == 0)
222     {
          PGAFitness (ctx, PGA_OLDPOP);
          //printf( "\nthe population size is %i and the number of
             generations is set to %i.\n\n", population_size,
             total_num_generations + 1);
        }

227   while (!PGADone (ctx, MPI_COMM_WORLD))
        {
          // while (!PGADone(ctx, NULL)) {

          if (rank == 0)
232         {
              PGASelect (ctx, PGA_OLDPOP);
              PGASortPop (ctx, PGA_OLDPOP);
              tmp (ctx, PGA_OLDPOP);
              for (cnt = 0; cnt < num; cnt++)
237             {
                  floor = cnt / 5;
                  //printf("%i\n",floor);
                  //        floor = 0;
                  idx = PGAGetSortedPopIndex (ctx, floor);
242               PGACopyIndividual (ctx, idx, PGA_OLDPOP, cnt,
                     PGA_NEWPOP);
                  PGASetEvaluationUpToDateFlag (ctx, cnt, PGA_NEWPOP,
                       PGA_FALSE);
                  // PGASetEvaluationUpToDateFlag(ctx, cnt,
                     PGA_OLDPOP, PGA_FALSE);
                  //printf("test %i", test);
                  if (cnt >= 1)
```

```
247                 {
                      check = 0;
                      while (check == 0)
                        {
                          check = MyMutation (ctx, cnt, PGA_NEWPOP,
                             rate_main, 1);
252
                        }
                    }
                }
            }
257       //PGASelect(ctx, PGA_NEWPOP);
          //PGAEvaluate(ctx, PGA_NEWPOP, MyEvaluate, MPI_COMM_WORLD);
          PGAEvaluate (ctx, PGA_NEWPOP, MyEvaluate, MPI_COMM_WORLD);
          //PGAEvaluate(ctx, PGA_OLDPOP, MyEvaluate, MPI_COMM_WORLD);
          //PGAFitness(ctx, PGA_NEWPOP);
262
          if (rank == 0)
            {
              PGAFitness (ctx, PGA_NEWPOP);
            }
267
          if (rank == 0)
            {
              MyPrint (ctx, stdout, PGA_OLDPOP);
            }
272       PGAUpdateGeneration (ctx, MPI_COMM_WORLD);

        }
      PGAPrintReport (ctx, stdout, PGA_NEWPOP);

277   PGADestroy (ctx);
      MPI_Finalize ();
    }


282 void
    tmp (PGAContext * ctx, int pop)
    {
      double sum, avg, var, diff, mut_weight;
      double x;
287   double score = PGAGetFitness (ctx, 0, pop);
      double y = PGAGetStringLength (ctx);
      //int elite_percent = PGAGetPopSize(ctx)/1;
      int elite_percent = 10;
      mut_weight = 0.4;
292   double *values;
```

```
  values = (double *) malloc (elite_percent * sizeof (double));
  int gen = PGAGetGAIterValue (ctx);
  int cnt_a, cnt_b;
  PGASelect (ctx, pop);
297 PGASortPop (ctx, pop);
  for (cnt_b = 0; cnt_b < y; cnt_b++)
    {
      sum = 0;
      for (cnt_a = 0; cnt_a < elite_percent; cnt_a++)
302     {
          x =
            PGAGetRealAllele (ctx, PGAGetSortedPopIndex (ctx,
               cnt_a), pop,
                              cnt_b);
          values[cnt_a] = x;
307       sum = sum + x;
          score = PGAGetFitness (ctx, PGAGetSortedPopIndex (ctx,
             cnt_a), pop);
          printf ("VARDAT %i %i %i %f %i %i %f\n", gen, pop,
                  PGAGetSortedPopIndex (ctx, cnt_a), x, cnt_b,
                     cnt_a, score);
        }
312   avg = sum / elite_percent;
      diff = 0;
      for (cnt_a = 0; cnt_a < elite_percent; cnt_a++)
        {
          diff = (avg - values[cnt_a]) * (avg - values[cnt_a]) +
             diff;
317       // printf("sum = %f\n",sum);
        }
      mut_rates[cnt_b] = sqrt (diff / cnt_a) * mut_weight;
      printf ("VARFIN %i %i %f %f %i\n", gen, cnt_b, sqrt (diff /
          cnt_a), avg,
              cnt_a);
322
    }
}

void
327 MyPrint (PGAContext * ctx, FILE * fp, int pop)
{
        /*
           ************************************************************

         *  Print the string referenced by p and pop to the file
            fp.                  *
```

```
          ************************************************************
             */
  int i;
  int allelemax = PGAGetStringLength (ctx);

  int chrommax = PGAGetPopSize (ctx);
  int allelecnt = 0;
  int chromcnt = 0;
  double value, fitness, evaluation;
  int gen;
  FILE *fp2;
  if (fp2 = fopen ("arraydata.txt", "a+"))
    {

      for (chromcnt = 0; chromcnt < chrommax; chromcnt++)
        {
          gen = PGAGetGAIterValue (ctx);
          fitness = PGAGetFitness (ctx, chromcnt, pop);
          evaluation = PGAGetEvaluation (ctx, chromcnt, pop);
          fprintf (fp2, "%i %i %f %f ", gen, chromcnt, fitness,
             evaluation);
          for (allelecnt = 0; allelecnt < allelemax; allelecnt++)
            {

              value = PGAGetRealAllele (ctx, chromcnt, pop,
                 allelecnt);
              fprintf (fp2, "%f ", value);
            }


          fprintf (fp2, "\n");
        }

    }
  else
    {
      printf ("Error opening arraydata.txt\n");
    }
}

/*  The evaluation function.    */
double
MyEvaluate (PGAContext * ctx, int p, int pop)
{
        /*
           ************************************************************
```

```
         *    gaiter = the current iteration in our GA evolution
                                *
         *
            *
         *    x      = a dummy variable, for-loop iterator
                                     *
         *
            *
377      *    fp     = a standard file pointer that will be used
            to write                 *
         *    the chromosome update
                                                          *
         ************************************************************
            */

        /*
           ************************************************************
382      *   Evaluate the string here, and return a double
            representing                 *
         *   the quality of the solution.
                                                    *
         ************************************************************
            */
   //PGASetEvaluationUpToDateFlag(ctx, p, pop, PGA_TRUE);

387 FILE *fp, *fp_tst;
   FILE *fp2;
   int x, gaiter;
   gaiter = PGAGetGAIterValue (ctx);
   char filename[80], command[500], inputname[80];
392
        /*
           ************************************************************
         *    begin writing the chromosome updates to disk
                                     *
         ************************************************************
            */

397 // int inner_rank;
   // inner_rank = PGAGetRank(ctx, MPI_COMM_WORLD);
   // sprintf(filename, "ffopt_data_%i",inner_rank);
   //    sprintf(inputname, "score");
   // fp=fopen(filename,"w");
```

```
402   double value[29];
      int test;
      test = 0;
      for (test = 0; test < 29; test++)
        value[test] = PGAGetRealAllele (ctx, p, pop, test);
407
      sprintf (command,
               "python qeqD.py organics.qeq.par OptAll4Bio.par  %f %f
                   %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f
                   %f %f %f %f %f %f %f %f %f %f> /dev/null \n",
               value[0], value[1], value[2], value[3], value[4],
                  value[5],
               value[6], value[7], value[8], value[9], value[10],
                  value[11],
412            value[12], value[13], value[14], value[15], value[16],
                   value[17],
               value[18], value[19], value[20], value[21], value[22],
                   value[23],
               value[24], value[25], value[26], value[27], value[28])
                  ;

      system (command);
417
      float score, testScore, scoreNew;
      FILE *t;
      PGASetEvaluationUpToDateFlag (ctx, p, pop, PGA_TRUE);
      fp = fopen ("score", "r");
422   fscanf (fp, "%f\n", &score);
      // fscanf(fp,"%f\n", &scoreNew);
      fclose (fp);

      fp_tst = fopen ("score_test", "r");
427   fscanf (fp_tst, "%f\n", &testScore);
      // fscanf(fp,"%f\n", &scoreNew);
      fclose (fp_tst);



432


      int gen;
      gen = PGAGetGAIterValue (ctx);

437   printf ("scores %i %i %f %f\n", gen, p, score, testScore);

      // printf("%f %f scores",scorese, scoreNew);
      //printf("rank %i score for %i %i is %f.\n",inner_rank, p,pop,
         score);
```

```
    return (score);
}
```

The integrater for Source Code 9

Algorithms and Source Code 10: The integrater for Source Code 9

```
#!/usr/bin/env python
import sys, re,math, numpy, fnmatch
import subprocess, os

def score(target,test,data_dict, data_dict2):
    fp_target = open(target)
    fp_test   = open(test)
    lines_target = fp_target.readlines()
    lines_test = fp_test.readlines()
    fp_target.close()
    fp_test.close()
    test=[]
    # data_dict={}
    target=[]
    atom=[]
    atom_type=[]

    total_score = 0

    for line in lines_test:
        if re.match("HETATM",line):
            test.append(float(line.split()[12]))
            atom.append(line.split()[2])

    for line in lines_target:
        if re.match("HETATM",line):
            atom_type.append(line.split()[9])
            target.append(float(line.split()[12]))
    for (x,y,name,a_type) in zip(test, target,atom,atom_type):
        delta_q = (x-y)**2

        if not data_dict.has_key(name):
            data_dict[name]=[delta_q]
        else:
            data_dict[name].append(delta_q)
        if not data_dict2.has_key(a_type):
            data_dict2[a_type]=[delta_q]
        else:
            data_dict2[a_type].append(delta_q)
    return [data_dict, data_dict2]
```

```
def main():
    Xis      = numpy.ones(40)
    outlines = []
    in_name  = sys.argv[1]
    out_name = sys.argv[2]
    weight   = float(sys.argv[3])
    if len(sys.argv)==32:
        for cnt in xrange(28):
            Xis[cnt]    = float(sys.argv[cnt+4])
    else:
        print "Expecting 32 arguments but found %i"%len(sys.argv)
        print sys.argv
        sys.exit(-1)
    actual_vals=numpy.ones(len(Xis))
    #check(actual_vals)
    print sys.argv
    fp = open(in_name)
    lines = fp.readlines()
    for cnt in range(12,len(lines)):
        print len(lines),len(lines)-12
        print lines[cnt]
        parts=lines[cnt].split()
        parts[2]=str(float(parts[2])*Xis[cnt-12])
        actual_vals[cnt-12]=float(parts[2])*Xis[cnt-12]

        parts[3]=str(float(parts[3])*Xis[cnt-12+14])
        actual_vals[cnt-12+14]=float(parts[3])*Xis[cnt-12+14]

        parts[5]=str(float(parts[5])*weight)
        outln = ' '.join([str.rjust(parts[0],2,' '),"1","%8.5f"%
           float(parts[2]),"%8.5f"%float(parts[3]),"%8.6f"%float(
           parts[4]),"%8.5f"%float(parts[5]),"%7.6f"%float(parts[
           6]),"%7.6f"%float(parts[7])])
        outlines.append(outln)
        print outln
    #check(actual_vals)
    fp_out=open(out_name,'w')
    for cnt in range(12):
        print >>fp_out,lines[cnt],
    for line in outlines:
        print >> fp_out,line
    # print >> fp_out,"\n"
    fp_out.close()

    [score ,info, out_dict, out_dictB]    = run_qeq(out_name,"
       training", weight)
```

```
    [score_2 ,info_2, out_dict_2, out_dictB_2]    = run_qeq(
       out_name,"testing", weight)

    fp_val=open("score_%08.5f" % weight,"w")
    print >>fp_val, score
    fp_val.close()

    fp_val = open("score","w")
    [value, lines]=procAtoms(out_dict,out_dictB)
    print >>fp_val,value
    for line in lines:
        print >>fp_val,line
    fp_val.close()

    fp_tst = open("score_test","w")
    [value, lines]=procAtoms(out_dict_2,out_dictB_2)
    print >>fp_tst,value
    for line in lines:
        print >>fp_tst,line
    fp_tst.close()




    # print "scores ", (out_score/out_num)**.5, (a/b)**.5

def procAtoms(data,dataB):
    """Standard """
    out_lines = []
    out_score = 0
    out_num   = 0
    for atom in data:
        out_lines.append("\n%2s    %8.5f %8i %f"% (atom , (sum(
           data[atom])/len(data[atom]))**.5,len(data[atom]),max(
           data[atom])**.5))
        out_score +=  sum(data[atom])
        out_num   +=  len(data[atom])
        for atom_type in dataB:
            if re.match("%s[_\b]" % atom,atom_type):
                out_lines.append("\t%5s    %8.5f %8i %f"% (
                   atom_type , (sum(dataB[atom_type])/len(dataB[
                   atom_type]))**.5,len(dataB[atom_type]),max(
                   dataB[atom_type])**.5))

    value =(out_score/ out_num)**.5

    return [value, out_lines]
```

```
def locate(pattern, root):
    for path, dirs, files in os.walk(root):
        for filename in [os.path.abspath(os.path.join(path,
            filename)) for filename in files if fnmatch.fnmatch(
            filename, pattern)]:
             yield filename

def run_qeq(par_file, target_dir ,weight):
    print "weight"
    total_score = 0
    cnt=0
    nfo=[]
    rp_data_dict={}
    rp_data_dict_atomtype={}
    mol_names=[]

    for bgf in locate("*.bgf", target_dir):
        mol_names.append(bgf)


    for name in mol_names:
        out = "tmp." + name.split("/")[-1]
        print out
        p= subprocess.Popen("/ul/yi/bin/pqeq %s exact %s %s > /
            dev/null" % (par_file, name, out) , shell=True)
        sts = os.waitpid(p.pid, 0)[1]
        [ rp_data_dict, rp_data_dict_atomtype] =score("%s"%name,
            out , rp_data_dict,rp_data_dict_atomtype)
        cnt+=1

    total_error=0
    for key in rp_data_dict:
        total_error += sum(rp_data_dict[key])/len(rp_data_dict[
            key])


    print "weight:value",weight,total_score/cnt,"weight",cnt
    # value = total_score/cnt
    value = total_error/len(rp_data_dict.keys())
    print rp_data_dict_atomtype
    return [value, nfo, rp_data_dict, rp_data_dict_atomtype]


if __name__=="__main__":
    main()
```

Index

Bibliography

The Python Language. URL http://www.python.org.

Scipy. URL http://www.scipy.org.

G. K. Ackers. Energetics of subunit assembly and ligand binding in human hemoglobin. *Biophys J*, 32(1):331–346, Oct 1980. doi: 10.1016/S0006-3495(80)84960-5. URL http://dx.doi.org/10.1016/S0006-3495(80)84960-5.

G. K. Ackers and J. M. Holt. Asymmetric cooperativity in a symmetric tetramer: Human hemoglobin. *Journal of Biological Chemistry*, 281(17):11441–11443, 2006.

G. S. Adair. The hemoglobin system. I. Classification of reactions. *Journal of Biological Chemistry*, 63(2):493–497, 1925a.

G. S. Adair. The hemoglobin system. II. Theory of reactions which do not obey the law of constant proportions. *Journal of Biological Chemistry*, 63(2):499–501, 1925b.

M. L. Blinov, J. R. Faeder, B. Goldstein, and W. S. Hlavacek. BioNetGen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. *Bioinformatics*, 20(17):3289–3291, 2004. ISSN 1367-4803. doi: 10.1093/bioinformatics.

T V P Bliss and T Lomo. Long-lasting potentiation of synaptic transmission in the dentate area of the anaesthetized rabbit following stimulation of the perforant path. *The Journal of Physiology*, 232(2):331–356, 1973. URL http://jp.physoc.org/cgi/content/abstract/232/2/331.

Brenda L. Bloodgood and Bernardo L. Sabatini. Neuronal Activity Regulates Diffusion Across the Neck of Dendritic Spines. *Science*, 310(5749):866–869, 2005. doi: 10.1126/science.1114816. URL http://www.sciencemag.org/content/310/5749/866.abstract.

T.R. Chay and C. Ho. Statistical Mechanics Applied to Cooperative Ligand Binding to Proteins. *Proceedings of the National Academy of Sciences*, 70(12):3914, 1973.

D. E. Clark. *Evolutionary algorithms in molecular design*. Methods and principles in medicinal chemistry. Wiley-VCH, Weinheim ; New York, 2000. 2001278178 GBA0-Y0740 edited by David E. Clark. ill. , 23 cm. Includes bibliographical references and index.

R.J. Colbran and T.R. Soderling. Calcium/calmodulin-independent autophosphorylation sites of calcium/calmodulin-dependent protein kinase II. Studies on the effect of phosphorylation of threonine 305/306 and serine 314 on calmodulin binding using synthetic peptides. *J Biol Chem*, 265(19):11213–9, 1990.

V. Danos and C. Laneve. Formal molecular biology. *Theoretical Computer Science*, 325(1):69–110, 2004.

M. Delbruck. Statistical Fluctuations in Autocatalytic Reactions. *The Journal of Chemical Physics*, 8(1):120–124, 1940.

A. Dosemeci and T.S. Reese. Inhibition of Endogenous Phosphatase in a Postsynaptic Density Fraction Allows Extensive Phosphorylation of the Major Postsynaptic Density Protein. *Journal of Neurochemistry*, 61(2):550–555, 1993. ISSN 1471-4159. doi: 10.1111/j.1471-4159.1993.tb02158.x. URL http://dx.doi.org/10.1111/j.1471-4159.1993.tb02158.x.

L.J. Fogel, A.J. Owens, and M.J. Walsh. *Artificial intelligence through simulated evolution*. Wiley, Chichester, WS, UK, 1966.

Q. H. Gibson. Reaction of Oxygen with Hemoglobin and Kinetic Basis of Effect of Salt on Binding of Oxygen. *Journal of Biological Chemistry*, 245(13):3285–&, 1970.

S.J. Gill and J. Wyman. *Binding and Linkage: Functional Chemistry of Biological Macromolecules*. University Science Books, 1990. ISBN 9781891389641. URL http://books.google.com/books?id=pGhEPgAACAAJ.

D. T. Gillespie. General Method for Numerically Simulating Stochastic Time Evolution of Coupled Chemical-Reactions. *Journal of Computational Physics*, 22(4):403–434, 1976.

D. T. Gillespie. Exact Stochastic Simulation of Coupled Chemical-Reactions. *Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

J. H. Holland. Outline for a Logical Theory of Adaptive Systems. *J. ACM*, 9(3):297–314, July 1962. ISSN 0004-5411. doi: 10.1145/321127.321128. URL http://doi.acm.org/10.1145/321127.321128.

J. M. Holt, A. L. Klinger, C. S. Yarian, V. Keelara, and G. K. Ackers. Asymmetric distribution of cooperativity in the binding cascade of normal human hemoglobin. 1. Cooperative and noncooperative oxygen binding in Zn-substituted hemoglobin. *Biochemistry*, 44(36):11925–11938, 2005.

J. J. Hopfield. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *Proceedings of the National Academy of Sciences*, 79:2554–2558, 1982. URL http://www.bibsonomy.org/bibtex/24041c6a004f343806a7e47c9cd7ccf59/brian.mingus.

R.P. Iczkowski and J.L. Margrave. Electronegativity. *Journal of the American Chemical Society*, 83(17):3547–3551, 1961.

K. Imai, L. Rossi-Bernardi, E. Antonini, and C. Emilia. [27] Analysis of ligand binding equilibria. Volume 76:470–486, 1981.

K. De Jong. Parameter Setting in EAs: a 30 Year Perspective,. *Studies in Computational Intelligence*, 54:1–18, 2007.

D. E. Koshland, G. Nemethy, and D. Filmer. Comparison of Experimental Binding Data and Theoretical Models in Proteins Containing Subunits. *Biochemistry*, 5(1): 365, 1966.

J. R. Koza. Evolving a Computer Program to Generate Random Numbers Using the Genetic Programming Paradigm. In Richard K. Belew and Lashon B. Booker, editors, *ICGA*, pages 37–44. Morgan Kaufmann, 1991. ISBN 1-55860-208-9.

Y. Lai, A.C. Nairn, F. Gorelick, and P. Greengard. Ca2+/calmodulin-dependent protein kinase II: identification of autophosphorylation sites responsible for generation of Ca2+/calmodulin-independence. *Proceedings of the National Academy of Sciences of the United States of America*, 84(16):5710–4, aug 1987. URL `http://ukpmc.ac.uk/abstract/MED/3475699`. Research Support, Non-U.S. Gov't,Research Support, U.S. Gov't, Non-P.H.S.,Research Support, U.S. Gov't, P.H.S.,.

D. Levine. Users Guide to the PGAPack Parallel Genetic Algorithm Library, 1996. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.55.5086`.

L. Lok and R. Brent. Automatic generation of cellular reaction networks with Moleculizer 1.0. *Nature Biotechnology*, 23(1):131–136, 2005.

E.W. Lund. Guldberg and Waage and the law of mass action. *Journal of Chemical Education*, 42(10):548, 1965. doi: 10.1021/ed042p548. URL `http://pubs.acs.org/doi/abs/10.1021/ed042p548`.

J.T. MacQueen. Some observations concerning the van't Hoff equation. *Journal of Chemical Education*, 44(12):755, 1967.

S.G. Miller and M.B. Kennedy. Regulation of Brain Type II Ca^{2+} Calmodulin-Dependent Protein Kinase by Autophosphorylation: A Ca^{2+}-Triggered Molecular Switch. *Cell*, 44:861–870, 1986.

F. C. Mills, M. L. Johnson, and G. K. Ackers. Oxygenation-Linked Subunit Interactions in Human Hemoglobin - Experimental Studies on Concentration-Dependence of Oxygenation Curves. *Biochemistry*, 15(24):5350–5362, 1976.

J. Monod, J. Wyman, and J. P. Changeux. On Nature of Allosteric Transitions - a Plausible Model. *Journal of Molecular Biology*, 12(1):88, 1965.

P. Mullasseril, A. Dosemeci, J.E. Lisman, and L.C. Griffith. A structural mechanism for maintaining the "on-state" of the CaMKII memory switch in the post-synaptic density. *Journal of Neurochemistry*, 103(1):357–364, 2007. URL http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2665908&tool=pmcentrez&rendertype=abstract.

R.S. Mulliken. A new electroaffinity scale; together with data on valence states and on valence ionization potentials and electron affinities. *The Journal of Chemical Physics*, 2:782, 1934.

L. Pauling. The oxygen equilibrium of hemoglobin and its structural interpretation. *Proceedings of the National Academy of Sciences of the United States of America*, 21:186–191, 1935.

A. K. Rappé and W. A. Goddard. Charge Equilibration for Molecular-Dynamics Simulations. *Journal of Physical Chemistry*, 95(8):3358–3363, 1991. Fh098 Times Cited:1239 Cited References Count:31.

I. Rechenberg. Cybernetic Solution path of an Experimental Problem. *Royal Aircraft Establishment, Farnborough*, Library Translation 1122, 1965.

A. Renyi. Treating chemical reactions using the theory of stochastic processes. *Magyar Tud. Akad. Alkalm. Mat. Int. Közl.*, 2:83–101, 1953.

B. L. Sabatini, T. G. Oertner, and K. Svoboda. The Life Cycle of Ca2+ Ions in Dendritic Spines. *Neuron*, 33(3):439–452, 2002. doi: 10.1016/S0896-6273(02)00573-1.

M. Sanhueza, G. Fernandez-Villalobos, I.S. Stein, G. Kasumova, P. Zhang, K.U. Bayer, N. Otmakhov, J.W. Hell, and J. Lisman. Role of the CaMKII/NMDA Receptor Complex in the Maintenance of Synaptic Strength. *Journal of Neuroscience*, 31(25):9170–9178, 2011. URL http://www.jneurosci.org/cgi/doi/10.1523/JNEUROSCI.1250-11.2011.

J. Sefcik, E. Demiralp, T. Cagin, and W.A. Goddard III. Dynamic Charge Equilibration-Morse stretch force field: Application to energetics of pure silica zeolites. *Journal of Computational Chemistry*, 23(16):1507–1514, 2002. URL http://dblp.uni-trier.de/db/journals/jcc/jcc23.html#SefcikDCG02.

J.M. Shifman, M.H. Choi, S. Mihalas, S.L. Mayo, and M.B. Kennedy. Ca2+/calmodulin-dependent protein kinase II (CaMKII) is activated by calmodulin with two bound calciums. *Proceedings of the National Academy of Sciences*, 103(38):13968–13973, 2006. doi: 10.1073/pnas.0606433103. URL http://www.pnas.org/content/103/38/13968.abstract.

K. Singer. Application of the Theory of Stochastic Processes to the Study of Irreproducible Chemical Reactions and Nucleation Processes. *Journal of the Royal Statistical Society. Series B (Methodological)*, 15(1):pp. 92–106, 1953. ISSN 00359246. URL http://www.jstor.org/stable/2983726.

A. Srinivasan. Kinpy. URL http://code.google.com/p/kinpy.

J. T. Su and W. A. Goddard. The dynamics of highly excited electronic systems: applications of the electron force field. *J Chem Phys*, 131(24):244501, Dec 2009. doi: 10.1063/1.3272671. URL http://dx.doi.org/10.1063/1.3272671.

A. C. T. van Duin, S. Dasgupta, F. Lorant, and W. A. Goddard. ReaxFF: A Reactive Force Field for Hydrocarbons. *The Journal of Physical Chemistry A*, 105(41):9396–9409, 2001.

Wikipedia. Ergodic process — Wikipedia, The Free Encyclopedia, 2009. URL http://en.wikipedia.org/wiki/Ergodic_process. [Online; accessed 8-February-2009].

J. R. Williamson. Cooperativity in macromolecular assembly. *Nat Chem Biol*, 4(8): 458–465, 2008.

M.R. Wright. *An introduction to chemical kinetics*. Wiley, 2004. ISBN 9780470090589. URL http://books.google.com/books?id=pZQn1RYcFuYC.