SLOPPINESS, MODELING, AND EVOLUTION IN BIOCHEMICAL NETWORKS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Ryan Nicholas Gutenkunst

January 2008

© 2008 Ryan Nicholas Gutenkunst

ALL RIGHTS RESERVED

SLOPPINESS, MODELING, AND EVOLUTION IN BIOCHEMICAL NETWORKS

Ryan Nicholas Gutenkunst, Ph.D.

Cornell University 2008

The wonderful complexity of livings cells cannot be understood solely by studying one gene or protein at a time. Instead, we must consider their interactions and study the complex biochemical networks they function in.

Quantitative computational models are important tools for understanding the dynamics of such biochemical networks, and we begin in Chapter 2 by showing that the sensitivities of such models to parameter changes are generically 'sloppy', with eigenvalues roughly evenly spaced over many decades. This sloppiness has practical consequences for the modeling process. In particular, we argue that if one's goal is to make experimentally testable predictions, sloppiness suggests that collectively fitting model parameters to system-level data will often be much more efficient that directly measuring them.

In Chapter 3 we apply some of the lessons of sloppiness to a specific modeling project involving in vitro experiments on the activation of the heterotrimeric G protein transducin. We explore how well time-series activation experiments can constrain model parameters, and we show quantitatively that the T177A mutant of transducin exhibits a much slower rate of rhodopsin-mediated activation than the wild-type.

All the preceding biochemical modeling work is performed using the SloppyCell modeling environment, and Chapter 4 briefly introduces SloppyCell and some of the analyses it implements. Additionally, the two appendices of this thesis contain

preliminary user and developer documentation for SloppyCell.

Modelers tweak network parameters with their computers, and nature tweaks such parameters through evolution. We study evolution in Chapter 5 using a version of Fisher's geometrical model with minimal pleiotropy, appropriate for the evolution of biochemical parameters. The model predicts a striking pattern of cusps in the distribution of fitness effects of fixed mutations, and using extreme value theory we show that the consequences of these cusps should be observable in feasible experiments.

Finally, this thesis closes in Chapter 6 by briefly considering several topics: sloppiness in two non-biochemical models, two technical issues with building models, and the effect of sloppiness on evolution beyond the first fixed mutation.

BIOGRAPHICAL SKETCH

The author was born in Wilkes-Barre, Pennsylvania on September 21, 1980. When he was seven, he and his family moved to Pueblo, Colorado. They were lured by a new job for his father and 300 days of sunshine a year for his mother. Much of Ryan's youth was spent hiking and skiing with his father, and Ryan considers himself a Colorado mountain boy.

Academically he was focused on math and science from a young age (at least after giving up his dream of being a garbage man), but among his most valuable experiences in high school was his time on debate team.

In the fall of 1998 he began his studies at the California Institute of Technology. There he worked with Dr. Eric Black on a simple description of a laser interferometric gravitational wave detector and with Prof. Anthony Leonard on experimental fluid mechanics. Ryan survived the Caltech physics curriculum while simultaneously developing an interest in the law, and in his senior year he applied to both physics graduate schools and law schools. Ultimately he decided that life in a suit was not for him, and he enrolled in the Cornell University physics graduate program in the fall of 2002.

Ryan was initially unsure where his research interests were focused, but his experiences in the IGERT program for nonlinear sciences led him toward mathematical and computational biology. His first serious graduate research experience was modeling the foraging behavior of tuna with Prof. Leah Edelstein-Keshet at the University of British Columbia during his IGERT summer internship.

Upon returning to Cornell, Ryan was offered a position in the group of Prof. James Sethna. Ryan's initial work with the Sethna group was focused on protein structure prediction, but he soon focused on sloppiness in biochemical networks and modeling the activation of heterotrimeric G proteins. Along the way, Ryan

took charge of SloppyCell, a port of Dr. Kevin Brown's simulation software from C++ to Python that was begun by Dr. Chris Myers. Recently, Ryan's interests have turned toward evolution, and his most recent project focused on an abstract model of adaptive evolution. Upon graduation Ryan will be starting a one-year post-doc with Scott Williamson in Cornell's department of Biological Statistics and Computational Biology.

To Shannon and my parents.

ACKNOWLEDGEMENTS

It gives me great pleasure to thank a few of the many people who have helped me throughout my academic career.

First I must offer heartfelt thanks to my advisor, Jim Sethna. The longer I've been in graduate school, the more I realize how lucky I've been to work with Jim. He not only brims with ideas but also brings an infectious enthusiasm to science. Jim's positive attitude helped keep me motivated after several rejections, and he often found the bright side to what I thought were project-ending difficulties.

I also thank the other members of my special committee. Eric Siggia's skeptical questions were sometimes intimidating, but they helped drive several projects forward. Rick Cerione and his group, particularly Jon Erickson and Sekar 'Ram' Ramachandran, led my initial forays into biochemistry. Finally, Carl Franck is perhaps the kindest man in the department, and his A exam question led to great research project.

Work would have been terribly dull were it not for the other members of the Sethna group. Josh Waterfall always had a cheerful word, and Fergal Casey's cynicism helped keep things interesting. They were invaluable resources to bounce ideas off of, as I came to appreciate after they graduated. The computer lab was quite the lonely place until new students joined the theory group.

Much of my work revolved around SloppyCell, and it could not have progressed so far without the help of others. Kevin Brown and Chris Myers, in particular, started the software, and their design shapes it to this day. Jordan Atlas and Bob Kuczenski both contributed significant ideas and code, while Sarah Stockwell and Tamara Galor-Neah courageously suffered through bugs in the early versions.

My short and interesting time spent studying accelerator design was prompted by Georg Hoffstaetter, and I could not have made any progress at all without

Christopher Mayes and David Sagan. Conversations with Jason Mezey and Ben Logsdon were very helpful in shaping my picture of evolution, and their kindness helped my interest in the subject flourish.

The first two years of my graduate career were funded by an NSF IGERT fellowship in nonlinear systems, and the following two were funded by an NIH molecular biophysics training grant. The IGERT program and its head, Steve Strogatz, showed me the wonderful breadth of subjects outside of traditional physics for which math and computation can offer insights. The IGERT fellowship also gave me the amazing opportunity to work with Leah Edelstein-Keshet at the University of British Columbia. Her charm and warmth gave me a much needed confidence boost in my abilities as a scientist, and I learned a great deal about productive research from my project with her and her post-doc Nathaniel Newlands.

Of course, I wouldn't be here without my parents. They taught me to value education and knowledge and fostered my interest in science without falling into the trap of being pushy and controlling.

Finally, I must thank my wife Shannon for the invaluable support she has given me throughout graduate school. Her love helped keep me sane during the rough times by reminding me how much more there is to life than computers and papers.

TABLE OF CONTENTS

Biographical Sketch . . . iii
Dedication . . . v
Acknowledgements . . . vi
Table of Contents . . . viii
List of Figures . . . xii
List of Tables . . . xiv
List of Listings . . . xv

1 Overview **1**

2 Universally Sloppy Parameter Sensitivities in Systems Biology Models **5**
2.1 Abstract . . . 5
2.2 Non-technical Summary . . . 6
2.3 Introduction . . . 7
2.4 Results . . . 9
2.4.1 Systems Biology Models have Sloppy Sensitivity Spectra . . . 9
2.4.2 Consequences of Sloppiness . . . 13
2.4.2.1 Parameter Values from Collective Fits . . . 15
2.4.2.2 Predictions from Direct Parameter Measurements . . . 18
2.5 Discussion . . . 20
2.6 Methods . . . 24
2.6.1 Hessian Computations . . . 24
2.6.2 Parameter Uncertainties . . . 24
2.6.3 Prediction Uncertainties . . . 25
2.6.4 Software . . . 25
2.7 Supporting Information . . . 25
2.7.1 Accession Numbers . . . 26
2.8 Acknowledgments . . . 26
2.9 Funding . . . 26
2.S1 Stiffest Eigenvectors . . . 27
2.S2 Effect of Other Poorly Determined Parameters . . . 36
2.S3 Fragility of Other Predictions . . . 37
2.S4 Rescaled Model of Brown et al. . . . 39
2.S5 Sloppy Model Analysis of Brodersen et al. Binding Studies . . . 41

3 Computational Kinetic Modeling of Experiments on the Activation of a Transducin Mutant **43**
3.1 Introduction . . . 43
3.2 Data . . . 44
3.3 Computational Model . . . 46
3.4 Fitting the Model . . . 47

3.5 Parameter Uncertainties . . . 50
3.5.1 Bayesian Ensembles . . . 52
3.5.2 Covariance Analysis . . . 55
3.6 Discussion . . . 56
3.A Model Equations . . . 58
3.B Sloppiness of the Models . . . 59

4 Falsifiable Modeling of Biochemical Networks with SloppyCell 62
4.1 Abstract . . . 62
4.2 Introduction . . . 62
4.3 Methods . . . 63
4.4 Features and Implementation . . . 66
4.5 Conclusions . . . 67
4.6 Acknowledgements . . . 67

5 Adaptive Mutation in a Geometrical Model of Biochemical Evolution 68
5.1 Abstract . . . 68
5.2 Introduction . . . 68
5.3 The Model . . . 70
5.4 Results . . . 72
5.4.1 Typical Mutation Size . . . 72
5.4.2 Adaptive Mutation Probability Densities . . . 74
5.4.3 Cusp Spacings . . . 78
5.4.4 Non-spherical Fitness Functions . . . 79
5.5 Discussion . . . 81
5.6 Acknowledgments . . . 84
5.A Extreme Value Theory for Δ . . . 84
5.B Numerical Evaluation of $\langle\Delta\rangle$ . . . 86

6 Potpourri 87
6.1 Other Sloppy Systems . . . 87
6.1.1 Cornell's Proposed Energy Recovery Linac . . . 88
6.1.2 Kinematics of Insect Hovering . . . 90
6.2 Scale Factor Entropy and Priors . . . 93
6.3 Faster Monte-Carlo Convergence in Curved Landscapes . . . 97
6.4 Biochemical Evolution Beyond the First Fixation . . . 103
6.4.1 Continuous Time Monte Carlo Simulation . . . 104

A SloppyCell User Documentation 108
A.1 Overview . . . 109
A.1.1 Working Interactively . . . 109
A.1.2 Accessing SloppyCell . . . 110
A.1.3 Networks . . . 110

A.1.4 Dynamics . . . 111
A.1.5 Models . . . 112
A.1.5.1 Priors . . . 112
A.1.6 Experiments . . . 113
A.1.6.1 Scale Factors . . . 113
A.1.6.2 Data Format . . . 114
A.1.7 Optimization . . . 115
A.1.8 Cost and Residual Derivatives . . . 115
A.1.9 Ensembles . . . 116
A.1.9.1 Assessing and Speeding Convergence . . . 117
A.1.9.2 Predicting from an Ensemble . . . 118
A.1.10 Plotting . . . 118
A.1.11 KeyedLists . . . 119
A.1.12 Input and Output . . . 119
A.1.13 Miscellaneous Utilities . . . 121
A.1.14 Parallelization . . . 121
A.2 JAK-STAT Example . . . 121
A.2.1 Other Examples . . . 126
A.3 Installation . . . 126
A.3.1 Required Dependencies . . . 126
A.3.2 Optional Dependencies . . . 127
A.3.3 On Linux . . . 127
A.3.4 OS X . . . 128
A.3.4.1 Pre-built binary . . . 128
A.3.4.2 From source code . . . 129
A.3.5 Windows . . . 129
A.3.5.1 Pre-built binary . . . 129
A.3.5.2 From source code . . . 129
A.3.6 Testing the Installation . . . 129
A.4 Troubleshooting . . . 130
A.4.1 Failing Integrations . . . 130

B SloppyCell Developer Documentation 131
B.1 Test Suite . . . 131
B.2 Logging . . . 132
B.3 Integrator . . . 132
B.4 ExprManip . . . 133
B.5 Dynamic Functions . . . 133
B.5.1 Compilation . . . 134
B.5.2 Execution . . . 134
B.6 Sensitivity Integration . . . 136
B.6.1 Handling Events . . . 137
B.6.1.1 Time Sensitivities . . . 137
B.6.1.2 Variable Sensitivities . . . 138

B.6.1.3 Implementation 139
B.6.2 Adjoint Method 140
B.7 Parallel Execution 141

Bibliography **143**

LIST OF FIGURES

1.1 Illustrative complex biochemical network 2

2.1 Parameter sensitivity spectra 11
2.2 Sloppiness and uncertainties 14
2.3 Fitting parameters to idealized data 16
2.4 Parameter and prediction uncertainties 19
2.5 Stiffest eigenvectors for model (a): Tyson's model of the eukaryotic cell cycle 27
2.6 Stiffest eigenvectors for model (b): Zwolak et al.'s model of the Xenopus egg cell cycle 28
2.7 Stiffest eigenvectors for model (c): Goldbeters's model of eukaryotic mitosis 28
2.8 Stiffest eigenvectors for model (d): Vilar et al.'s generic circadian rhythm model 29
2.9 Stiffest eigenvectors for model (e): Edelstein et al.'s model of nicotinic acetylcholine intra-receptor dynamics 29
2.10 Stiffest eigenvectors for model (f): Kholodenko's model of a generic kinase cascade 30
2.11 Stiffest eigenvectors for model (g): Lee et al.'s model of Xenopus Wnt signaling 30
2.12 Stiffest eigenvectors for model (h): Leloup and Goldbeters's model of Drosophila circadian rhythm 31
2.13 Stiffest eigenvectors for model (i): Brown et al.'s model of rat growth-factor signaling 31
2.14 Stiffest eigenvectors for model (j): von Dassow et al.'s model of the Drosophila segment polarity network 32
2.15 Stiffest eigenvectors for model (k): Ueda et al.'s model of Drosophila circadian rhythm 32
2.16 Stiffest eigenvectors for model (l): Locke et al.'s model of Arabidopsis circadian rhythm 33
2.17 Stiffest eigenvectors for model (m): Zak et al.'s model of an in silico regulatory network 33
2.18 Stiffest eigenvectors for model (n): Curto et al.'s model of human purine metabolism 34
2.19 Stiffest eigenvectors for model (o): Chassagnole et al.'s model of E. coli carbon metabolism 34
2.20 Stiffest eigenvectors for model (p): Chen et al.'s model of the budding yeast cell cycle 35
2.21 Stiffest eigenvectors for model (q): Sasagawa et al.'s model of rat growth-factor signaling 35
2.22 Effect of other missing parameters on example prediction 36
2.23 Prediction uncertainties for Ras activity given EGF stimulation . . 38

2.24 Prediction uncertainties for Mek activity given NGF stimulation . . 38
2.25 Eigenvalue for rescaled Brown et al. models 40
2.26 Sloppy eigenvalues of Brodersen et al.'s models 42

3.1 Fits to reduced data 45
3.2 Illustration of our model for heterotrimeric G protein activation . . 47
3.3 Optimized initial condition adjustments 51
3.4 Example ensemble parameter distributions 54
3.5 Eigenvalues for G protein models 60
3.6 Stiffest eigenvectors for fits with fixed initial conditions 60
3.7 Eigenvector components for fits with optimized initial conditions . 61

4.1 Uncertainty analysis of JAK-STAT model 65

5.1 Illustration of the biochemical geometrical model 71
5.2 Fitness effects probability densities 76
5.3 Mean relative cusp spacing Δ versus N 79

6.1 Eigenvalue spectrum for a model of Cornell's proposed ERL 89
6.2 Stiffest three eigenvectors for a model Cornell's ERL 90
6.3 Eigenvalue spectrum for a model of the kinematics of fruitfly flight 91
6.4 Stiffest three eigenvectors for the fruitfly 92
6.5 Effect of scale factor priors on ensemble construction 95
6.6 $J^{\mathsf{T}}J$ eigenvalues from several points along PC12 ensemble 97
6.7 Importance-sampled Monte-Carlo in the Rosenbrock function . . . 99
6.8 Monte-Carlo with $J^{\mathsf{T}}J$ recalculation in the Rosenbrock function . . 101
6.9 Comparison of Monte-Carlo algorithm convergence 102
6.10 Adaptive walks in spherical and sloppy landscapes 105
6.11 'Trapping' in 2-D evolution 106

A.1 Example autocorrelation functions 117
A.2 Automatically generated network diagram for G protein model . . 120
A.3 Optimal fit for JAK-STAT example 124
A.4 Histogram of log `'r3'` for the JAK-STAT model 125
A.5 Prediction uncertainties for the JAK-STAT model 126

LIST OF TABLES

3.1 Best-fit costs for G-protein models 49
3.2 Ensemble temperatures 53
3.3 k_{ex} confidence intervals 55

LIST OF LISTINGS

A.1 Example of experimental data format 114
A.2 Example SloppyCell script for the JAK-STAT model 123

CHAPTER 1

OVERVIEW

Biology has made astonishing progress characterizing the molecular components of life, culminating in the sequencing of whole genomes [1]. Understanding the remarkable complexity of life, however, requires more than just a list of parts; it requires understanding the networks of interactions between those parts [2]. (As an example, Figure **??** illustrates a complex biochemical network with particularly important medical consequences.) Mathematical and computational modeling will play an important role in our quest to understand the organization and dynamics of these networks [3, 4]. Even after decades of research [5], however, best practices for modeling such complex multi-parameter systems are still being developed.

One important consideration is how to deal with uncertainties in the data [6], in the fit parameters, and in resulting predictions. Brown et al. rigorously explored one source of uncertainty in their model of growth-factor signaling in PC12 cells; their analysis considered not just the set of parameters that best fit the data but a statistical sampling of *all* parameter sets that fit the data [7, 8]. Like in many other systems [9], the space of parameter sets that could fit the data was vast. Perhaps surprisingly, some predictions were still very well constrained even in the face of this enormous parameter uncertainty. Brown et al. found a striking 'sloppy' pattern in the sensitivity of their model to parameter changes; when plotted on a logarithmic scale, the sensitivity eigenvalues were roughly evenly spaced over many decades.

Since that study, sloppiness has been a focus of our group's work. Notably, the PC12 model was not unique; a model of atomic interatomic potentials is sloppy [10], as is the classic problem of fitting exponentials [11]. In fact, their appears to be a universality class of sloppy models [11]. This thesis opens in Chapter 2 by

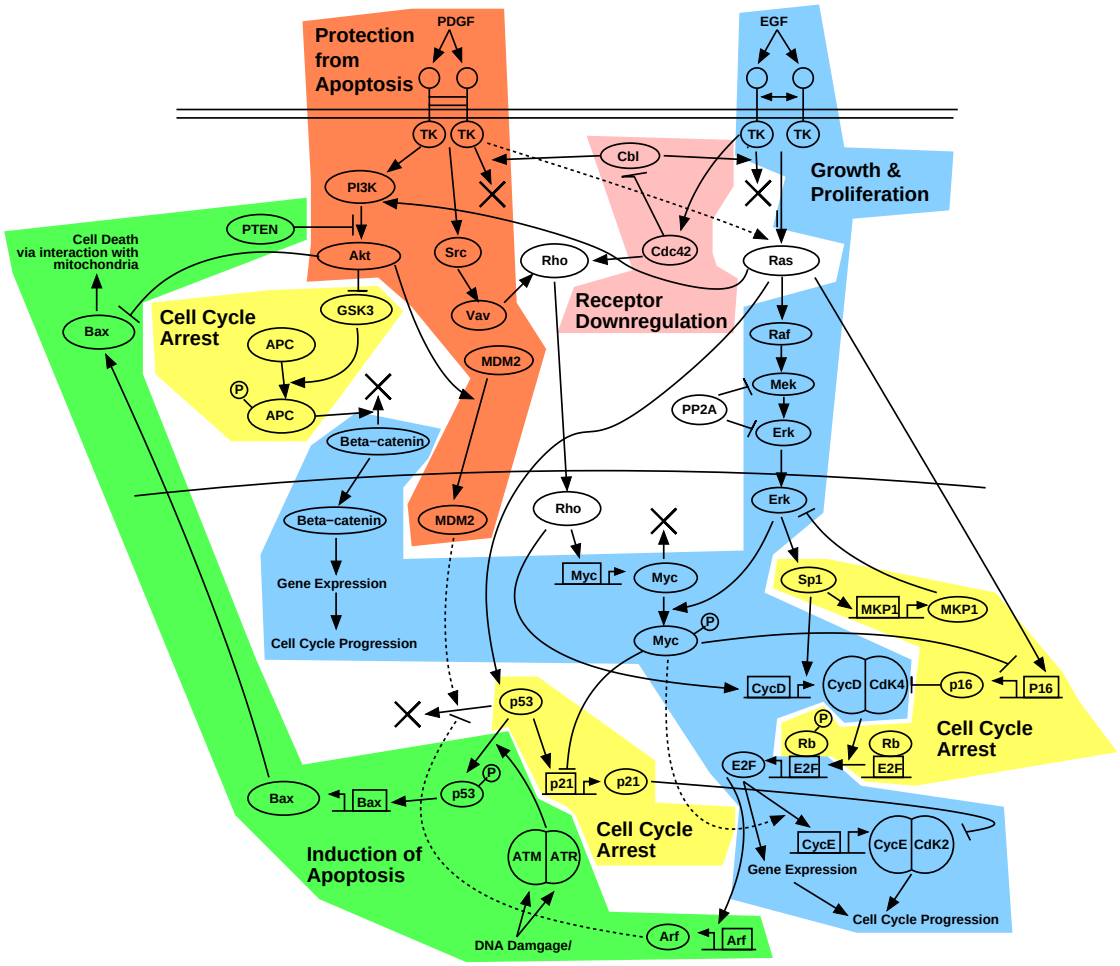


Figure 1.1: Shown is a cartoon of an illustrative biochemical network. Compiled by Rick Cerione and Kevin Brown and illustrated by the author of this thesis, this network includes many of the players involved in cancer biology.

empirically testing the universality of sloppiness in models biochemical networks. We consider a diverse collection of models from the systems biology literature and show they they all have sloppy sensitivity spectra.

In Chapter 2 we also explore some practical consequences of sloppiness for modelers. In particular, we show that direct parameter measurements may be a very inefficient way to build predictive models. Predictions of the behavior of the system as a whole are best constrained by experiments that probe that system behavior, even when those experiments only very loosely constrain a model's parameters.

Chapter 3 discusses an application of the 'sloppy modeling' approach. Using data collected by Sekar Ramachandran in the Cerione lab, we study a model of the activation of heterotrimeric G proteins. The model itself is relatively simple; the challenge lies in connecting it with noisy and extracting as much insight as possible from a limited view of the dynamics.

Much of my work has focused on SloppyCell, a software environment for model building that has found application both inside [12] (Chapters 2 and 3) and outside [13] the Sethna group. Chapter 4 is a short introduction to and advertisement for the SloppyCell. The code is open source in part because of my desire to facilitate reproducible computational research [14]. For example, the supplementary material for the paper in Chapter 2 includes SloppyCell scripts to reproduce most of the results.

Theodosius Dobzhansky famously pointed out [15, 16] that: "Nothing in biology makes sense except in the light of evolution." Evolution has fascinated many groups in physics [17] and the possible evolutionary implications of sloppiness have interested our group for quite some time. Chapter 5 represents our first concrete step in addressing evolution. There we consider evolution not in terms of sequences or phenotypes, but in terms of biochemical parameters. Our geometrical model

predicts striking, experimentally-accessible cusps in the distribution fitness of effects of adaptive mutations. Perhaps surprisingly, we also find that sloppiness has little influence on these cusps, suggesting that it leaves little signature on single adaptive steps.

The body of this thesis closes with Chapter 6, which discusses a smattering of topics. These include two non-network sloppy models, some technical details about building effective parameter ensembles, and a brief look at the effects of sloppiness on evolution beyond the first step in an adaptive walk.

Finally, in the interests of keeping SloppyCell alive, this thesis includes two appendices documenting SloppyCell, one for users and one for developers. The group cannot claim to be experts at software design, but we have tried to adhere to good software development practices [18], and these appendices are an important part of making the code useful to others. After submission of this thesis, these appendices will be included on the SloppyCell website and in the SloppyCell source code distribution.

CHAPTER 2

UNIVERSALLY SLOPPY PARAMETER SENSITIVITIES IN SYSTEMS BIOLOGY MODELS*

2.1 Abstract

Quantitative computational models play an increasingly important role in modern biology. Such models typically involve many free parameters, and assigning their values is often a substantial obstacle to model development. Directly measuring in vivo biochemical parameters is difficult, and collectively fitting them to other experimental data often yields large parameter uncertainties. Nevertheless, in earlier work we showed in a growth-factor-signaling model that collective fitting could yield well-constrained predictions, even when it left individual parameters very poorly constrained. We also showed that the model had a 'sloppy' spectrum of parameter sensitivities, with eigenvalues roughly evenly distributed over many decades. Here we use a collection of models from the literature to test whether such sloppy spectra are common in systems biology. Strikingly, we find that every model we examine has a sloppy spectrum of sensitivities. We also test several consequences of this sloppiness for building predictive models. In particular, sloppiness suggests that collective fits to even large amounts of ideal time-series data will often leave many parameters poorly constrained. Tests over our model collection are consistent with this suggestion. This difficulty with collective fits may seem to argue for direct parameter measurements, but sloppiness also implies that such measurements must be formidably precise and complete to usefully constrain many model predictions. We confirm this implication in our growth-factor-signaling model. Our results suggest that sloppy sensitivity spectra

*In press at *PLoS Computational Biology* with authors Ryan N. Gutenkunst, Joshua J. Waterfall, Fergal P. Casey, Kevin S. Brown, Christopher R. Myers and James P. Sethna.

are universal in systems biology models. The prevalence of sloppiness highlights the power of collective fits and suggests that modelers should focus on predictions rather than on parameters.

2.2 Non-technical Summary

Dynamic systems biology models typically involve many kinetic parameters, the quantitative determination of which has been a serious obstacle to using these models. Previous work showed for a particular model that useful predictions could be extracted from a fit long before the experimental data constrained the parameters, even to within orders of magnitude. This was attributed to a 'sloppy' pattern in the model's parameter sensitivities; the sensitivity eigenvalues were roughly evenly spaced over many decades. Consequently, the model behavior depended effectively on only a few 'stiff' parameter combinations. Here we study the converse problem, showing that direct parameter measurements are very inefficient at constraining the model's behavior. To yield effective predictions such measurements must be very precise and complete; even a single imprecise parameter often destroys predictivity. We also show here that the characteristic sloppy eigenvalue pattern is reproduced in sixteen other diverse models from the systems biology literature. The apparent universality of sloppiness suggests that predictions from most models will be very fragile to single uncertain parameters and that collective parameters fits can often yield tight predictions with loose parameters. Together these results argue that focusing on parameter values may be a very inefficient route to useful models.

2.3 Introduction

Dynamic computational models are powerful tools for developing and testing hypotheses about complex biological systems [19, 20, 21]. It has even been suggested that such models will soon replace databases as the primary means for exchanging biological knowledge [22]. A major challenge with such models, however, is that they often possess tens or even hundreds of free parameters whose values can significantly affect model behavior [23, 24]. While high-throughput methods for discovering interactions are well-developed [25], high-throughput methods for measuring biochemical parameters remain limited [26]. Furthermore, using values measured in vitro in an in vivo application may introduce substantial inaccuracies [27, 28]. On the other hand, collectively fitting parameters [29, 30] by optimizing the agreement between the model and available data often yields large parameter uncertainties [9, 31, 32]. In approaches typically more focused on steady-state distributions of fluxes in metabolic networks, metabolic control analysis has been used to quantify the sensitivity of model behavior with respect to parameter variation [33], and flux-balance analysis and related techniques have probed the robustness of metabolic networks [34, 35].

One way to cope with the dearth of reliable parameter values is to focus on predictions that are manifestly parameter-independent [36], but these are mostly qualitative. An alternative is not to forsake quantitative predictions, but to accompany them with well-founded uncertainty estimates based on an ensemble of parameter sets statistically drawn from all sets consistent with the available data [7]. (Uncertainties in the model structure itself may be important in some cases. Here we focus on parameter uncertainties, as they are often important on their own.)

Brown et al. took this approach in developing a computational model of the well-studied growth-factor-signaling network in PC12 cells [8]. They collectively

fit their model's 48 parameters to 68 data points from 14 cell-biology experiments (mostly Western blots). After the fit, all 48 parameters had large uncertainties; their 95% confidence intervals each spanned more than a factor of 50. Surprisingly, while fitting this modest amount of data did not tightly constrain any single parameter value, it did enable usefully tight quantitative predictions of behavior under interventions, some of which were verified experimentally.

In calculating their uncertainties, Brown et al. found that the quantitative behavior of their model was much more sensitive to changes in certain combinations of parameters than others. Moreover, the sensitivity eigenvalues were approximately equally spaced in their logarithm, a pattern deemed 'sloppy'. Such sloppy sensitivities were subsequently seen in other multi-parameter fitting problems, from interatomic potentials [10] to sums of exponentials [11]. The fact that sloppiness arises in such disparate contexts suggests that it may be a universal property of nonlinear multi-parameter models. (Here the term 'universal' has a technical meaning from statistical physics, denoting a shared common property with a deep underlying cause; see [11]. Universality in this sense does not imply that all models must necessarily share the property.)

In this work, we begin by empirically testing seventeen systems biology models from the literature, examining the sensitivity of their behavior to parameter changes. Strikingly, we find that Brown et al's model is not unique in its sloppiness; every model we examine exhibits a sloppy parameter sensitivity spectrum. (Thus, in the models we've examined sloppiness is also universal in the common English sense of ubiquity.) We then study the implications of sloppiness for constraining parameters and predictions. We argue that obtaining precise parameter values from collective fits will remain difficult even with extensive time-series data, because the behavior of a sloppy model is very insensitive to many parameter

combinations. We also argue that, to usefully constrain model predictions, direct parameter measurements must be both very precise and complete, because sloppy models are also conversely very sensitive to some parameter combinations. Tests over our collection of models support the first prediction, and detailed analysis of the model of Brown et al. supports the second contention.

Sloppiness, while not unique to biology, is particularly relevant to biology, because the collective behavior of most biological systems is much easier to measure in vivo than the values of individual parameters. Much work has focused on optimizing experimental design to best constrain model parameters with collective fits [37, 38, 39]. We argue against this focus on parameter values, particularly when our understanding of a system is tentative and incomplete. Concrete predictions can be extracted from models long before their parameters are even roughly known [8], and when a system is not already well-understood, it can be more profitable to design experiments to directly improve predictions of interesting system behavior [12] rather than to improve estimates of parameters.

2.4 Results

2.4.1 Systems Biology Models have Sloppy Sensitivity Spectra

Our collection of 17 systems biology models [20, 38, 40, 41, 42, 43, 44, 45, 46, 47, 8, 48, 49, 50, 51, 52, 53] was drawn primarily from the BioModels database [54], an online repository of models encoded in the Systems Biology Markup Language (SBML) [55]. The collected models encompass a diverse range of biological systems, including circadian rhythm, metabolism, and signaling. All the models are formu-

lated as systems of ordinary differential equations, and they range from having about ten to more than two hundred parameters. In most cases, these parameters were not systematically fit or measured in the original publication.

We quantified the change in model behavior as parameters θ varied from their published values θ^* by the average squared change in molecular species time courses:

$$\chi^2(\theta) \equiv \frac{1}{2\,N_c\,N_s} \sum_{s,c} \frac{1}{T_c} \int_0^{T_c} \left[\frac{y_{s,c}(\theta,t) - y_{s,c}(\theta^*,t)}{\sigma_s} \right]^2 \mathrm{d}t, \tag{2.1}$$

a kind of continuous least-squares fit of parameters θ to 'data' simulated from published parameters θ^*. Here $y_{s,c}(\theta,t)$ is the time course of molecular species s given parameters θ in condition c, and T_c is the 'measurement' time for that condition. We took the species normalization σ_s to be equal to the maximum value of species s across the conditions considered; other consistent normalizations yield the same qualitative conclusions.

For each model, the sum in Equation 2.1 runs over all molecular species in the model and (except where infeasible) over all experimental conditions considered in the corresponding paper—an attempt to neutrally measure system behavior under conditions deemed significant by the original authors. (The total number of conditions and species are denoted by N_c and N_s, respectively.) SBML files and SloppyCell [56] scripts for all models and conditions are available in Dataset S1.

To analyze each model's sensitivity to parameter variation, we considered the Hessian matrix corresponding to χ^2:

$$H^{\chi^2}_{j,k} \equiv \frac{d^2\chi^2(\theta)}{d\log\theta_j\, d\log\theta_k}. \tag{2.2}$$

We took derivatives with respect to $\log\theta$ to consider *relative* changes in parameter values, because biochemical parameters can have different units and widely varying scales. Analyzing H^{χ^2} corresponds to approximating the surfaces of constant model behavior deviation (as quantified by χ^2) to be N_p-dimensional ellipsoids, where N_p

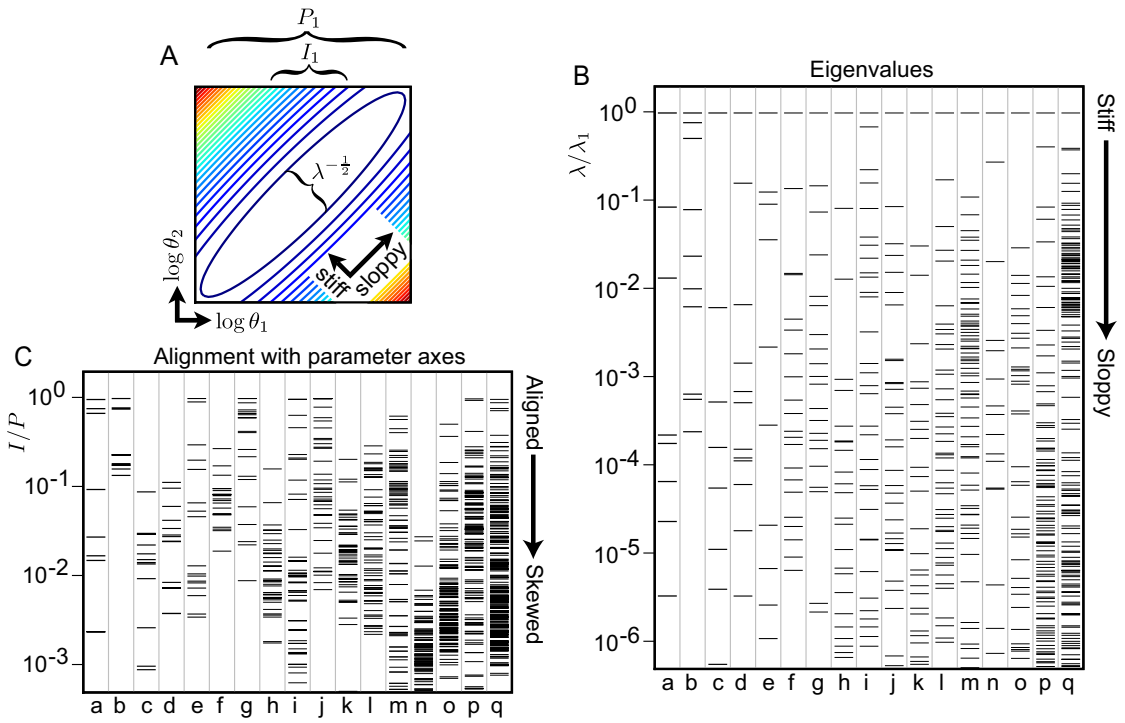


Figure 2.1: Subfigure A illustrates the quantities we calculate from H^{χ^2}, while subfigures B and C show that all the models we examined have sloppy sensitivity spectra. A: Analyzing H^{χ^2} corresponds to approximating the surfaces of constant model behavior change (constant χ^2) as ellipsoids. The width of each principal axis is proportional to one over the square root of the corresponding eigenvalue. The inner ellipsoid's projection onto and intersection with the $\log\theta_1$ axis are denoted P_1 and I_1, respectively. B: Plotted are the eigenvalue spectra of H^{χ^2} for our collection of systems biology models. The many decades generally spanned indicate the ellipses have very large aspect ratio. (The spectra have each been normalized by their largest eigenvalue. Not all values are visible for all models.) C: Plotted is the spectrum of I/P for each parameter in each model in our collection. Generally very few parameters have $I/P \approx 1$, suggesting that the ellipses are skewed from the bare parameter axes. (Not all values are visible for all models.) The models are ordered by increasing number of free parameters and are: (a) eukaryotic cell cycle [40], (b) Xenopus egg cell cycle [41], (c) eukaryotic mitosis [42], (d) generic circadian rhythm [43], (e) nicotinic acetylcholine intra-receptor dynamics [44], (f) generic kinase cascade [45], (g) Xenopus Wnt signaling [46], (h) Drosophila circadian rhythm [47], (i) rat growth-factor signaling [8], (j) Drosophila segment polarity [48], (k) Drosophila circadian rhythm [49], (l) Arabidopsis circadian rhythm [20], (m) in silico regulatory network [38], (n) human purine metabolism [50], (o) E. coli carbon metabolism [51], (p) budding yeast cell cycle [52], (q) rat growth-factor signaling [53].

is the number of parameters in the model. Figure 2.1A schematically illustrates these ellipsoids and some of their characteristics. (Details of calculating H^{χ^2} and related quantities are found in Methods. Dataset S1 includes H^{χ^2} for each model.)

The principal axes of the ellipsoids are the eigenvectors of H^{χ^2}, and the width of the ellipsoids along each principal axis is proportional to one over the square root of the corresponding eigenvalue. The narrowest axes are called 'stiff', and the broadest axes 'sloppy' [7]. The eigenvalue spectra for the models in our collection are shown in Figure 2.1B (each normalized by its largest eigenvalue). In every case, the eigenvalues span many decades. All but one span more than 10^6, indicating that the sloppiest axes of the ellipsoids illustrated in Figure 2.1A are generally more than one thousand times as long as the stiffest axes. In each spectrum the eigenvalues are also approximately evenly spaced in their logarithm; there is no well-defined cutoff between 'important' and 'unimportant' parameter combinations.

The Hessian matrix is a local quadratic approximation to the generally nonlinear χ^2 function. Principal component analysis of extensive Monte Carlo runs in the Brown et al. model, however, indicates that the sloppiness revealed by H^{χ^2} is indicative of full nonlinear χ^2 function [7].

Along with their relative widths, the degree to which the principal axes of the ellipsoids are aligned to the bare parameter axes is also important. We estimated this by comparing the ellipsoids' intersections I_i with and projections P_i onto each bare parameter axis i. If $I_i/P_i = 1$ then one of the principal axes of the ellipsoids lies along bare parameter direction i. Figure 2.1C plots the I/P spectrum for each model. In general, very few axes have $I/P \approx 1$; the ellipses are skewed from single parameter directions.

Naively, one might expect the stiff eigenvectors to embody the most important parameters and the sloppy directions to embody parameter correlations that might

suggest removable degrees of freedom, simplifying the model. Empirically, we have found that the eigenvectors often tend to involve significant components of many different parameters; plots of the five stiffest eigenvectors for each model are in Section 2.S1. This is understandable theoretically; the nearly-degenerate sloppy eigenvectors should mix, and the stiff eigenvectors can include arbitrary admixtures of unimportant directions to a given important parameter combination. (Indeed, in analogous random-matrix theories the eigenvectors are known to be uncorrelated random vectors [57].) While the relatively random eigenvectors studied here may not be useful in guiding model reduction, more direct explorations of parameter correlations have yielded interesting correlated parameter clusters [58].

These characteristic parameter sensitivities that evenly span many decades and are skewed from bare parameter axes define a 'sloppy' model [7]. Figures 2.1B and 2.1C show that every model we have examined has a sloppy sensitivity spectrum. Next we discuss some broad questions about the relation between model predictions, collective fits, and parameter measurements and see how the sloppy properties of these models may suggest answers.

2.4.2 Consequences of Sloppiness

The difficulty of extracting precise parameter values from collective fits in systems biology modeling is well-known [39]. Sloppiness offers an explanation for this and predicts that it will be true even for fitting to complete data that the model can fit perfectly. In a collective fit, the parameter set ensemble samples from all sets of parameters for which the model behavior is consistent with the data. Because sloppy models are very insensitive to parameter combinations that lie along sloppy directions, the parameter set ensemble can extend very far in those directions, as illustrated schematically in Figure 2.2A. As a result, individual parameters can be

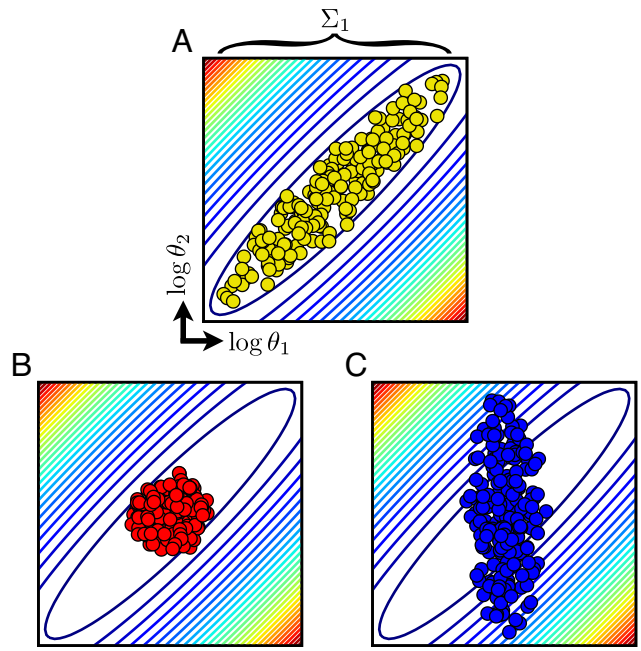


Figure 2.2: As in Figure 2.1A, the contours represent surfaces of constant model behavior deviation. The clouds of points represent parameter set ensembles.
A: Collective fitting of model parameters naturally constrains the parameter set ensemble along stiff directions and allows it to expand along sloppy directions. The resulting ensemble may be very large, yet encompass little variation in model behavior, yielding small prediction uncertainties despite large parameter uncertainties. (Σ_1 denotes the 95% confidence for the value of θ_1.)
B: If all parameters are directly measured to the same precision, the parameter set ensemble is spherical. The measurement precision required for well-constrained predictions is set by the stiffest direction.
C: If one parameter (here θ_2) is known less precisely than the rest, the cloud is ellipsoidal. If not aligned with a sloppy direction, the cloud will admit many model behaviors and yield large prediction uncertainties. (Note that the aspect ratio of the real contours can be greater than 1000.)

very poorly determined (e.g., confidence interval indicated by Σ_1 in Figure 2.2A). Below we discuss a test of this prediction over all the models in our collection.

Unless one has direct interest in the kinetic constants for the underlying reactions, uncertainties in model predictions are generally more important than uncertainties in model parameters. The parameter set ensemble illustrated in Figure 2.2A yields large uncertainties on individual parameters, but can yield small uncertainties on predictions. While the fitting process allows the ensemble to expand along sloppy directions, the fit naturally constrains the ensemble along stiff directions, so that model behavior varies little within the ensemble, and predictions can be consequently tight.

Direct parameter measurements, on the other hand, will have uncertainties that are uncorrelated with the model's underlying stiff and sloppy directions. For example, if all parameter measurements are of the same precision, the parameter set ensemble is spherical, as illustrated in Figure 2.2B. For tight predictions, this ensemble must not cross many contours, so the required precision is set by the stiffest direction of the model. Consequently, high precision parameter measurements are required to yield tight predictions. Moreover, these measurements must be complete. If one parameter is known less precisely, the parameter set ensemble expands along that parameter axis, as illustrated in Figure 2.2C. If that axis is not aligned with a sloppy direction, model behavior will vary dramatically across the parameter set ensemble and predictions will have correspondingly large uncertainties. Below we discuss tests of both these notions, exploring the effects of direct parameter measurement uncertainty on predictions of a particular model.

2.4.2.1 Parameter Values from Collective Fits

Does the sloppiness of these models really prevent one from extracting parameters from collective fits? Here we discuss a test of this prediction using an idealized

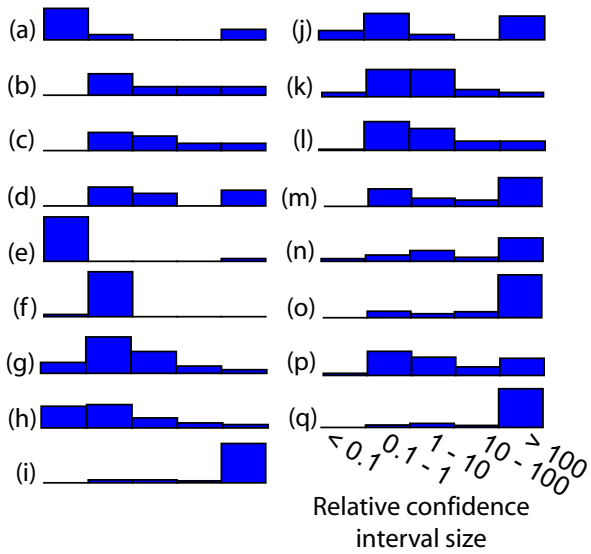


Figure 2.3: Shown are histograms of the relative confidence interval size Σ for each parameter in each model of our collection, after fitting 100 times as many time-series data points (each with 10% uncertainty) as parameters. In most cases a large number of parameters are left with greater than 100% uncertainty. (A parameter constrained with 95% probability to lie between 1 and 100 would have $\Sigma \approx 100$.)
Labels are as in Figure 2.1.

fitting procedure.

Our χ^2 measure of model behavior change (Equation 2.1) corresponds to the cost function for fitting model parameters to continuous time-series data that the model fits perfectly at parameters θ^*; H^{χ^2} is the corresponding Fisher information matrix (Equation 2.2). We used this idealized situation to test the prediction that collective fits will often poorly constrain individual parameters in our collection of sloppy models.

We defined the relative 95% confidence interval size Σ_i as the ratio between parameter i at the upper and lower extremes of the interval, minus one. (A parameter value constrained after the fit to lie between 10 and 1000 would have $\Sigma \approx 100$, while one constrained between 1.0 and 1.5 would have $\Sigma = 0.5$.) We assumed 100 times as many data points (each with 10% uncertainty) as the number of parameters in each model. Figure 2.3 shows histograms of the quadratic approximation to Σ for each parameter in each model after fitting such data. (See Methods.) For most of the models, the figure indicates that such fitting leaves many parameters with greater than 100% uncertainty ($\Sigma > 1$). Indeed, even fitting this large amount of ideal data can leave many parameter values very poorly determined, as expected from the sloppiness of these models and our discussion of Figure 2.2A.

The fact that nonlinear multiparameter models often allow a wide range of correlated parameters to fit data has long been appreciated. As one example, a 1987 paper by Brodersen et al. on ligand binding to hemoglobin and albumin empirically found many sets of parameters that acceptably fit experimental data, with individual parameter values spanning huge ranges [9]. Our sloppy model perspective ([7, 8, 11], Figure 1) shows that there is a deep underlying universal pattern in such least-squares fitting. Indeed, an analysis of the acceptable binding parameter sets from the 1987 study shows the same characteristic sloppy eigenvalue

spectrum we observed in Figure 2.1B (Section 2.S5).

2.4.2.2 Predictions from Direct Parameter Measurements

Figures 2.2B and 2.2C suggests that direct parameter measurements must be both precise and complete to usefully constrain predictions in sloppy systems. Here we discuss a test of this notion in a specific model.

We worked with the 48-parameter growth-factor-signaling model of Brown et al., shown schematically in Figure 2.4A [8]. The parameters in this model were originally collectively fit to 14 time-series cell-biology experiments. We focused on this model because it is instructive to compare our results concerning direct parameter measurements with prior results from collective fitting. For our analysis, we assumed that hypothetical direct parameter measurements would be centered about the original best-fit values.

One important test of the model was a prediction of the time-course of ERK activity upon EGF stimulation, given inhibition of the PI3K branch of the pathway. The yellow shaded region in Figure 2.4B shows the uncertainty bound on this prediction from the original collective fit, calculated by exhaustive Monte Carlo [8]. The tightness of this prediction is remarkable considering the huge uncertainties the collective fit left in the individual parameter values (yellow circles in Figure 2.4C). Not a single parameter was constrained to better than a factor of 50.

How precise would direct parameter measurements have to be to yield as tight a prediction as the collective fit? For this prediction, the PI3K branch (inhibited) and C3G branch (NGF-dependent) of the pathway are irrelevant in the model; the remaining reactions involve 24 parameters. To achieve the red prediction in Figure 2.4B, all 24 involved parameters must be measured to within a factor of plus or minus 25% (Figure 2.4C, red squares). With current techniques, measuring even a single in vivo biochemical parameter to such precision would be a challenging

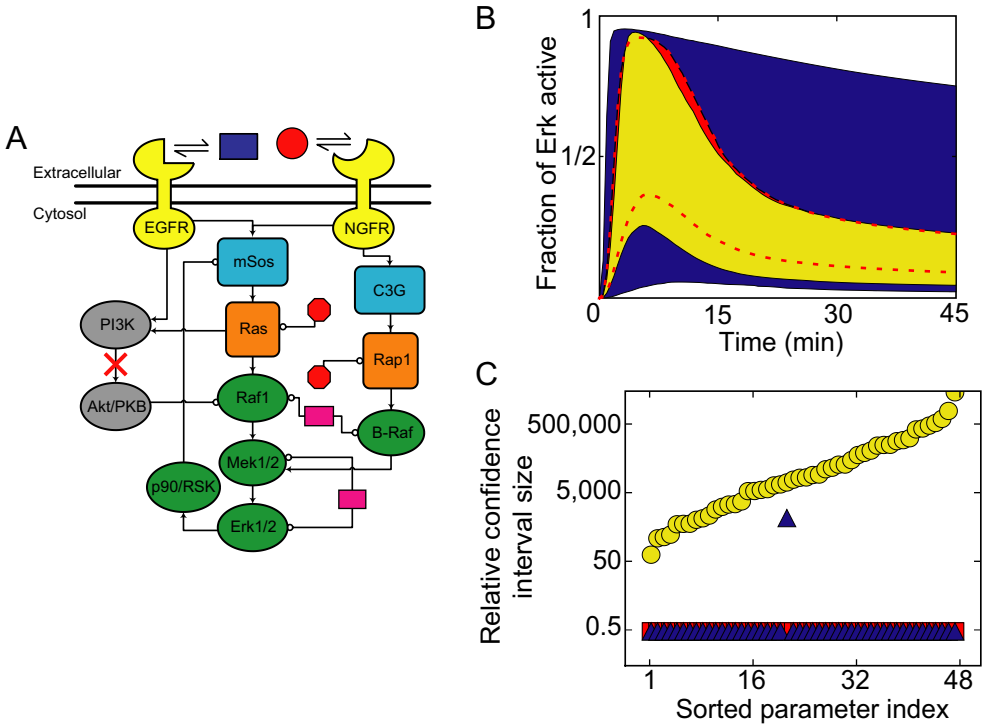


Figure 2.4: A: Our example prediction is for ERK activity upon EGF stimulation given PI3K inhibition in this 48-parameter model of growth-factor-signaling in PC12 cells [8].
B: Shaded regions are 95% confidence intervals calculated via exhaustive Monte Carlo for our example prediction given various scenarios for constraining parameter values.
C: Plotted is the relative size Σ of the 95% confidence interval for each parameter. The scenarios represented are: (red, squares) all model parameters individually measured to high precision, (blue, triangles) all parameters precisely measured, except one estimated to low precision, (yellow, circles) all parameters collectively fit to 14 real cell-biology experiments. Precisely measured individual parameter values enable a tight prediction (B: middle red band), but even one poorly known parameter can destroy predictive power (B: wide blue band). In contrast, the collective fit yields a tight prediction (B: tightest yellow band) but only very loose parameter constraints (C: circles). The large parameter uncertainties from the collective fit (C: circles) calculated here by Monte Carlo are qualitatively similar to those seen in the linearized fit to idealized data (Figure 2.3, model (i)). (For clarity, the dashed red lines trace the boundary of the red confidence interval.)

experiment. Such high precision is required because, as illustrated in Figure 2.2B, the measurements need to constrain the stiffest combination of model parameters.

What if a single parameter is left unmeasured? For example, consider high precision measurements of 23 of the 24 involved parameters, all but the rate constant for the activation of Mek by Raf1. For this unmeasured parameter, we assumed that an informed estimate could bound it at 95% confidence to within a total range of 1000 (e.g., between $1s^{-1}$ and $1000s^{-1}$). The resulting prediction (blue in Figure 2.4B) has very large uncertainty and would likely be useless. Note that these hypothetical measurements constrain every individual parameter value more tightly than the original collective fit (blue triangles versus yellow circles in Figure 2.4C), yet the prediction is much less well-constrained. Neither this parameter nor this prediction is unique. Uncertainty for this prediction is large if any one of about 18 of the 24 involved parameters is unmeasured (Section 2.S2). Furthermore, other possible predictions in this model are similarly fragile to single unmeasured parameters (Section 2.S3).

To usefully constrain Brown et al.'s model, direct parameter measurements would need to be both precise and complete. By contrast, collective parameter fitting yielded tight predictions with only a modest number of experiments. These results are expected given the model's sloppiness.

2.5 Discussion

By examining seventeen models from the systems biology literature [20, 38, 40, 41, 42, 43, 44, 45, 46, 47, 8, 48, 49, 50, 51, 52, 53], we showed that their parameter sensitivities all share striking common features deemed 'sloppiness'; the sensitivity eigenvalues span many decades roughly evenly (Figure 2.1B), and tend not to be aligned with single parameters (Figure 2.1C). We argued that sloppy parameter

sensitivities help explain the difficulty of extracting precise parameter estimates from collective fits, even from comprehensive data. Additionally, we argued that direct parameter measurements should be inefficient at constraining predictions from sloppy models. We then showed that collective parameter fits to complete time-series data do indeed yield large parameter uncertainties in our model collection (Figure 2.3). Finally, we confirmed for the 48-parameter signaling model of Brown et al. [8] that direct parameter measurements must be formidably precise and complete to usefully constrain model predictions (Figure 2.4).

What causes sloppiness? (1) Fundamentally, sloppiness involves an extraordinarily singular coordinate transformation in parameter space between the bare parameters natural in biology (e.g., binding affinities and rate constants) and the eigenparameters controlling system behavior, as discussed in [11]. Both experimental interventions and biological evolution work in the bare parameter space, so this parameterization is fundamental to the system, not an artifact of the modeling process. (2) Sloppiness depends not just upon the model, but also on the data it is fit to; exhaustive experiments designed to decouple the system and separately measure each parameter will naturally not yield sloppy parameter sensitivities. (3) In biological systems fit to time-series data, Brown and Sethna [7] note that sloppiness may arise due to under-determined systems, proximity to bifurcations, and separation of time or concentration scales, but they doubt that these can explain all the sloppiness found in their model. Our analysis includes complete data on all species, and hence is overdetermined. Small eigenvalues near bifurcations are associated with dynamic variables, and not the system parameters we investigate. To study the effect of time and concentration scales we calculated H^{χ^2} for a version of the Brown et al. model in which all concentrations and rate constants were scaled to one[1]. The resulting model remains sloppy, with eigenvalues roughly

[1]This analysis was suggested by Eric Siggia

uniformly spanning five decades (Section 2.S4). (4) Motivated by simple example systems, we have argued elsewhere that sloppiness emerges from a redundancy between the effects of different parameter combinations [11]. We are presently investigating decompositions of parameter space into sloppy subsystems [58] and the use of physically or biologically motivated nonlinear coordinate changes to remove sloppiness or motivate simpler models. These potential methods for model refinement, however, demand a complete and sophisticated understanding of the system that is unavailable for many biological systems of current interest.

Parameter estimation has been a serious obstacle in systems biology modeling. With tens of unknown parameters, a typical modeling effort might draw some values from the literature (possibly from in vitro measurements or different cell lines) [45, 50], set classes of constants to the same value (e.g., phosphorylation rates) [43, 44, 53], and adjust key parameters to qualitatively best fit the existing data [20, 49, 52]. In retrospect, these approaches may be successful because the models are sloppy—they can be tuned to reality by adjusting one key parameter per stiff direction, independently of how reliably the other parameters are estimated.

Computational modeling is a potentially invaluable tool for extrapolating from current experiments and distinguishing between models. But we cannot trust the predictions of these models without testing how much they depend on uncertainties in these estimated parameters. Conversely, if we insist upon a careful uncertainty analysis, it would seem unnecessary to insist upon tight prior estimates of the parameters, since they do not significantly enhance model predictivity. Because the behavior of a sloppy model is dominated by a few stiff directions that nonetheless involve almost all the parameters, direct parameter measurements constrain predictions much less efficiently than comparably difficult experiments probing collective system behavior.

Our suggestion of making predictions from models with very poorly known parameters may appear dangerous. A model with tens or hundreds of unmeasured parameters might seem completely untrustworthy; we certainly believe that any prediction derived solely from a best-fit set of parameters is of little value. Uncertainty bounds derived from rigorous sensitivity analysis, however, distinguish those predictions that can be trusted from those that cannot. Of course, successful fits and predictions may arise from models that are incorrect in significant ways; for example, one model pathway with adjusted parameters may account for two parallel pathways in the real system. A model that is wrong in some details may nevertheless be useful in guiding and interpreting experiments. For computational modeling to be useful in incompletely understood systems, we must focus not on building the final, perfect, model with all parameters precisely determined, but on building incomplete, tentative and falsifiable models in the most expressive and predictive fashion feasible.

Given that direct parameters measurements do not efficiently constrain model behavior, how do we suggest that experimentalists decide what experiment to do next? If the goal is to test the assumptions underlying a model, one should look for predictions with tight uncertainty estimates that can be readily tested experimentally. If the goal is to reduce uncertainty in crucial model predictions, one must invoke the statistical methods of optimal experimental design, which we have studied elsewhere [12] and which may be conveniently implemented in modeling environments that incorporate sensitivity analysis (such as SloppyCell [56]).

In our approach, the model and its parameters cannot be treated in isolation from the data that informed model development and parameter fitting. This complicates the task of exchanging knowledge in the modeling community. To support our approach, standards such as SBML [55] that facilitate automated model ex-

change will need to be extended to facilitate automated data exchange.

Every one of the 17 systems biology models we studied exhibits a sloppy spectrum of parameter sensitivity eigenvalues; they all span many decades roughly evenly and tend not be aligned with single parameters. This striking and apparently universal feature has important consequences for the modeling process. It suggests that modelers would be wise to try collective parameter fits and to focus not on the quality of their parameter values but on the quality of their predictions.

2.6 Methods

2.6.1 Hessian Computations

H^{χ^2} can be calculated as

$$H^{\chi^2}_{j,k} = \frac{1}{N_c\, N_s} \sum_{s,c} \frac{1}{T_c\, \sigma_s^2} \int_0^{T_c} \frac{d\, y_{s,c}(\theta^*, t)}{d \log \theta_j} \frac{d\, y_{s,c}(\theta^*, t)}{d \log \theta_k} \bigg|_{\theta^*} \mathrm{d}t. \tag{2.3}$$

Second derivative terms $(d^2\, y_{s,c}(\theta^*, t)/d \log \theta_i\, d \log \theta_j)$ might be expected, but they vanish because we evaluate H^{χ^2} at θ^*. Equation 2.3 is convenient because the first derivatives $d\, y_{s,c}(\theta^*, t)/d \log \theta_k$ can be calculated by integrating sensitivity equations. This avoids the use of finite-difference derivatives, which are troublesome in sloppy systems.

The projections P_i of the ellipsoids shown in Figure 2.2A onto bare parameter axis i are proportional to $\sqrt{\left(\operatorname{inv} H^{\chi^2}\right)_{i,i}}$. The intersections I_i with axis i are proportional to $\sqrt{1/H^{\chi^2}_{i,i}}$, with the same proportionality constant.

2.6.2 Parameter Uncertainties

To rescale H^{χ^2} so that it corresponds to fitting N_d data points, each with uncertainty a fraction f of the species' maximal value, we multiply H^{χ^2} by N_d/f^2. In the

quadratic approximation, the one-standard-deviation uncertainty in the logarithm of parameter θ_i after such a collective fit is given by $\sigma^2_{\log\theta_i} = (f^2/N_d)\left(\operatorname{inv} H^{\chi^2}\right)_{i,i}$. The relative size of the 95% confidence interval is then $\Sigma_i = \exp\left(4\sigma_{\log\theta_i}\right) - 1$.

2.6.3 Prediction Uncertainties

The red and blue prediction uncertainties shown in Figure 2.4B were calculated by randomly generating 1000 parameter sets consistent with the stated parameter uncertainties. (For each parameter θ_i, the logarithm of its value is drawn from a normal distribution with mean $\log\theta_i$ and standard deviation $\sigma_{\log\theta_i}$ specified by desired Σ.) For each parameter set, the Erk time course was calculated, and at each timepoint the shaded regions in the figure contain the central 95% of the time courses.

2.6.4 Software

All computations were performed in the open-source modeling environment SloppyCell, version 0.81 [56]. SBML files and SloppyCell scripts to reproduce all presented calculations are in Dataset S1.

2.7 Supporting Information

Text S1: Stiffest Eigenvectors

Text S2: Effect of Other Poorly Determined Parameters

Text S3: Fragility of Other Predictions

Text S4: Rescaled Model of Brown et al.

Text S5: Eigenvalue Analysis of Brodersen et al. Binding Studies

Dataset S1: SBML Files, SloppyCell Scripts, and χ^2-Hessians

2.7.1 Accession Numbers

Models discussed that are in the BioModels database [54] are:

(a) BIOMD0000000005, (c) BIOMD0000000003, (d) BIOMD0000000035, (e) BIOMD0000000002, (f) BIOMD0000000010, (h) BIOMD0000000021, (i) BIOMD0000000033, (k) BIOMD0000000022, (l) BIOMD0000000055, (n) BIOMD0000000015, (o) BIOMD0000000051, (p) BIOMD0000000056, (q) BIOMD0000000049.

2.8 Acknowledgments

We thank Eric Siggia for suggesting study of the rescaled model of Brown et al. We also thank Rick Cerione and Jon Erickson for sharing their biological insights and John Guckenheimer, Eric Siggia, and Kelvin Lee for helpful discussions about dynamical systems. Computing resources were kindly provided by the USDA-ARS plant pathogen systems biology group in Ithaca, NY. Finally, we thank several anonymous reviewers whose comments strengthened the manuscript.

2.9 Funding

RNG was supported by an NIH Molecular Biophysics Training Grant, T32-GM-08267. JJW was supported by a DOE Computational Science Graduate Fellowship. CRM acknowledges support from USDA-ARS project 1907-21000-017-05. This work was supported by NSF grant DMR-0218475.

2.S1 Stiffest Eigenvectors

The following figures show the four stiffest eigenvectors of H^{χ^2} (corresponding to the four largest eigenvalues) for each model in our collection. In each eigenvector the five parameters with the largest contributions are labeled. With few exceptions, the eigenvectors tend to be inscrutable combinations of many parameters, and they tend not to have immediately obvious biological interpretation.

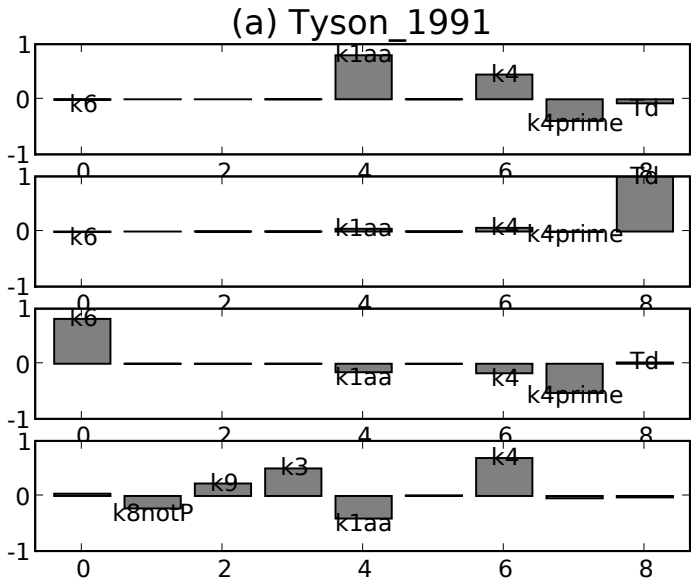


Figure 2.5: Model (a): Tyson's model of the eukaryotic cell cycle [40].

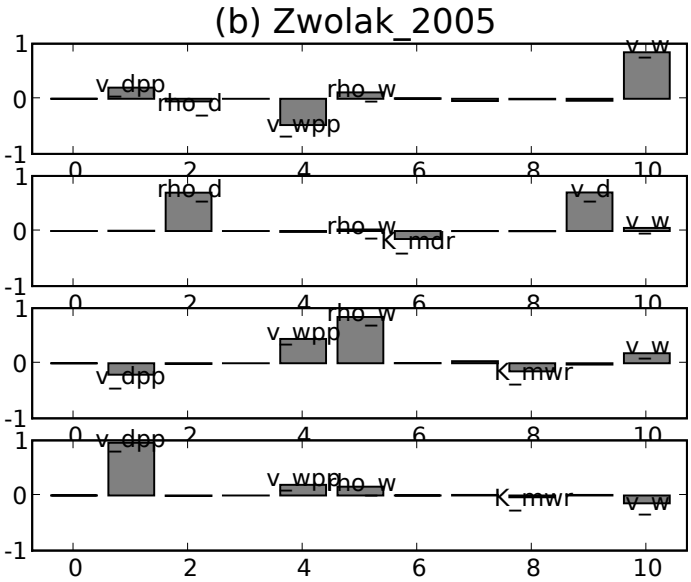


Figure 2.6: Model (b): Zwolak et al.'s model of the Xenopus egg cell cycle [41].

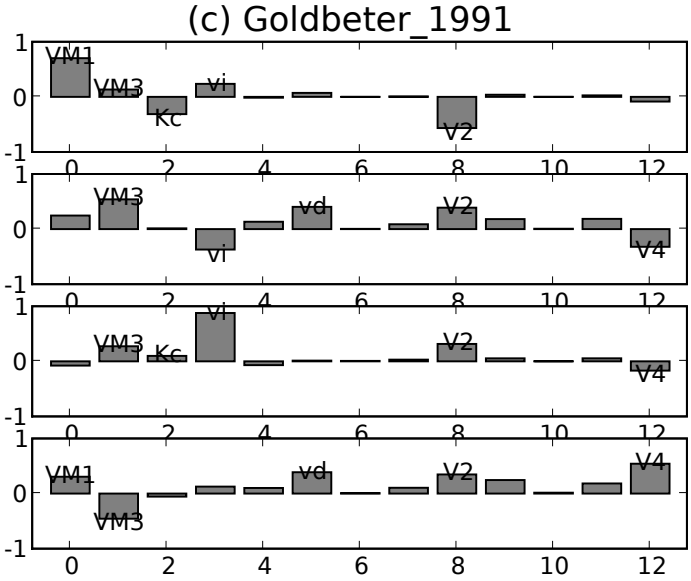


Figure 2.7: Model (c): Goldbeters's model of eukaryotic mitosis [42].

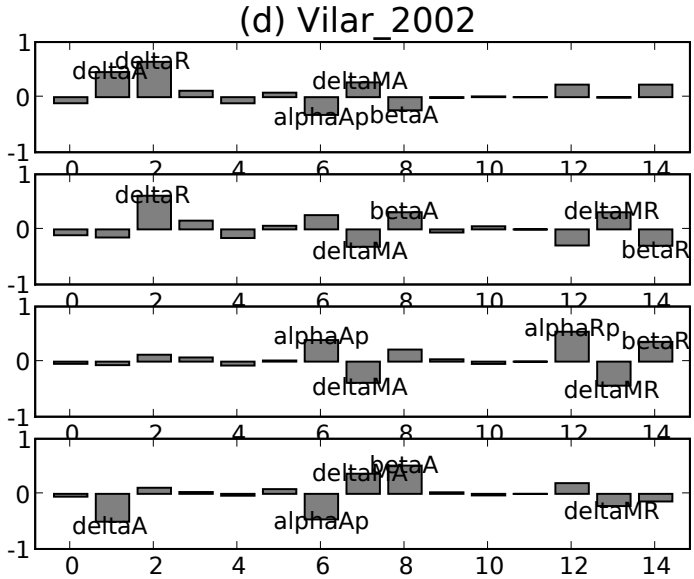


Figure 2.8: Model (d): Vilar et al.'s generic circadian rhythm model [43].

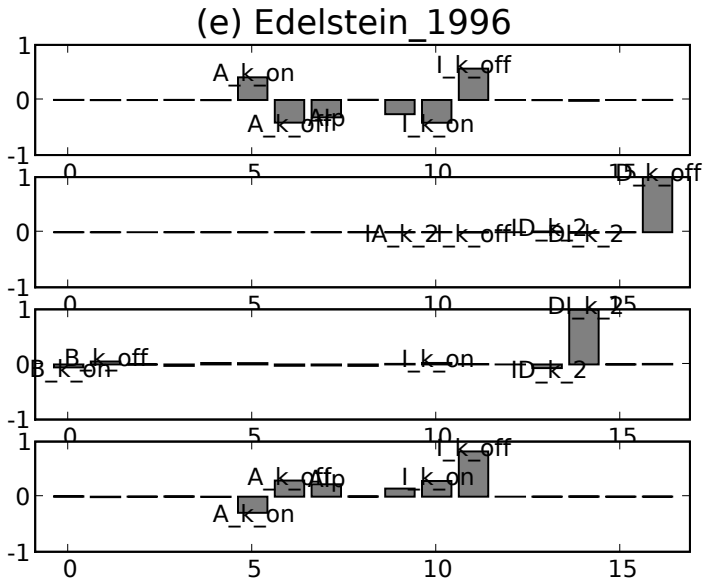


Figure 2.9: Model (e): Edelstein et al.'s model of nicotinic acetylcholine intra-receptor dynamics [44].

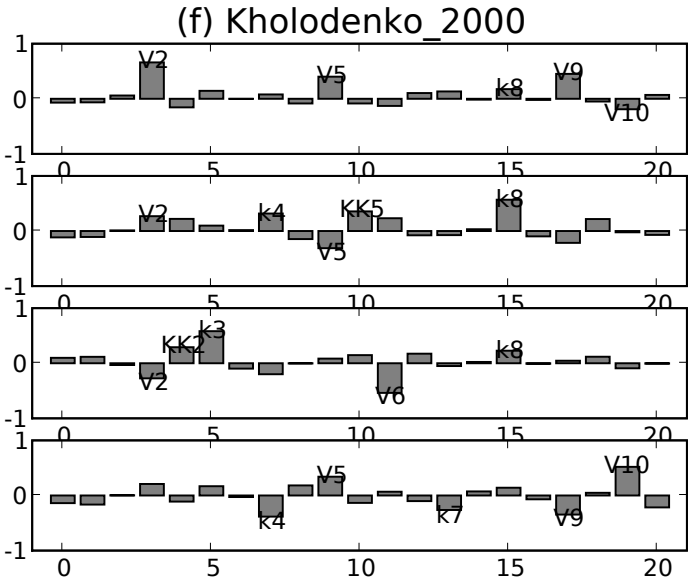


Figure 2.10: Model (f): Kholodenko's model of a generic kinase cascade [45].

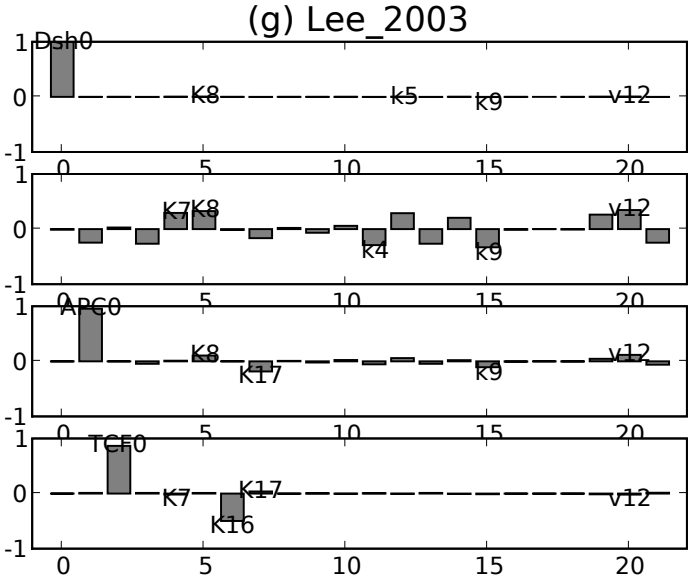


Figure 2.11: Model (g): Lee et al.'s model of Xenopus Wnt signaling [46].

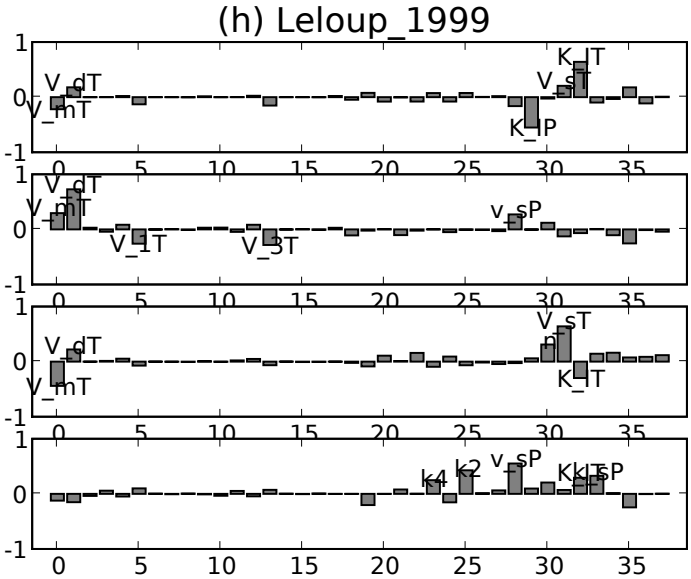


Figure 2.12: Model (h): Leloup and Goldbeters's model of Drosophila circadian rhythm [47].

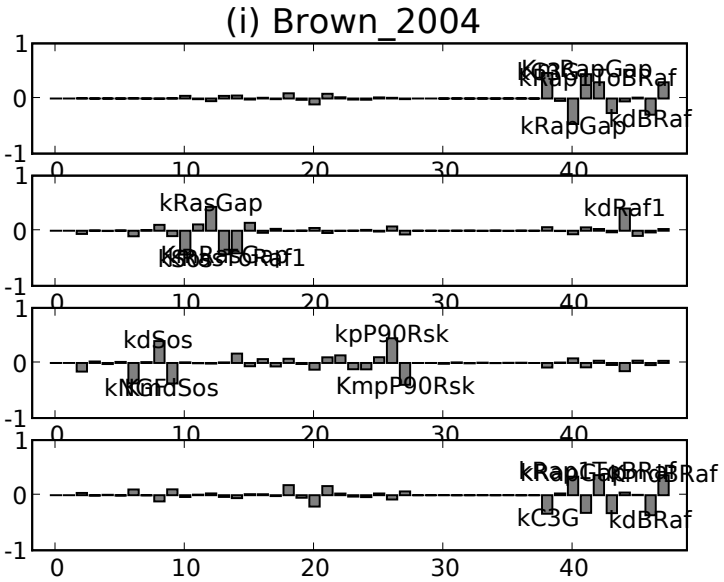


Figure 2.13: Model (i): Brown et al.'s model of rat growth-factor signaling [8].

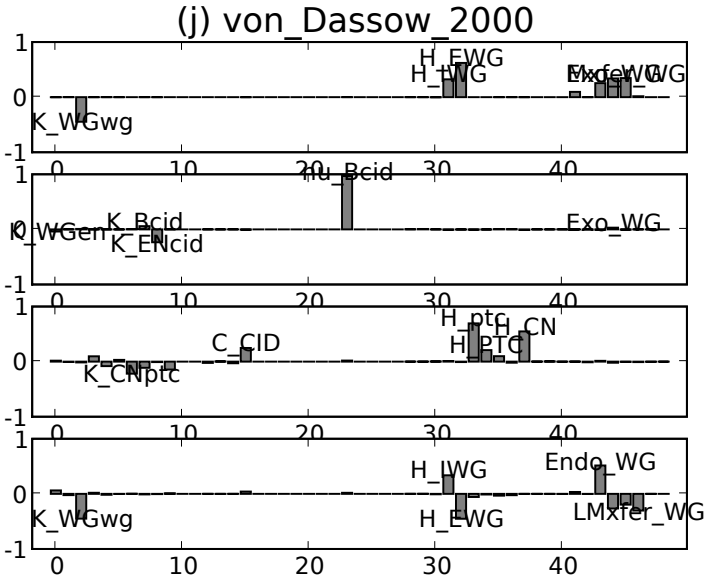


Figure 2.14: Model (j): von Dassow et al.'s model of the Drosophila segment polarity network [48].

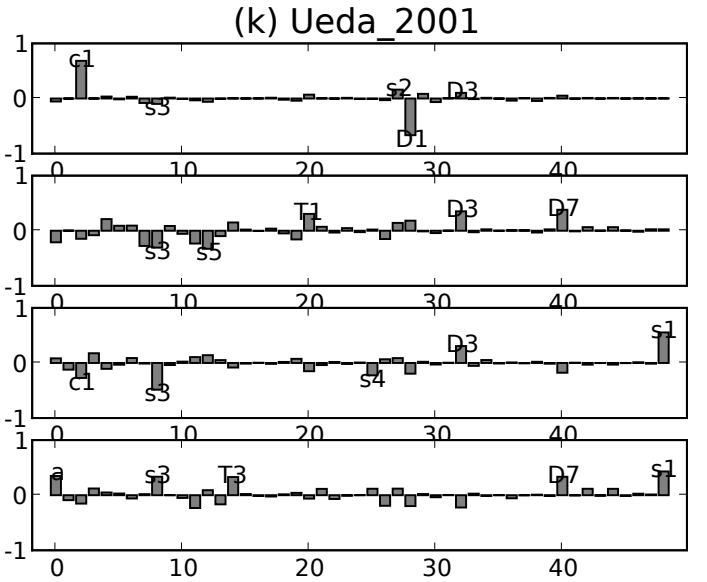


Figure 2.15: Model (k): Ueda et al.'s model of Drosophila circadian rhythm [49].

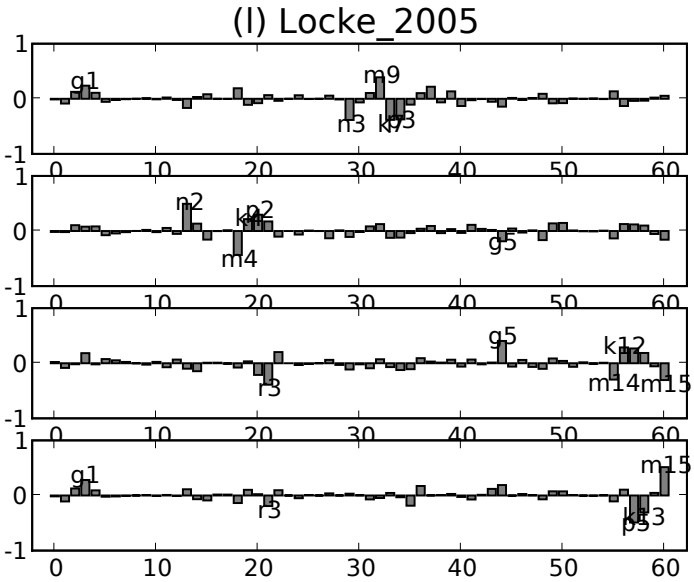


Figure 2.16: Model (l): Locke et al.'s model of Arabidopsis circadian rhythm [20].

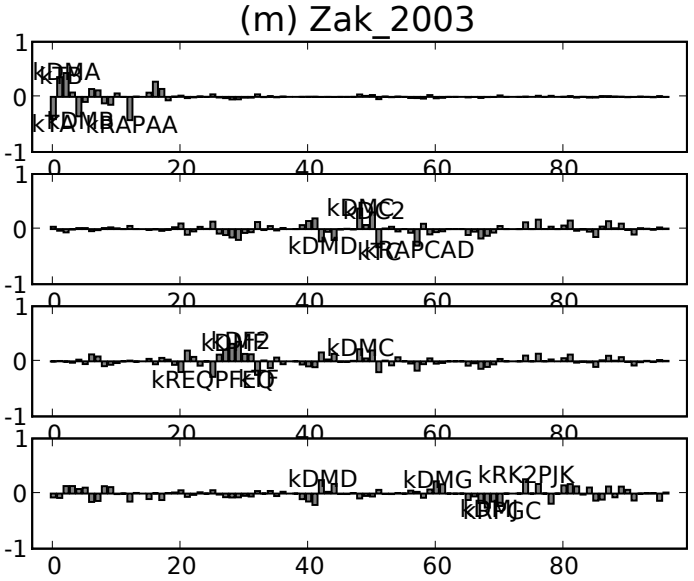


Figure 2.17: Model (m): Zak et al.'s model of an in silico regulatory network [38].

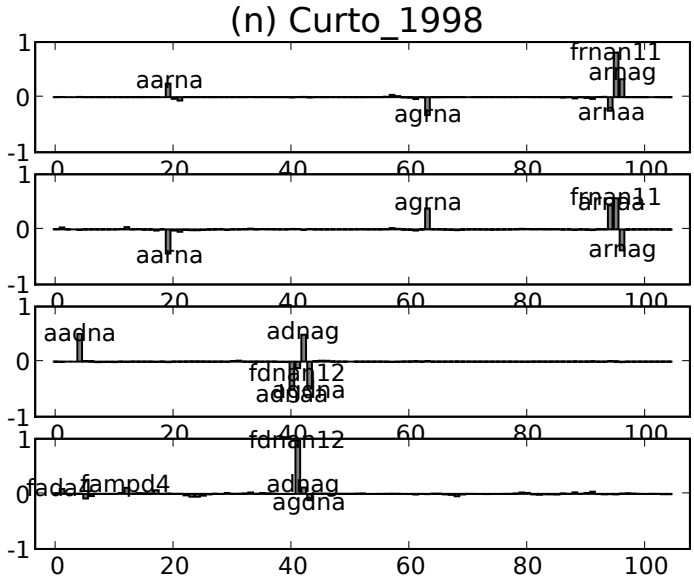


Figure 2.18: Model (n): Curto et al.'s model of human purine metabolism [50].

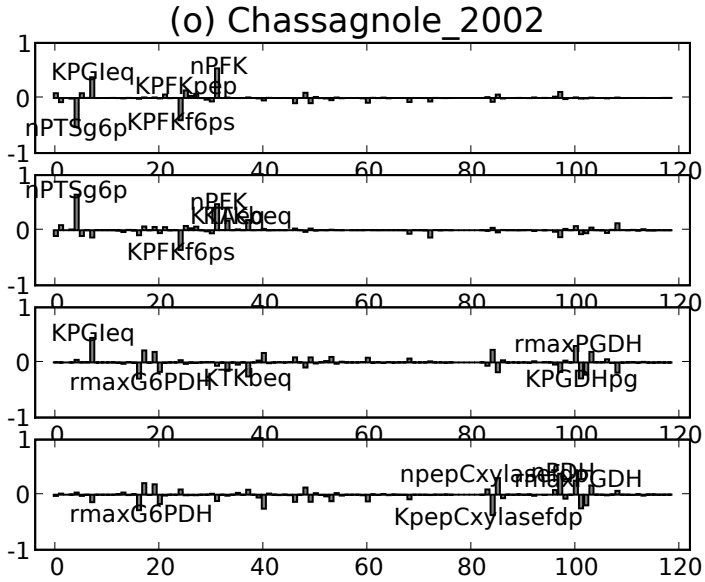


Figure 2.19: Model (o): Chassagnole et al.'s model of E. coli carbon metabolism [51].

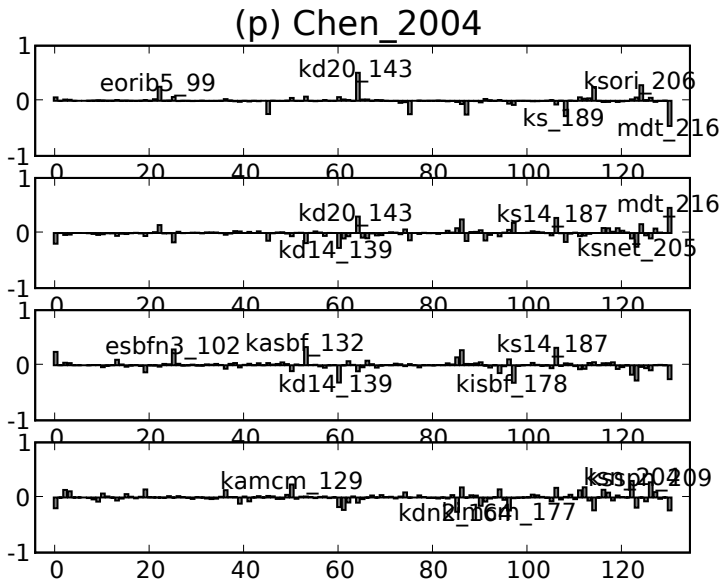


Figure 2.20: Model (p): Chen et al.'s model of the budding yeast cell cycle [52].

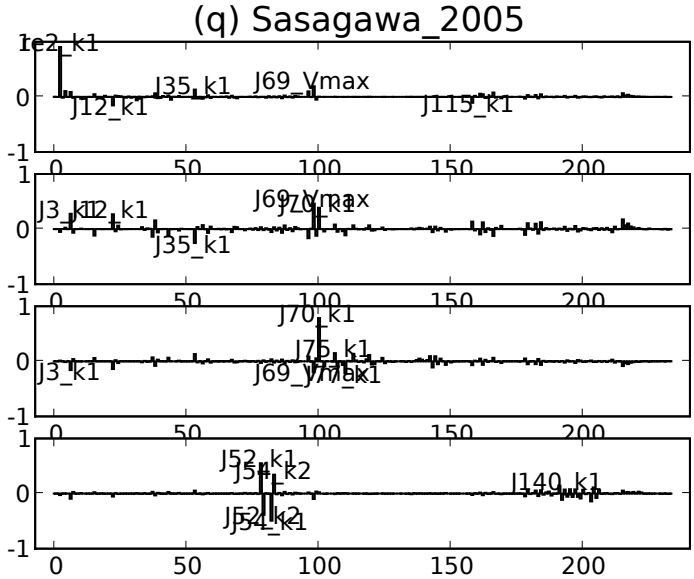


Figure 2.21: Model (q): Sasagawa et al.'s model of rat growth-factor signaling [53].

2.S2 Effect of Other Poorly Determined Parameters

The subfigures in Figure 2.22 correspond to Figure 2.4 , for missing the measurement of each of the 24 model parameters involved in our prediction. (Uncertainty in those parameters involved in the inhibited PI3K branch or in NGF-dependent C3G branch can have no effect on this prediction.) The blue regions are 95% confidence intervals given a single poorly determined parameter. For comparison, shown in red is the 95% confidence interval from precisely determining all parameters. For the exact role of each parameter in the model, see the original model paper [8].

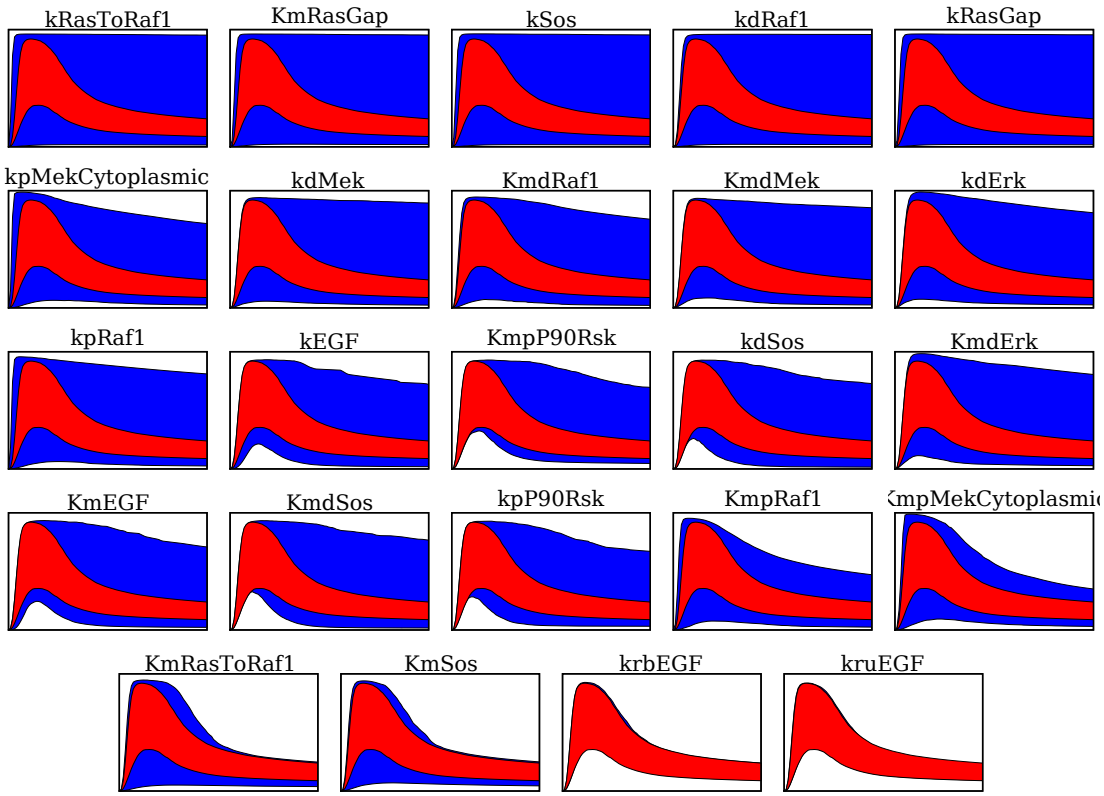


Figure 2.22: Effect of other missing parameters on our example prediction.

2.S3 Fragility of Other Predictions

Figures 2.23 and 2.24 show the 95% confidence intervals for two other example predictions, as in Figure 2.4 . The red intervals correspond to measuring all parameters to within plus or minus 25% (at 95% confidence). The blue intervals correspond to measuring all parameters as before, except for one, which is estimated to within a total range of 1000. The yellow intervals correspond to the collective parameter fit from Brown et al. [8].

Figure 2.23 shows a prediction of the activity of Ras given EGF stimulation in a wild-type cell. Missing a measurement of the rate constant for activation of p90 by Erk yielded the large blue interval.

Figure 2.23 shows a prediction of Mek activity given NGF stimulation of a wild-type cell, with the blue region corresponding to a missing measurement of the rate constant for the activation of B-Raf by Rap1. In this case the collective fit gave only an upper bound on the activity of Mek, so precisely measuring each individual parameter would yield a stronger prediction than the collective fit. (A measurement of zero Mek activity upon stimulation would be consistent with the model as constrained by the collective fit, but inconsistent the model as constrained by direct parameter measurements.) Nevertheless, the prediction with one missing parameter measurement remains much less informative than the collective fit.

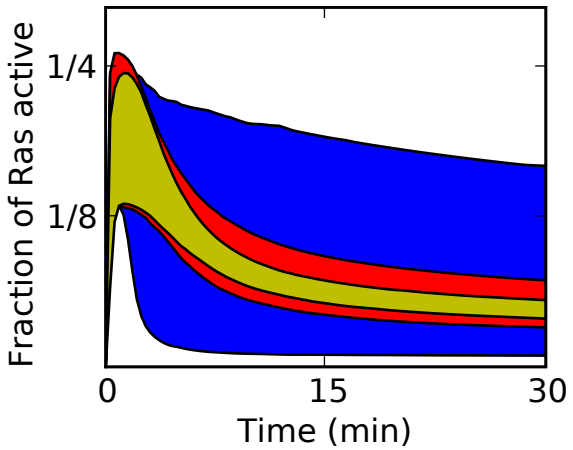


Figure 2.23: Prediction uncertainties for Ras activity given EGF stimulation.

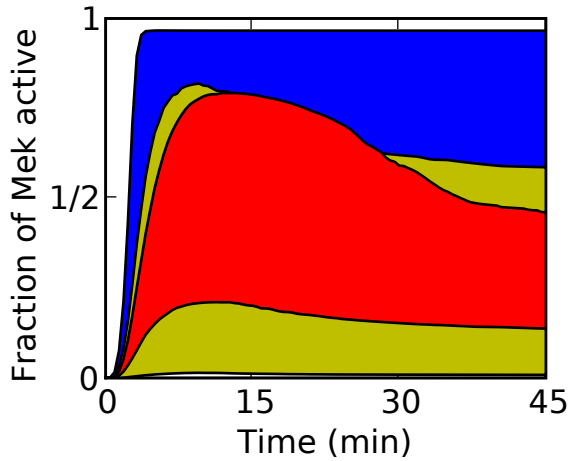


Figure 2.24: Prediction uncertainties for Mek activity given NGF stimulation.

2.S4 Rescaled Model of Brown et al.

To study the effect of time and concentration scales on sloppiness, we consider two modified versions of the model of Brown et al. [8].

In the first ('Rescaled'), we attempt to adjust concentration and time scales while maintaining fit quality. All concentrations are scaled to one, and all Michaelis constants are set to one. Additionally, the binding of EGF to its receptor is set to equilibrium, to remove one known sloppy mode. (NGF binding must be slow to fit the experimental data.) All rate constants are then re-optimized, adding an additional constraint on their total range. The resulting fit to the data has an approximately 50% higher cost. The eigenvalues of H^{χ^2} for this version of the model are shown in the central column of figure 2.25.

To fully remove the effects of time and concentration scales in the model, we set all non-zero initial conditions to 1 and all parameters to 1 ('All One'). The resulting eigenvalues of H^{χ^2} are plotted in the right column in figure 2.25.

Note that both adjusted models remain sloppy, with eigenvalues roughly evenly spaced over many decades.

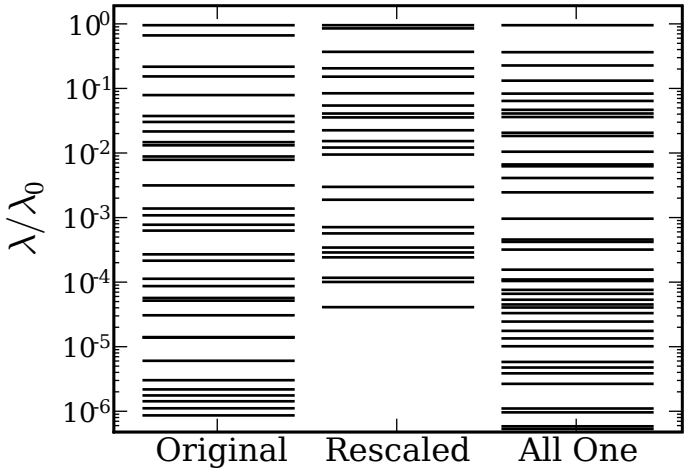


Figure 2.25: Eigenvalue for rescaled Brown et al. models.

2.S5 Sloppy Model Analysis of Brodersen et al. Binding Studies

Brodersen et al. [9] studied the equilibrium binding of salicylate to human serum albumin and of oxygen to hemoglobin. In both cases, they fit a binding model by least-squares to data consisting of ligand per protein versus free ligand concentration. They repeated the fitting procedure 30 times in each case, terminating each optimization once a parameter set was found that yielded an acceptable fit to the data within the experimental noise.

The resulting two collections of parameter sets are not statistically weighted ensembles like that we built to make predictions from the Brown et al. model. Nevertheless, Brodersen et al.'s collections of acceptable parameter sets likely approximate such statistical ensembles. Using principal component analysis to fit a multidimensional gaussian to each of Brodersen's parameter collections yields a Hessian matrix we can test for sloppiness.

Figure 2.26 shows eigenvalue spectra derived from Brodersen's acceptable parameter sets for the Albumin (Alb ens) and Hemoglobin (Heme ens) models. For comparison, the eigenvalues of the χ^2 Hessians for the two models are also shown (Alb chi^2 and Heme chi^2).

The sloppiness of both models is evident, using both our χ^2 measure of system behavior and with Brodersen's parameter set collection. In each case the eigenvalues span several decades roughly evenly.

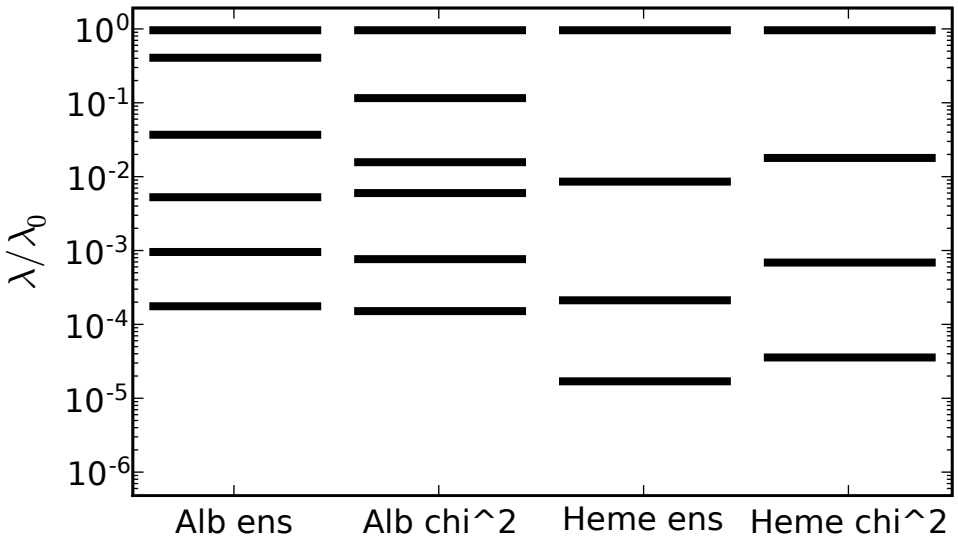


Figure 2.26: Sloppy eigenvalues of Brodersen et al.'s models.

CHAPTER 3

COMPUTATIONAL KINETIC MODELING OF EXPERIMENTS ON THE ACTIVATION OF A TRANSDUCIN MUTANT

3.1 Introduction

G protein-coupled receptors (GPCRs) are large membrane-bound proteins that are found in many higher eukaryotes. They are involved in a very diverse range of biological processes, from vision to smell to the immune response; the human genome is estimated to estimated to encode approximately 950 GPCRs [59]. Given the many processes they help regulate, it is little surprise that GPCRs are medically important. In fact, over 30% of all drugs target GPCRs [60].

Active GPCRs signal across the membrane by interacting with heterotrimeric (or 'large') G proteins, which consist of two subunits, G_α and $G_{\beta\gamma}$. The G_α subunit is structurally related to the 'small' G proteins of the Ras superfamily and binds guanine nucleotides. Typically the subunit is inactive in signaling when bound to guanine diphosphate (GDP) and active when bound to guanine triphosphate (GTP). The $G_{\beta\gamma}$ subunit binds G_α^{GDP}, and most researchers suspect that the $G_\alpha G_{\beta\gamma}$ complex dissociates upon receptor-driven exchange of GDP for GTP on G_α.

The prototypical heterotrimeric G protein signaling cycle begins with an inactive receptor R and the complex $G_\alpha^{GDP} G_{\beta\gamma}$. Upon stimulation, the receptor adopts an activated conformation R^*. The activated receptor binds $G_\alpha^{GDP} G_{\beta\gamma}$ and drives the release of GDP and subsequent binding of GTP. (In cells GTP is typically at much higher concentration than GDP.) Upon GTP binding, the complex dissociates into R^*, G_α^{GTP}, and $G_{\beta\gamma}$. Both the G_α^{GTP} and $G_{\beta\gamma}$ can signal down-stream effectors. The G_α^{GTP} subunit ceases to signal when its intrinsic GTP hydrolysis activity converts the bound GTP into GDP, and the $G_{\beta\gamma}$ subunit can be silenced

by binding to G_α^{GDP}. Finally, the cycle is catalytic, in that an activated receptor can typically activate many G proteins before returning to the inactive state.

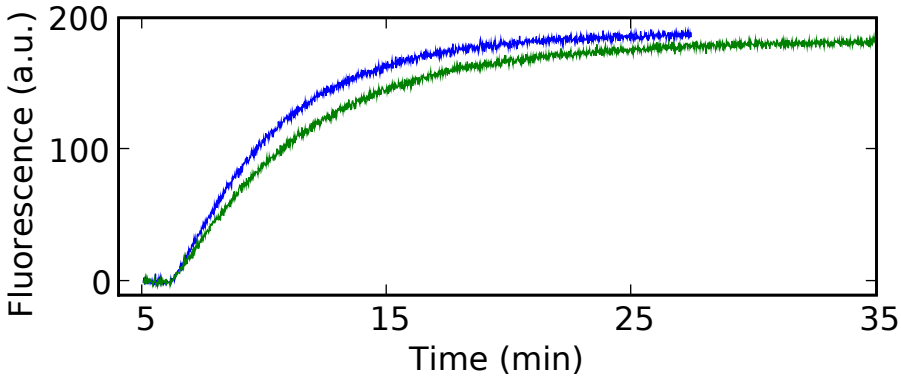
The above activation cycle is well-accepted in outline, but many details remain controversial. For example, several receptors may need to bind to promote nucleotide exchange [61], or the $G_{\beta\gamma}$ subunit may not dissociate from the G_α subunit upon GTP binding [62]. Current methods to study the activation of heterotrimeric G proteins include docking of component crystal structures [63], molecular dynamics [64], various labeling techniques [65], and biochemical studies, including those on mutants [66, 67, 68]. Here we study the activation cycle by coupling in vitro biochemical studies and mutagenesis with detailed computational kinetic modeling.

We work with the vision pathway, where the receptor is rhodopsin and the G protein is transducin [69, 70]; this is perhaps the best studied heterotrimeric G protein system. In particular, we are interested in elucidating the role played by a threonine residue found in the guanine nucleotide binding pocket of G_α [71]. This threonine (T177 in transducin) hydrogen bonds to the terminal phosphate of GTP and also coordinates a Mg^{2+} ion; it is one of the few residues conserved between both the large and small G proteins.

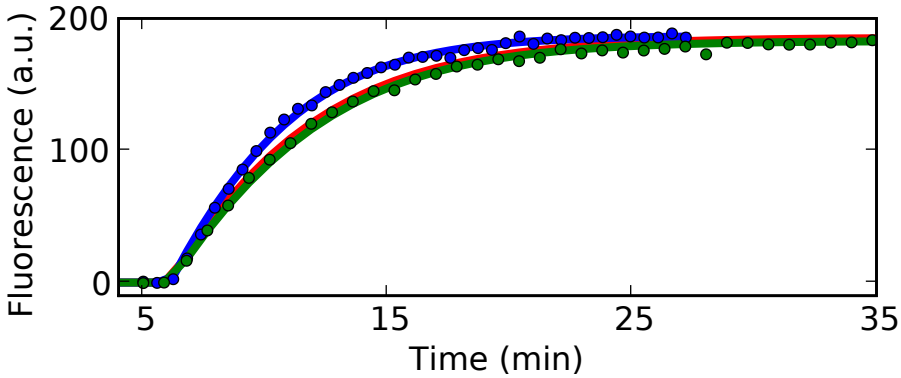
3.2 Data

All the data we work with were collected by Sekar Ramachandran in the lab of Rick Cerione.

Mammalian transducin does not express well in E. coli, so the experiments were done with a chimeric version which does express well and is known to act similarly to the pure mammalian version [72]. Herein we refer to this chimera as the 'wild-type'. The threonine at position 177 was mutated to alanine in the background of the chimera, and this T177A mutant was also expressed in E. coli.



(a) Example raw fluorescence data



(b) Example data fits

Figure 3.1: (a) Shown is the change in fluorescence of a preparation of 292 nM wild-type G_α, 16 nM $G_{\beta\gamma}$, and 2.3 nM R^* upon addition of 1 μM of GTPγS. Note that the difference between the traces is much larger than the fluctuations within each trace. This motivated us to fit a model including systematic errors in the initial conditions. These traces were taken on the same day; traces taken on different days differ more. (b) Before fitting, the data is down-sampled for computational convenience. The dots are the down-sampled data points from the traces in (a); the uncertainties assigned to each data point are approximately the size of each of the dots. The red curve (almost completely hidden by the green) is the optimal fit with initial conditions fixed at their reported values. The blue and green curves are the optimal fits allowing initial conditions to vary.

$G_{\beta\gamma}$ and rhodopsin were purified from bovine retina.

G_α contains several intrinsically fluorescent tryptophan residues, and the fluorescence of the protein changes upon GDP-GTP exchange [73, 74]. This allows G_α activation to be monitored with very high temporal resolution. Ramachandran monitored the fluorescence change of preparations of G_α^{GDP}, $G_{\beta\gamma}$, and solubilized R^* upon addition of the non-hydrolyzable GTP analog GTPγS for both the wild-type and mutant over a large range of initial conditions. In most cases, G_α^{GDP} was at higher concentration than $G_{\beta\gamma}$ or R^*, so the cycle had to be catalytic, with each $G_{\beta\gamma}$ and R^* activating several G_α^{GDP}. The wild-type data set consists of 26 traces taken on 7 different days, for a total of 25,376 data points. The T177A data set consists of 73 traces taken on 12 different days, for a total 155,862 data points. A representative pair of traces is shown in Figure 3.1(a). Details on the processing of the data and connecting it with the computational model are discussed in Section 3.4.

3.3 Computational Model

The model we fit to the data is illustrated in Figure 3.2. Essentially it encapsulates the traditional view of the heterotrimeric G protein cycle. First consider G_α^{GDP}, at the lower left. This can irreversibly lose its bound GDP to become $G_\alpha^{()}$. $G_\alpha^{()}$ degrades spontaneously, but it can also irreversibly bind GTP to become G_α^{GTP}. Going clockwise around the circle, G_α^{GDP} can reversibly bind $G_{\beta\gamma}$ to form $G_\alpha^{GDP}G_{\beta\gamma}$, which can reversibly bind R^*. The complex of $G_\alpha^{GDP}G_{\beta\gamma}$ and R^*, denoted RT, undergoes irreversible GDP-GTP exchange to form $G_\alpha^{GTP}G_{\beta\gamma}$ and free R. Finally $G_\alpha^{GTP}G_{\beta\gamma}$ can reversibly dissociate to from G_α^{GTP} and free $G_{\beta\gamma}$. $G_\alpha^{GTP}G_{\beta\gamma}$ and G_α^{GTP} are assumed to contribute equally to the fluorescence signal.

The reactions are all modeled with mass-action kinetics, and the equations

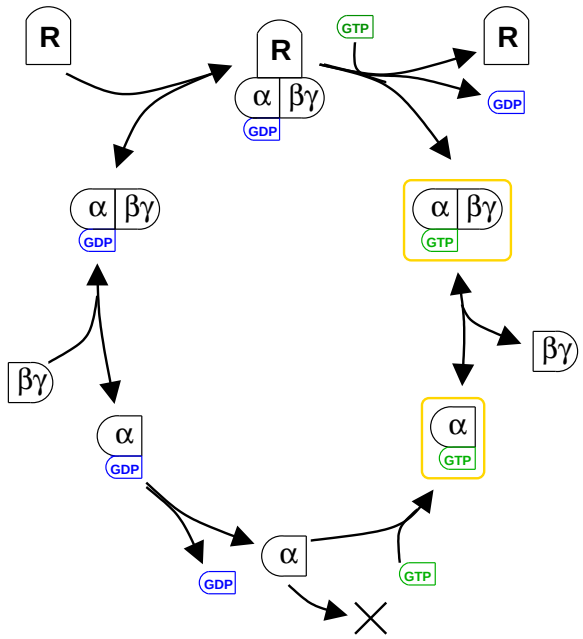


Figure 3.2: Our computational model for the activation of heterotrimeric G proteins is illustrated here. For a detailed discussion of the model, see Section 3.3. The species boxed in yellow are assumed to contribute to the fluorescence signal.

defining the model are reproduced in Appendix 3.A.

3.4 Fitting the Model

We fit the model via least-squares, minimizing the cost function:

$$C(\theta) = \frac{1}{2} \sum_d \sum_i \left(\frac{B_d \, y_i(\theta) - d_i}{\sigma_i} \right)^2 + \text{priors}. \tag{3.1}$$

The B_d are *scale factors* [7] which account for the unknown conversion factor between measured fluorescence and concentration of G_α^{GTP}. The subscript d indicates that each scale factor is shared among runs performed on a given day; we partition the scale factors by day to account for possible instability of the fluorimeter. Data point d_i is compared with the corresponding model result $y_i(\theta)$ where θ is the current set of parameters. σ_i is the Gaussian uncertainty assumed for each data point and is estimated as the standard deviation of the first minute of each trace,

before GTPγS addition. The 'priors' terms reflect additional constrains that will be discussed below. The model is fit separately for the wild-type and mutant data sets, and several additional data manipulations are necessary beforehand.

As seen in Figure 3.1(a), each trace contains many data points. Clearly, however, the trace can be well-described by many fewer points. Thus we down-sample using the autocorrelation of the traces. The autocorrelation time is estimated for each raw data trace as the lag at which the autocorrelation function drops below 0.5. (Note that this is *not* the noise autocorrelation time, which is very small.) Each trace is then down-sampled by taking points separated by one-half the above estimated correlation time. The points in Figure 3.1(b) are the down-sampled values from the traces in Figure 3.1(a); the error bars on each point are approximately the size of the points themselves. This very conservative down-sampling reduces the number of data points considerably, to 1,055 points for the wild-type data and 2,656 points for the T177A data. This reduction greatly speeds up calculation of the cost $C(\theta)$ for any given set of parameters.

Figure 3.1(a) shows that the variation in traces between replicate experiments is much larger than the variation within any one trace. Note that the two traces shown were taken on the same day, and variation between traces taken on different days are larger. This variation could be accounted for by widening the uncertainties assumed for each trace. However, given the relatively low number of replicate experiments, it is unclear how much additional uncertainty to assume. The experiments are performed in vitro, so we don't expect the intrinsic variability that can plague cell-based assays. Different batches of protein may, however, have degraded to different degrees, and the amount of protein added in each experiment may be imprecise. Thus we include additional parameters η that allow the initial conditions to vary from the reported values. Their inclusion may, however, over-fit

Table 3.1: Tabulated are the best-fit data costs for our models of wild-type and T177A activation, with initial conditions fixed at their reported values, optimized without constraints, and optimized with constraints. 'Prior cost' indicates the portion of the total cost due to the 'prior' residuals. The total optimized cost is the sum of the 'Data' and 'Prior' costs.

	wild-type			T177A		
	fixed	free	constrained	fixed	free	constrained
Data cost	23,699	1,350	2,612	199,443	2678	20,339
Data cost per pt	21.0	1.2	2.3	69.5	1.0	7.7
Prior weight w			10,000			15,625
Prior cost			2,559			22,796

the data and lower the power of our analysis, as it adds dozens of parameters to the wild-type fit and over two hundred to the T177A fit. To better understand the effect of adjusting initial conditions, we perform several optimizations and compare results. First the models are optimized with the initial conditions for each experiment fixed at their reported values. They are then optimized with initial conditions allowed to fluctuate with no constraint. This yields unrealistically large adjustments, so additional optimizations are performed with prior terms of the form

$$\frac{w}{2}\left(\log \eta_i - \log 1.0\right)^2 \tag{3.2}$$

in the cost. The weights w for these constraints on the ηs are separately adjusted for the wild-type and T177A to minimize these initial-condition fluctuations while maintaining a fit that remains reasonable to the eye.

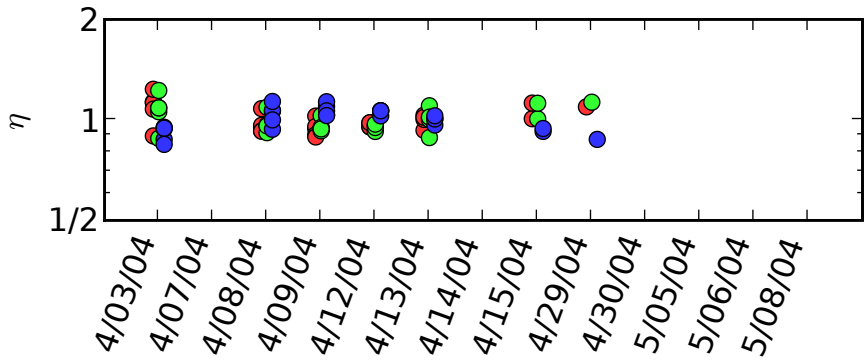
The optimizations are performed with SloppyCell (Chapter 4), using the Nelder-Mead simplex and conjugate-gradient algorithms from the SciPy library of scientific routines [75, 76] and SloppyCell's Levenberg-Marquardt routine. The final optimized costs for the fits with initial conditions fixed at their reported values, optimized without constraint, and optimized with prior constraint are shown

in Table 3.1. Figure 3.1(b) illustrates the improvement in fit when initial conditions are freely optimized. The final costs per data point in the fits with freely optimized initial conditions are approximately one. A perfect fit would yield a cost per data point of one-half, so although our fit is decent, we are not obviously over-fitting. With the assigned weights, w, the cost due to the prior constraints approximately equals the cost to fit the data in both constrained fits. Note, however, that the T177A constrained fit has a much higher data cost per residual than the wild-type fit, indicating a much poorer fit.

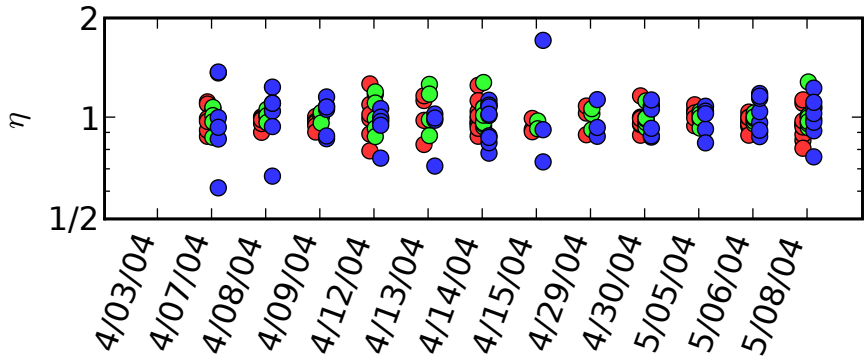
Figure 3.3 shows the initial condition perturbations η that optimally fit the wild-type and T177A data with the applied prior constraints. Note that the adjustments are substantially larger for fitting the T177A data. The standard deviation of η is 0.08 for the wild-type fit and 0.12 for the T177A fit, corresponding to 8% and 12% variations in the initial conditions. It is estimated that the uncertainty in each dispensing due to pipette accuracy is about 5% (S. Ramachandran, personal communication). Our initial condition adjustments are of the same order, but slightly larger. The only obvious trend is that G_α^{GDP} for T177A seems to require the most substantial adjustments. Perhaps this is because the protein has been destabilized by the mutation.

3.5 Parameter Uncertainties

Basing one's inferences solely on a single best-fit set of parameters is dangerous, because many other sets may fit the data almost as well (Chapter 2). Here we consider two ways to estimate the uncertainty of our parameters: the Bayesian ensemble approach and covariance analysis.



(a) Wild-type



(b) T177A

Figure 3.3: The optimal initial condition adjustments η are plotted on a logarithmic scale when fitting (a) the wild-type and (b) the T177A data. Points are colored by the initial condition they effect: (red) rhodopsin, (green) $G_{\beta\gamma}$, (blue) G_{α}^{GDP}. (A value of $\eta = 1/2$ corresponds to the data being best-fit by an initial condition of 1/2 the reported value.)

3.5.1 Bayesian Ensembles

We used SloppyCell to build a ensemble of parameter sets consistent with the fluorescence data we are fitting [7]. In the ensemble, parameter sets are statistically sampled via a Markov-Chain Monte Carlo random walk with equilibrium probability density

$$P(\theta) \propto e^{-G(\theta,T)/T}. \tag{3.3}$$

Here $G(\theta, T)$ is the *free energy* at parameters θ and temperature T. This is the cost plus an additional term accounting for fluctuations in the scale factors[1].

Note that generating well-converged ensembles with all initial conditions allowed to fluctuate would take an enormous amount of computer time. To have a reasonable acceptance ratio, SloppyCell chooses each attempted step in the ensemble so that it changes the cost by approximately one unit of temperature. The overall expected fluctuations in the cost are one-half unit of temperature per parameter. Also, a random walk on average travels a distance proportional to the square root of the number of steps taken. Together these facts suggest that the number of steps required to converge (governed by the number to achieve the required cost fluctuations) should scale with the *square* of the number of parameters. Allowing all initial conditions to fluctuate multiplies the number of parameters in the wild-type fit by a factor of nine, suggesting that such an ensemble would take approximately eighty times as many steps to converge as one with the initial conditions fixed. For the T177A fit, it is much worse. There are 214 initial conditions, so that the fluctuating initial conditions ensemble requires approximately *five hundred* times as many steps to converge as one with fixed initial conditions. Rather than generating these ensembles, we generate ensembles with initial conditions fixed at

[1] During the ensemble build, the scale factors are assigned a Gaussian prior on their logarithm, centered at their best-fit mean with a standard deviation of 50%. (See Section 6.2 for a discussion of the subtleties of scale factor priors.)

Table 3.2: Shown are the ensemble temperatures using Frederiksen et al.'s suggestion [10] for the various ensembles. 'reported i.c.' and 'optimal i.c.' refer to the ensembles with initial conditions fixed at their reported and optimized values, respectively. 'fluctuating i.c.' is the ensemble (not actually built here) where initial conditions are allowed to fluctuate.

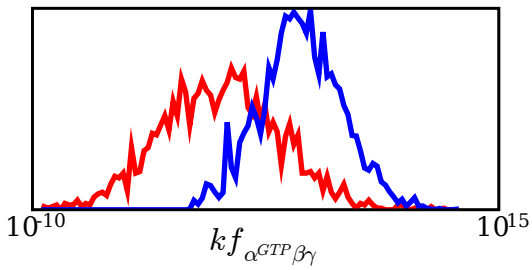
	wild-type	T177A
reported i.c.	4740	39888
optimal i.c.	1034	8473
fluctuating i.c.	118	385

their reported and optimized values and use covariance analysis to estimate the impact of initial condition fluctuations.

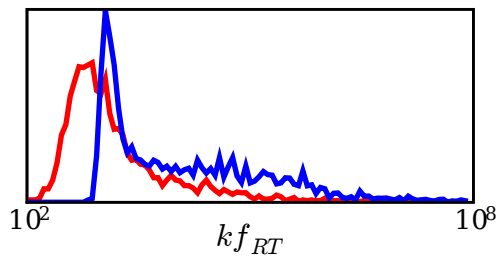
If all our data points were statistically independent and fit well by the model, then a temperature of one would be statistically appropriate. However, our data points are highly correlated even after down-sampling, and when initial conditions are fixed at their reported values, the fits are poor. Following Frederiksen et al. [10], we thus set the temperature T to twice the best-fit cost divided by the number of parameters. At this temperature it is expected that typical members of the ensemble will have approximately twice the cost of the best fit. Table 3.2 shows the resulting temperatures for the various possible ensembles.

To speed convergence and to avoid taxing the integrator unnecessarily, when building the ensemble we place weak priors on the logarithms of the parameter values, to prevent them from wandering to zero or infinity. These priors restrict the parameters to be within a factor of 10^6 larger or smaller than their best-fit values.

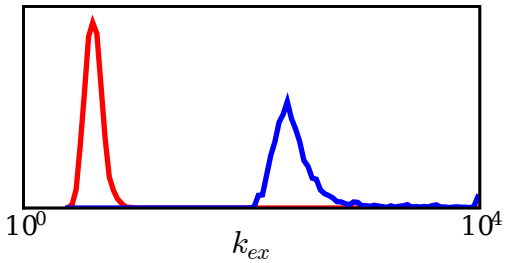
Figure 3.4 shows the ensemble distributions of three parameters that illustrate the different possible cases of constraint. All come from the ensembles with initial conditions fixed at their optimized values. Note that $kf_{\alpha^{GTP}\beta\gamma}$ is only constrained by the artificial priors we added to protect the integrator, while kf_{RT} appears to



(a) Very unconstrained



(b) Constrained only from below



(c) Usefully constrained

Figure 3.4: Shown are the distributions for three particular parameters in the ensemble built with initial conditions fixed at their optimized values. Blue curves correspond to the wild-type fit and red to the T177A fit. The only constraint on $kf_{\alpha^{GTP}\beta\gamma}$ is the prior we placed to prevent integration errors, while kf_{RT} is constrained from below, but not from above. k_{ex}, on the other hand, is tightly constrained.

Table 3.3: Tabulated are the confidence intervals for k_{ex}, the rate of rhodopsin-mediated GDP-GTP exchange, from our ensembles and from the covariance analysis. (The units of k_{ex} are $\mu\text{M}^{-1}\text{min}^{-1}$.)

	reported ensemble	optimized ensemble	fluctuating $J^{\mathsf{T}}J$
wild-type	(43, ∞)	(74, 3032)	(52, 269)
T177A	(0.6, 6.0)	(1.1, 2.4)	(0.7, 4.0)

additionally be constrained from below. (This value at which it is constrained, however, depends sensitively on how tightly we constrain the initial condition adjustments in the optimization.) k_{ex}, on the other hand, is well-constrained, and the difference between wild-type and T177A is obvious. In fact, k_{ex} is the only parameter which the ensembles consistently constrain well; Table 3.3 shows the resulting confidence intervals for k_{ex}.

3.5.2 Covariance Analysis

Covariance analysis involves a quadratic approximation to the shape of the cost surface around the best fit set of parameters. Here we use the $J^{\mathsf{T}}J$ matrix to describe the cost surface:

$$J^{\mathsf{T}}J \equiv \sum_k \frac{dr_k}{d\log\theta_i}\frac{dr_k}{d\log\theta_j}. \tag{3.4}$$

The sum is over residuals r_k where each residual corresponds to a single data point: $r_k \equiv (B_d y_k(\theta) - d_k)/\sigma_k$. The $J^{\mathsf{T}}J$ approximation to the Hessian is accurate when the model fits the data well, and it is quite useful because SloppyCell can calculate first derivatives without resorting to finite-differences. In the covariance analysis approximation, the uncertainty in the logarithm of parameter i is then equal to

$$\sigma_{\theta_i} = \sqrt{T\left((J^{\mathsf{T}}J)^{-1}\right)_{i,i}}. \tag{3.5}$$

This analysis ignores nonlinearities that are captured by the ensemble approach, but it is much less computationally expensive, so we use it to estimate the effect of initial-condition parameter fluctuations on our results.

The final column of Table 3.3 shows the confidence intervals for k_{ex} obtained from this analysis. They are quite similar to those obtained from the ensemble with fixed initial conditions, again supporting the conclusion that k_{ex} is much smaller for the T177A mutant than the wild-type.

Appendix 3.B discusses the 'sloppiness' of the models as quantified by $J^{\mathsf{T}}J$.

3.6 Discussion

As seen in Table 3.3, our detailed analysis has placed on a firm statistical footing the observation that rhodopsin-mediated nucleotide exchange on the T177A mutant is much slower than on the wild-type. It was expected that the mutation would affect this rate of exchange, and here we have quantified that change.

This project offers several lessons about the modeling process. In particular, note that our uncertainties of k_{ex} are smaller after fitting initial conditions, even though that introduced many parameters (Table 3.3). This is directly related to our use of Frederiksen et al.'s temperature prescription [10]; the better fit (and resulting lower ensemble temperature) compensates for the extra degrees of freedom added by fitting the initial conditions.

The constraints placed on the initial conditions during optimization are here set quite arbitrarily. A better understanding of the actual initial condition uncertainties is vital to make further progress. This is particularly true because the optimized initial conditions may additionally be compensating for the effects of biochemical steps that we have not included in the model. To estimate the actual initial condition uncertainties, we could replicate the dispensing process several

times, but rather than carrying on with the experiment, carefully assay how much protein ended up in the test tube. We could then adjust the prior weights w to achieve the expected scatter in ηs.

In these fits we have used only one form of data. Other experiments have been performed on the T177A mutant, and it may be very instructive to include them in our analysis. Incorporating additional experiments will, however, require care. Because the fluorescence data contains so many data points, it must be re-weighted to avoid swamping experiments that are just as informative yet contain many fewer data points. The re-weighting may, however, unavoidably be subjective, based on how much information we think each experiment actually contains.

Given better control of the initial conditions or more data, it would perhaps be interesting to consider alternative models for the activation process. Our parameter estimates are contingent both on the data we fit and the model we use, and it is possible that there are signatures of more complicated mechanisms (e.g. receptor oligomerization) in the data that are being obscured by our choice of model or our use of all the initial conditions as fitting parameters. One possibly important simplification made in the current model is that binding of $\mathrm{G}_{\beta\gamma}$ protects $\mathrm{G}_{\alpha}^{\mathrm{GDP}}$ from spontaneous loss of GDP; there is experimental evidence that rates of spontaneous exchange are similar for $\mathrm{G}_{\alpha}^{\mathrm{GDP}}$ and $\mathrm{G}_{\alpha}^{\mathrm{GDP}}\mathrm{G}_{\beta\gamma}$ (S. Ramachandran, personal communication), and including this may improve the model fit with less need for initial condition adjustments.

Finally, we could consider a coupled fit to both data sets, with all parameters required to be equal between wild-type and T177A (except for k_{ex}, which was anticipated to change due to the mutation). The equal parameters model is nested within the model with independent parameters, so a likelihood-ratio test or Bayes factor [77] can be used to assess whether the extra parameters improve the fit in

a statistically significant sense. This would provide a rigorous way of assessing whether any parameters other than k_{ex} must differ between the mutant and the wild-type to explain the data.

Our work here has helped quantify the effect of the T177A mutation on transducin activation, showing that the rate of rhodopsin-mediated GDP-GTP exchange is much slower in the mutant. Our work also illustrates how important a thorough understanding of the experimental uncertainties will be to continued modeling of such experiments.

3.A Model Equations

$$
\begin{aligned}
\frac{d\,[\mathrm{R}]}{dt} &= \mathrm{kd}_{\mathrm{RT}} \cdot [\mathrm{RT}] \\
&\quad + \mathrm{k}_{\mathrm{ex}} \cdot [\mathrm{RT}] \cdot [\mathrm{GTP}] \\
&\quad - \mathrm{kf}_{\mathrm{RT}} \cdot [\mathrm{R}] \cdot \left[\alpha^{\mathrm{GDP}}\beta\gamma\right]
\end{aligned}
$$

$$
\begin{aligned}
\frac{d\left[\alpha^{\mathrm{GDP}}\right]}{dt} &= \mathrm{kd}_{\alpha^{\mathrm{GDP}}\beta\gamma} \cdot \left[\alpha^{\mathrm{GDP}}\beta\gamma\right] \\
&\quad - \mathrm{kd}_{\alpha^{\mathrm{GDP}}} \cdot \left[\alpha^{\mathrm{GDP}}\right] \\
&\quad - \mathrm{kf}_{\alpha^{\mathrm{GDP}}\beta\gamma} \cdot \left[\alpha^{\mathrm{GDP}}\right] \cdot [\beta\gamma]
\end{aligned}
$$

$$
\begin{aligned}
\frac{d\,[\beta\gamma]}{dt} &= \mathrm{kd}_{\alpha^{\mathrm{GDP}}\beta\gamma} \cdot \left[\alpha^{\mathrm{GDP}}\beta\gamma\right] \\
&\quad + \mathrm{kd}_{\alpha^{\mathrm{GTP}}\beta\gamma} \cdot \left[\alpha^{\mathrm{GTP}}\beta\gamma\right] \\
&\quad - \mathrm{kf}_{\alpha^{\mathrm{GDP}}\beta\gamma} \cdot \left[\alpha^{\mathrm{GDP}}\right] \cdot [\beta\gamma] \\
&\quad - \mathrm{kf}_{\alpha^{\mathrm{GTP}}\beta\gamma} \cdot \left[\alpha^{\mathrm{GTP}}\right] \cdot [\beta\gamma]
\end{aligned}
$$

$$
\begin{aligned}
\frac{d\,[\mathrm{GTP}]}{dt} &= -\mathrm{kf}_{\alpha^{\mathrm{GTP}}} \cdot \left[\alpha^{()}\right] \cdot [\mathrm{GTP}] \\
&\quad - \mathrm{k}_{\mathrm{ex}} \cdot [\mathrm{RT}] \cdot [\mathrm{GTP}]
\end{aligned}
$$

$$
\begin{aligned}
\frac{d\left[\alpha^{()}\right]}{dt} &= \mathrm{kd}_{\alpha^{\mathrm{GDP}}} \cdot \left[\alpha^{\mathrm{GDP}}\right] \\
&\quad - \mathrm{kf}_{\alpha^{\mathrm{GTP}}} \cdot \left[\alpha^{()}\right] \cdot [\mathrm{GTP}] \\
&\quad - \mathrm{kd}_{\alpha^{()}} \cdot \left[\alpha^{()}\right]
\end{aligned}
$$

$$
\begin{aligned}
\frac{d\,[\mathrm{GDP}]}{dt} &= \mathrm{kd}_{\alpha^{\mathrm{GDP}}} \cdot \left[\alpha^{\mathrm{GDP}}\right] \\
&\quad + \mathrm{k}_{\mathrm{ex}} \cdot [\mathrm{RT}] \cdot [\mathrm{GTP}]
\end{aligned}
$$

$$
\begin{aligned}
\frac{d\left[\alpha^{\mathrm{GTP}}\right]}{dt} &= \mathrm{kf}_{\alpha^{\mathrm{GTP}}} \cdot \left[\alpha^{()}\right] \cdot [\mathrm{GTP}] \\
&\quad + \mathrm{kd}_{\alpha^{\mathrm{GTP}}\beta\gamma} \cdot \left[\alpha^{\mathrm{GTP}}\beta\gamma\right] \\
&\quad - \mathrm{kf}_{\alpha^{\mathrm{GTP}}\beta\gamma} \cdot \left[\alpha^{\mathrm{GTP}}\right] \cdot [\beta\gamma] \\
\\
\frac{d\left[\alpha^{\mathrm{GDP}}\beta\gamma\right]}{dt} &= \mathrm{kf}_{\alpha^{\mathrm{GDP}}\beta\gamma} \cdot \left[\alpha^{\mathrm{GDP}}\right] \cdot [\beta\gamma] \\
&\quad + \mathrm{kd}_{\mathrm{RT}} \cdot [\mathrm{RT}] \\
&\quad - \mathrm{kd}_{\alpha^{\mathrm{GDP}}\beta\gamma} \cdot \left[\alpha^{\mathrm{GDP}}\beta\gamma\right] \\
&\quad - \mathrm{kf}_{\mathrm{RT}} \cdot [\mathrm{R}] \cdot \left[\alpha^{\mathrm{GDP}}\beta\gamma\right] \\
\\
\frac{d\left[\alpha^{\mathrm{GTP}}\beta\gamma\right]}{dt} &= \mathrm{kf}_{\alpha^{\mathrm{GTP}}\beta\gamma} \cdot \left[\alpha^{\mathrm{GTP}}\right] \cdot [\beta\gamma] \\
&\quad + \mathrm{k}_{\mathrm{ex}} \cdot [\mathrm{RT}] \cdot [\mathrm{GTP}] \\
&\quad - \mathrm{kd}_{\alpha^{\mathrm{GTP}}\beta\gamma} \cdot \left[\alpha^{\mathrm{GTP}}\beta\gamma\right] \\
\\
\frac{d\,[\mathrm{RT}]}{dt} &= \mathrm{kf}_{\mathrm{RT}} \cdot [\mathrm{R}] \cdot \left[\alpha^{\mathrm{GDP}}\beta\gamma\right] \\
&\quad - \mathrm{kd}_{\mathrm{RT}} \cdot [\mathrm{RT}] \\
&\quad - \mathrm{k}_{\mathrm{ex}} \cdot [\mathrm{RT}] \cdot [\mathrm{GTP}]
\end{aligned}
$$

3.B Sloppiness of the Models

Our group has argued that least-squares fits to data universally have a 'sloppy' pattern of eigenvalues in their Hessian and $J^{\mathsf{T}}J$ matrices, with eigenvalues roughly evenly spaced over many decades [7, 11] (Chapter 2). The eigenvalue spectra for both the wild-type and T177A data fits are shown in Figure 3.5. To test the sloppiness of the models, here we calculate $J^{\mathsf{T}}J$ using the parameters optimized with no constraints on the ηs.

The eigenvalue spectra for both the wild-type and T177A fits with fixed initial conditions are characteristically sloppy. The stiffest two eigenvectors (those corresponding to the largest eigenvalues) for each of these fits are shown in Figure 3.6. The stiff eigenvectors indicate combinations of parameters that are particularly well-constrained by the data. In the wild-type fit the equilibrium constant for formation of the $\mathrm{G}_{\alpha}^{\mathrm{GDP}}\mathrm{G}_{\beta\gamma}$ dimer ($kd_{\alpha^{GDP}\beta\gamma}/kd_{\alpha^{GDP}\beta\gamma}$) is a substantial component

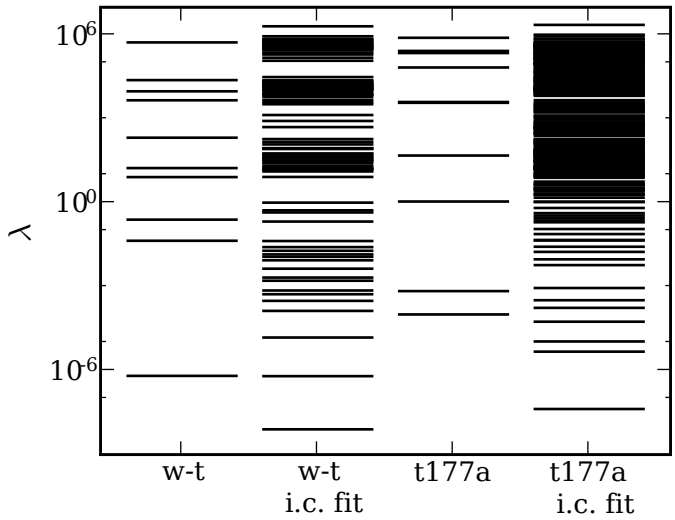
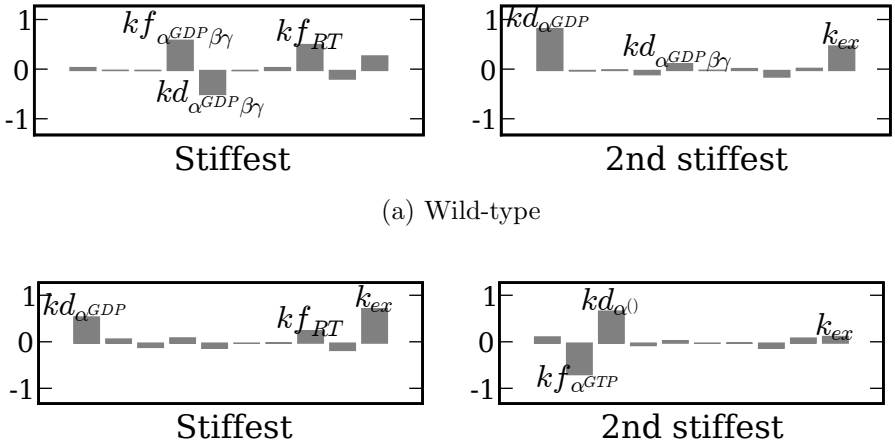


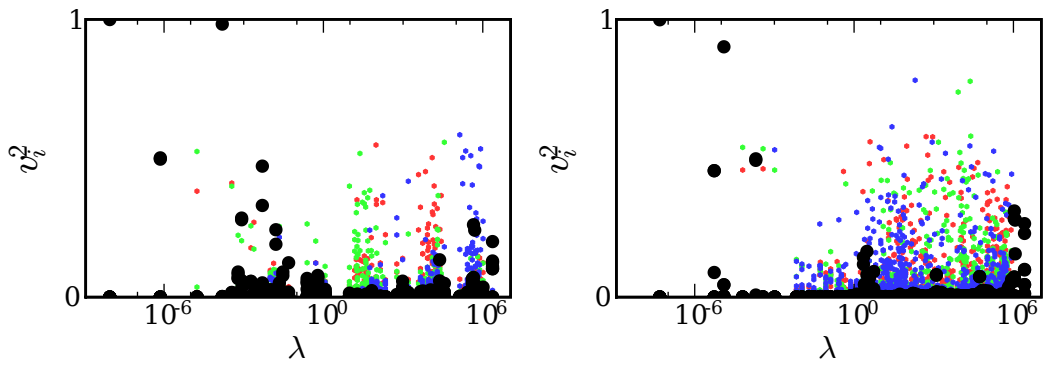
Figure 3.5: Shown are the eigenvalue spectra for the models fit to both the wild-type and T177A data sets, with and without initial conditions allowed to vary. Both fits with fixed initial conditions show characteristically sloppy spectra [7, 11] (Chapter 2). The fits with the initial conditions allowed to vary, on the other hand, have a substantial dearth of small eigenvalues.



(a) Wild-type

(b) T177A

Figure 3.6: Shown are the two stiffest eigenvectors (corresponding to the largest two eigenvalues) for both data fits with fixed initial conditions. These vectors correspond to particularly well-constrained combinations of parameters. Interestingly, the second stiffest wild-type vector is very similar to the stiffest T177A vector.



(a) Wild-type (b) T177A

Figure 3.7: Plotted are the squared magnitudes of each component of each eigenvector versus the corresponding eigenvalue. Components corresponding to the 10 core model parameters are shown by black dots, while red dots are η_R, green are $\eta_{\beta\gamma}$ and blue are η_α.

of the stiffest eigenvector, while the second stiffest vector involves the tradeoff between spontaneous loss of GDP from G_α^{GDP} ($kd_{\alpha^{GDP}}$) and GDP-GTP exchange driven by the receptor (k_{ex}). Similarly, these two parameters comprise the stiffest eigenvector in the T177A fit, while the second stiffest vector for that fit constrains the spontaneous capture of GTP by empty G_α subunits.

The fits with varying initial conditions deviate from the classical sloppy eigenvalue spectrum, with a much higher density of eigenvalues near the large end of the spectrum. Figure 3.7 shows that the eigenvectors corresponding to the glut of eigenvalues are mostly comprised of the initial condition adjustment parameters η. The vectors with large components along the parameters intrinsic to the model (indicated by the black dots) are spread out amongst eigenvalues with a characteristic sloppy spacing, particularly in the wild-type fit. Interestingly, the stiffest eigenvectors of the wild-type fit tend to have large components along the ηs for adjusting the G_α initial condition (blue dots); this is not true for the T177A fit.

CHAPTER 4

FALSIFIABLE MODELING OF BIOCHEMICAL NETWORKS WITH SLOPPYCELL*

4.1 Abstract

Summary: To falsify model assumptions, predictions must statistically incorporate the effects all parameter sets consistent with the model and available data. SloppyCell is an SBML-compatible modeling environment that implements two algorithms for generating such falsifiable predictions.

Availability: Open Source (BSD) at http://sloppycell.sourceforge.net

Contact: rng7@cornell.edu

4.2 Introduction

Models of biochemical networks often involve dozens or hundreds of parameters, and often very few of those parameters will have been directly measured. Thus the parameters must be estimated, typically by fitting the model to time-series or other system-level data via nonlinear optimization [29]. Numerous packages exist to facilitate such parameter estimation. Among those that import models encoded in the Systems Biology Markup Language (SBML) [55] are Systems Biology Toolbox [78], Systems Biology Workbench [79], COPASI [80], and the SBML Parameter Estimation Toolkit (SBML-PET) [81]. Parameter estimation is complicated by practical parameter unidentifiability; there is often a large, highly-correlated, collection of parameter sets that fit the data well [9] (Chapter 2).

*In preparation for submission as an Applications Note to *Bioninformatics* with authors Ryan N. Gutenkunst, Jordan C. Atlas, Robert S. Kuczenski, Fergal P. Casey, Joshua J. Waterfall, Kevin S. Brown, Christopher R. Myers and James P. Sethna.

In most cases, however, the ultimate goal of modeling a system is not to deduce parameter values, but to test a particular set of assumptions about how the system works and to generate useful predictions. Given that parameter unidentifiability is common in fits of biochemical networks to data (Chapter 2), it is important to not just make a prediction from the best-fit set of parameters, but to estimate that prediction's uncertainty by accounting for other sets of parameters that are statistically consistent with the available data [7, 8]. Such uncertainty estimates help distinguish between predictions that can and cannot be trusted. Moreover, if a new experiment lies outside the uncertainty bounds of the prediction, it is strong evidence that some assumption in the model structure is wrong, rather than just a parameter estimate.

SloppyCell is an SBML-compatible modeling environment focused on studying the ensemble of model parameter sets consistent with a set of data and thus making falsifiable predictions.

4.3 Methods

Fitting a model to data typically typically involves minimizing some cost function $C(\theta|D, M)$ which measures how much the predictions of model M differ from from the data D at parameters θ; the best-fit set of parameters θ^* minimizes the cost. Assuming Gaussian uncertainties σ_d on each of the data points d, this corresponds to a weighted least-squares fit:

$$C(\theta|D, M) = \sum_{d \in D} \frac{(y_d(\theta) - d)^2}{2\sigma_d^2}, \tag{4.1}$$

where $y_d(\theta)$ is the model result for d at parameters θ. The uncertainties in a model prediction depend on how quickly the cost increases away from the best-fit set of parameters and on the sensitivity of that prediction to parameter changes.

SloppyCell implements two methods for estimating model prediction uncertainties, Linearized Covariance Analysis and Bayesian Monte-Carlo [82].

Linearized Covariance Analysis approximates the cost function by a quadratic form about the best-fit and considers only the linear sensitivity of the prediction to parameter variation. The standard deviation σ_y of a quantity y is then given by:

$$\sigma_y^2 \approx \sum_{i,j} \frac{\partial y}{\partial \theta_i} \left(H^{-1}\right)_{ij} \frac{\partial y}{\partial \theta_j}\bigg|_{\theta^*}, \tag{4.2}$$

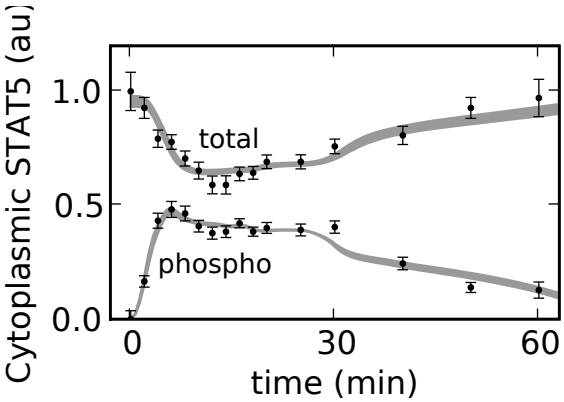
where the Hessian matrix $H_{ij} \equiv \partial^2 C(\theta|D,M)/\partial\theta_i\partial\theta_j$ is the quadratic form for the cost, $\partial y/\partial\theta_i$ is the *sensitivity* of quantity y to parameter θ_i, and all quantities are evaluated at the best-fit set of parameters θ^*.

The Bayesian Monte-Carlo approach [83, 7] explores the the full nonlinear cost surface, sampling from the probability distribution of acceptable parameter sets,

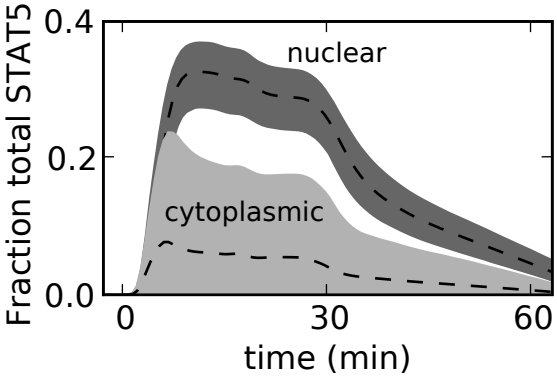
$$P(\theta|D,M) \propto e^{-C(\theta|D,M)}, \tag{4.3}$$

via Markov-Chain Monte Carlo [84]. Uncertainties in any quantity of interest can then be computed simply by evaluating that quantity over the resulting ensemble of parameter sets. This Monte-Carlo analysis is much more expensive computationally than Linearized Covariance Analysis, but it fully captures nonlinearities; such nonlinearities can be particularly important in models with relatively sparse data.

Figure 4.1 shows an example application of the Monte-Carlo method to a model of the JAK-STAT signaling pathway [85] that was also used as a test case by SBML-PET. The upper panel shows the data that were fit, which are relative measurements of total and phosphorylated STAT5 in the cytoplasm; the shaded regions are the ensemble of acceptable model fits. The lower panel shows predictions of the fraction of STAT5 in dimer form in the cytoplasm and nucleus; the



(a) Data fit to



(b) Fraction STAT in cytoplasmic dimer

Figure 4.1: JAK-STAT model [85] uncertainty analysis. (a) The fit data are relative measurements of cytoplasmic STAT5. The error bars on the data correspond to one standard deviation, and the shaded regions are the central 68% of the ensemble of statistically acceptable model fits derived from a Bayesian Monte-Carlo analysis. (b) Shown are predictions of STAT5 dimer concentrations. The dashed lines are from the best-fit set of parameters and the shaded regions are 95% uncertainty bounds.

dashed lines are from the best-fit set of parameters while the shaded regions are the 95% uncertainty bounds derived from the ensemble of parameter sets consistent with the data. Note the large uncertainty bound on dimeric cytoplasmic STAT5; given the model and data, there could be fives times as much of this species as predicted by the best fit set of parameters.

Several SBML-compatible tools exist for generating and analyzing linear parameter sensitivities $\partial y/\partial\theta$; among them are the SBML ODE Solver Library [86], the Systems Biology Toolbox [78], and BioSens [87]. To our knowledge, SloppyCell is unique in its focus on using real data to generate prediction uncertainties and in its inclusion of nonlinear Monte-Carlo-based analyses.

The STAT5 model is relatively small and well-constrained, with only 5 free parameters and data from one condition. SloppyCell and the associated methods have been used to study much larger systems, with dozens of conditions (Chapter 3) or more than fifty free parameters [12].

4.4 Features and Implementation

Most of SloppyCell is implemented in the high-level programming language Python (`http://www.python.org`) [88], and SloppyCell makes extensive use of the SciPy library of scientific routines (`http://www.scipy.org`). Like other Python-based projects [89, 90, 91] we find that the Python environment allows users great flexibility while remaining easy to use, particularly with the enhanced IPython console [92]. SloppyCell's plotting capabilities are based on matplotlib [93].

Deterministic integrations are driven by the Fortran library DASKR [94, 95]. SloppyCell interfaces with DASKR via F2PY (`http://www.scipy.org/F2py`) which is also used to automatically build C model functions so that integrations are done with full C speed. Sensitivities $\partial y/\partial\theta_i$ are calculated by integrat-

ing differential sensitivity equations derived by automatic analytic differentiation of the model equations; this avoids the imprecision of explicit finite-differences. The Bayesian Monte Carlo method can consider stochastic systems, and stochastic integrations are performed in SloppyCell using the Gillespie algorithm [96]. Cost and sensitivity calculations are parallelized using the Pypar MPI interface (http://pypar.sourceforge.net).

SloppyCell supports most of the SBML level 2, version 3 specification, including events and algebraic rules. Models can be constructed directly within SloppyCell or imported from other sources, such as the BioModels database [54] (Chapter 2) or GUI network tools like CellDesigner [97]. SloppyCell also has built-support for handling data such as Western blots or microarrays which yield only relative measurements.

4.5 Conclusions

As biochemical models progress from description to prediction, it is essential to rigorously consider the uncertainties of those predictions so that the models can be falsified. SloppyCell has proven a flexible and powerful research tool for building and exploring models of complex biochemical networks, and the SloppyCell team welcome new users, contributors and collaborators.

4.6 Acknowledgements

The authors thank Tamara Galor-Neah and Sarah Stockwell for help testing early versions of SloppyCell, and Eric Siggia for helpful suggestions. The authors acknowledge support from the National Institutes of Health, the Department of Energy, the Department of Agriculture, and the National Science Foundation.

CHAPTER 5

ADAPTIVE MUTATION IN A GEOMETRICAL MODEL OF BIOCHEMICAL EVOLUTION*

5.1 Abstract

The distribution of fitness effects of adaptive mutations remains poorly understood, both empirically and theoretically. Most recent theoretical work on the subject has focused on either the genotypic or phenotypic level; here we focus on the level of biochemical parameters (the "chemotype"). We study a version of Fisher's geometrical model formulated in terms of such parameters, wherein pleiotropy is minimal. This model generically predicts that there are striking singular cusps in the distribution of fitness effects of fixed mutations and that a single biochemical parameter should comprise all the mutations at the high end of that distribution. Using extreme value theory we show that the farthest pair of these cusps are typically well-separated, even when hundreds or thousands of biochemical parameters are relevant, implying that the effects we predict should be observable in realistically precise experiments. More broadly, our work demonstrates that new insight can be gain by viewing evolution with a biochemical perspective.

5.2 Introduction

Many aspects of the theory of evolution are well-developed [98], but our understanding of adaptive mutation remains limited [99, 100]. Recently experimental evolution has provided a new window onto adaptation [101], and there has been a resurgence of theoretical work. Most of that work has focused either at the level

*In preparation for submission to *Evolution* with authors Ryan N. Gutenkunst and James P. Sethna.

of genotype or phenotype.

The mutational landscape model focuses on the genotype, considering adaptation in the space of genetic sequences [102]. The assumption that only a small fraction of single nucleotide changes will result in genotypes fitter than the current wild type motivates the application of extreme value theory [103] to fitness. Of particular note, the model predicts that the distribution of fitness effects of adaptive mutations is exponential and that a population should often fix (become homogeneous for) the most beneficial mutation possible [104, 105]; recent experimental results are consistent with both predictions [106, 107].

R. A. Fisher's geometrical model focuses on the phenotype, considering adaptation in an N-dimensional "trait" space [108]. Fisher used the geometrical model to argue that evolution is driven by the accumulation of many mutations of small effect. This argument was influential until Motoo Kimura pointed out that selection favors the fixation of mutations with larger effect, implying that it is mutations of intermediate effect that are most likely to be fixed in the population and thus drive observable evolution [109]. Recent studies have applied Fisher's model to a gamut of questions in evolutionary biology and population genetics; these include sequential adaptations [110, 111], the load of deleterious mutations carried by finite populations [112, 113], and organismal complexity and its evolutionary "cost" [114, 115, 116]. Predictions from the model regarding epistasis compare favorably with data [117], and the distribution of fitness effects the model predicts is consistent with the mutational landscape model [118]. The abstract nature of trait space, however, can lead to difficulty when interpreting model predictions [119, 120, 121].

An organism's genotype determines its phenotype through the function of complex biochemical networks. The field of systems biology is beginning to decipher

the organization and dynamics of these networks [2], and several groups have begun applying this knowledge to evolutionary theory [122, 123]. Here we focus on the evolution of an organism's "chemotype",[1] the set of biochemical parameters (binding affinities, reaction rate constants, etc.) that quantify the dynamics of the organism's biochemical networks. We use a version of Fisher's geometrical model and apply it to adaptation in chemotype space, in which pleiotropy is minimal. We focus on the distribution of fitness effects of adaptive mutations, both before and after fixation. We find that the model predicts that the distribution for fixed mutations should exhibit a striking pattern of cusps and that mutations in the high-fitness tail of the distribution should all involve a single biochemical parameter. Using extreme value theory, we show that these predicted effects should be experimentally observable, even when a great many biochemical parameters are relevant to fitness.

5.3 The Model

We consider adaptation of the N biochemical parameters that quantify an organism's biochemical network, and we refer to such a set of parameters as an organism's chemotype. These parameters might include, for example, the binding affinity between a given protein and regulatory region of DNA or the rate constant for a particular enzyme.

As illustrated in Figure 5.1, an organism's chemotype can be represented as a point, $\vec{z} = z_1, z_2, \ldots z_N$, in N-dimensional space. The relative change in biochemical parameter values caused by a mutation can be described by an N-dimensional

[1] To our knowledge, the term "chemotype" has been used in two other contexts. It has been used to refer to common structural features in related organic compounds (a usage similar to "moiety") [124], and to refer to strains of plants [125] or bacteria [126] that are morphologically similar but that differ in their production of particular chemicals.

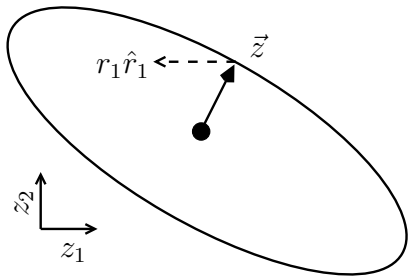


Figure 5.1: We consider evolution in biochemical space, where a population is characterized by N biochemical parameters, its "chemotype". The current and uniform chemotype $\hat{z}$ of a population is indicated by the solid arrow, and the optimal chemotype is indicated by the dot and is the origin of our coordinate system. The ellipse traces a contour of constant fitness. In this model mutations change one parameter at a time; the dashed arrow indicates an adaptive mutation of magnitude r_1 in parameter k_1.

vector $\vec{r}$; the mutant has chemotype $\vec{r} + \vec{z}$.

Importantly, single-nucleotide changes are the dominant type of mutation in short-term evolution. Such a mutation will typically only change a single region of a protein or a single DNA binding site, corresponding to one or a few biochemical parameters, so that most pairs of mutations are *orthogonal* in chemotype space. In biological terms, we expect minimal pleiotropy at the biochemical level. Thus in the model we restrict our mutations to those which change a single parameter at a time, so that $\vec{r} = r\hat{r}_i$, where r is the size of the mutation and $\hat{r}_i$ indicates that the mutation affects parameter i. This distinguishes our model from most of versions of Fisher's model, which consider maximal pleiotropy where mutations can change all traits simultaneously. (Other authors have considered zero pleiotropy models in the context of drift load [113, 127] or restricted pleiotropy as a form of modularity [115].)

Close to the optimum chemotype, any smooth fitness landscape can be approximated by a quadratic form, and comparisons between empirical mutation effect distributions in different environments for several organisms support a Gaussian

form [128]. Thus we study a Gaussian fitness landscape:

$$W(\vec{z}) = \exp\left(-\frac{1}{2}\vec{z}^{\mathsf{T}}\mathbf{S}\vec{z}\right), \tag{5.1}$$

where $\mathbf{S}$ is a positive definite matrix. Many of the analytic results below are derived for spherically symmetric fitness functions, for which $\mathbf{S} = \lambda\mathbf{I}$, where $\mathbf{I}$ is the identity matrix. We show numerically, however, that our qualitative conclusions are robust to even dramatically non-spherical fitness functions.

In this manuscript we work in the limit of strong selection and weak mutation, so that the population is genetically homogenous aside from rare mutants that arise one at a time and either fixate or go extinct before the next mutation arises. In this case, the state of the entire population corresponds to a single point $\vec{z}$ in chemotype space, and fixation of the mutation $\vec{r}$ moves the entire population to chemotype $\vec{z} + \vec{r}$.

Finally, in our analyses it is convenient to work with the logarithmic fitness change Q introduced by Waxman and Welch [129] and defined as

$$Q \equiv \log\left[\frac{W(\vec{z}+\vec{r})}{W(\vec{z})}\right]. \tag{5.2}$$

Equivalently, $Q = \log(1+s)$, where s is the selection coefficient. Mutations with $Q > 0$ are adaptive, and note that

$$Q(r_i) = -\vec{z}\mathbf{S}\hat{r}_i r_i - \frac{1}{2}\hat{r}\mathbf{S}\hat{r}_i r_i^2. \tag{5.3}$$

5.4 Results

5.4.1 Typical Mutation Size

In our model, the dynamics of the evolutionary process depend on (1) the shape of the fitness landscape defined by $\mathbf{S}$, (2) the initial state $\vec{z}$ of the population, and

(3) the distribution of mutational effects r on biochemical parameters. In this section we consider what the typical size of mutation parameter effects must be to reproduce the observation that most non-neutral mutations are deleterious [130, 131]. To do so, we calculate what the probability of beneficial mutation P_{ben} would be if the distribution of mutational effects was identical for all parameters and uniform over the range of possible beneficial mutations. This situation leads to an unrealistically high probability of beneficial mutation, even in the limit of large N. This indicates that the distribution of mutation parameter effects must have a scale larger than that of the largest possible beneficial mutation; attempted mutations must often 'hop over' the region of possible beneficial mutations.

The largest mutation ρ_i that can be made to parameter i without decreasing the fitness is found by solving $Q(r_i) = 0$ (see Equation 5.3), yielding

$$\rho_i = 2\frac{|\vec{z}\mathbf{S}\hat{r}_i|}{\hat{r}_i\mathbf{S}\hat{r}_i} \tag{5.4}$$

$$= 2|\vec{z}\cdot\hat{r}_i|, \tag{5.5}$$

where the second expression for ρ_i (Equation 5.5) specializes to a spherical fitness function and is simply twice the magnitude of the ith component of $\vec{z}$. Intuitively, in the spherical case, the fitness is proportional to $|\vec{z}|^2 = \sum_i |z_i|^2$. A mutation of size ρ_i of the proper sign simply changes the sign of z_i, leaving the fitness unchanged. Smaller mutations of that sign reduce $|z_i|$ and thus increase the fitness.

If the probability density of mutation chemotype effects is uniform over $\pm\max_i \rho_i$, the probability of a random mutation being adaptive is

$$P_{\text{ben}} = \frac{1}{2N}\frac{\sum_i \rho_i}{\max_i \rho_i} \tag{5.6}$$

$$= \frac{\langle|\hat{z}\cdot\hat{r}_i|\rangle}{2\max_i |\hat{z}\cdot\hat{r}_i|}. \tag{5.7}$$

Asymptotically for large N, $\hat{z}\cdot\hat{r}_i$ has a Gaussian probability density with variance $1/N$, which implies that $\langle|\hat{z}\cdot\hat{r}_i|\rangle = \sqrt{2}/\sqrt{\pi N}$. The largest absolute value of N

samples drawn from from a Gaussian density with variance $1/N$ is asymptotically $\sqrt{2\log\left(N/\sqrt{2\pi}\right)/N}$ [103]. Thus

$$P_{\text{ben}}(N) \sim \frac{1}{2\sqrt{\pi\log(N/\sqrt{2\pi})}}. \tag{5.8}$$

This probability remains substantial even for large N (e.g. $P_{\text{ben}}(10,000) \approx 0.1$). This suggests that, for a realistically large fraction of mutations to be deleterious, the typical scale of chemotype effects for mutations must be larger than $\max_i \rho_i$. Thus, for every parameter the mutation leading to the largest possible increase in fitness is accessible.

Little is known empirically about the distribution of effects of random mutations on biochemical parameters, and the complete distribution of fitness effects for all mutations depends sensitively on this distribution. The distribution for adaptive mutations, on the other hand, depends only on the small effect tail of the distribution for chemotypic effects. The above argument shows that the typical scale for chemotypic effects of mutations must be larger than the scale corresponding to the largest beneficial mutation. Thus below we make the approximation that the distribution of mutation chemotype effects is identical for all parameters and is uniform over the range required to generate the largest beneficial mutation.

5.4.2 Adaptive Mutation Probability Densities

The probability density of fitness effects for adaptive mutations $f_{\text{a}}(Q)$ is

$$f_{\text{a}}(Q) \propto \sum_i \int dr_i f(r_i)\,\delta(Q - Q(r_i)), \tag{5.9}$$

where $f(r_i)$ is the probability density of chemotypic mutation effects. Making the variable substitution $u = Q(r_i)$ yields

$$f_{\text{a}}(Q) \propto \sum_i \int \frac{du\, f(r_i(u))\,\delta(Q-u)}{\sqrt{(\vec{z}\mathbf{S}\hat{r}_i)^2 - 2\hat{r}_i\mathbf{S}\hat{r}_i u}}. \tag{5.10}$$

Assuming that $f(r_i)$ is uniform over the range where $Q > 0$ yields

$$f_{\mathrm{a}}(Q) \propto \sum_i \frac{1}{\sqrt{\hat{r}_i \mathbf{S} \hat{r}_i}\sqrt{\zeta_i - Q}}. \tag{5.11}$$

The typical scale ζ_i for fitness effects of mutations of parameter i is

$$\zeta_i \equiv \frac{(\vec{z}\mathbf{S}\hat{r}_i)^2}{2\hat{r}_i \mathbf{S} \hat{r}_i} \tag{5.12}$$

$$= \frac{\lambda |\vec{z} \cdot \hat{r}_i|^2}{2} \equiv Q_0 |\hat{z} \cdot \hat{r}_i|^2. \tag{5.13}$$

where the second line specializes to the spherically symmetric fitness function. In that expression $\hat{z}$ is a unit vector along $\vec{z}$ and

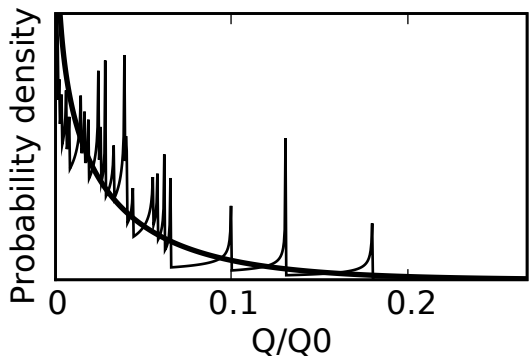
$$Q_0 \equiv -\log W(\vec{z}) \tag{5.14}$$

is Q corresponding to a mutation that yields the global optimal fitness.

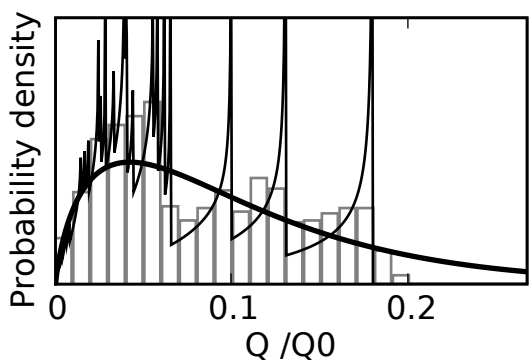
The probability density $f_a(Q)$ for a spherical fitness function is plotted in Figure 5.2(a) for $N = 30$ and $\vec{z}$ a single random unit vector. At each ζ_i the density has a singular cusp, corresponding to mutations that yield the optimal fitness attainable by changing parameter i. Intuitively, the range of mutations Δr_i about r_i that produce fitnesses in a given range ΔW is inversely proportional to the slope of $W(r_i \hat{r}_i)$. At each fittest mutation $W(r_i \hat{r}_i)$ has zero slope along $\hat{r}_i$, yielding a cusp.

The ensemble average $f_{\mathrm{a,e}}(Q)$ of the probability density $f_a(Q)$ over different initial $\vec{z}$ can be calculated by integrating $f_a(Q)$ (Equation 5.11) over the probability density of ζ_i (Equation 5.15). For a spherical fitness function, the ζ_i are proportional to the squared magnitudes of the components of the unit vector $\hat{z}$. Asymptotically as the number of dimensions $N \to \infty$, these are squares of Gaussian variables and have probability density

$$f(\zeta_i) \propto \exp\left[-\zeta_i N/(2Q_0)\right]/\sqrt{\zeta_i}, \tag{5.15}$$



(a) Adaptive mutations



(b) Fixed mutations

Figure 5.2: (a) Plotted is the probability distribution of the fitness effect of adaptive mutations for $N = 30$, a spherical fitness function, and a particular random $\vec{z}$. The singular cusps occur at each ζ_i. The smooth curve is the ensemble average approximation corresponding to maximal pleiotropy. (b) Shown is the probability distribution of fitness effects for fixed mutations in the large-population limit for the same $\vec{z}$ as in (a). Notice how the cusps at large Q are much more prominent. The histogram corresponds to 1000 samples from the distribution, each smeared by a Gaussian to mimic a 1% error in the measurement of Q/Q_0. With this level of measurement noise the cusps are not distinguishable. The smooth curve is the ensemble average approximation to the probability distribution.

which is a χ^2 density with one degree of freedom. Taking this average also corresponds to considering to the alternative model of maximum pleiotropy, in which a single mutation can change all parameters [129]. For the spherical fitness function the result is:

$$f_{\mathrm{a,e}}(Q) = \int_Q^\infty f_{\mathrm{a}}(Q) f(\zeta_i) d\zeta_i \tag{5.16}$$

$$\propto \exp(-QN/(4Q_0))\, \mathrm{K}_0(QN/(4Q_0)), \tag{5.17}$$

where K_0 is the zero-order modified Bessel function of the second kind. The smooth solid curve in Figure 5.2(a) shows this ensemble average, which is very similar to the distributions predicted by other theories [132]. The ensemble average corresponds to averaging over populations with different initial chemotypes; repeated experiments with identical initial populations in identical environments will yield the cusped distribution.

In the limit of an infinitely large population, the probability that an adaptive mutation with fitness effect Q fixates in the population is, for small Q, proportional to Q [109, 133]. The probability density of fitness effects for fixed mutations $f_f(Q)$ is thus

$$f_f(Q) \propto Q f_a(Q). \tag{5.18}$$

This density of fixed mutations is shown in Figure 5.2(b) for the same spherical fitness function and initial chemotype $\vec{z}$ as in Figure 5.2(a). Note that the cusps at large Q are much more prominent in the distribution of fixed mutation fitness effects. We now turn to the question of how difficult these predicted cusps are to observe in evolution experiments.

5.4.3 Cusp Spacings

Experimental measurements of the distribution of mutation effects of fixed mutations are limited by two factors: (1) Beneficial mutations rarely arise, and those that do are often lost to genetic drift without fixing in the population. Thus studies tend to have few samples from the distribution. (2) Experimental uncertainties in the fitness measurements blur out fine features in the distribution. This second effect is illustrated by the histogram in Figure 5.2(b). It represents 1000 samples from $f_f(Q)$, each of which has been polluted by Gaussian noise in the measurement of Q/Q_0 with standard deviation 0.01. Even given this large number of samples, the cusps are not resolved due to the errors in fitness measurement. It will be experimentally challenging to observe these cusps directly.

Note that each cusp in Figure 5.2 corresponds to mutations affecting a different biochemical parameter ζ. Our model thus not only predicts cusps, but also that the most beneficial mutations will all affect the *same* biochemical parameter. To experimentally observe this prediction, it suffices to measure relative fitness differences of order Δ, where $\Delta \equiv (\zeta_1 - \zeta_2)/\zeta_1$ is the separation between the two cusps with the highest fitness, normalized by the fitness of the fittest cusp. We derive the distribution of Δ predicted by our model in Appendix 5.A, using the methods of extreme value theory [103].

The solid line in Figure 5.3 is the exact asymptotic result (using Equations 5.23 and 5.24) for the mean of Δ given a spherical fitness function. The dashed line is an approximation to this result:

$$\langle \Delta \rangle \approx \frac{1}{\log N + \log\left(\sqrt{2/\pi}\right) + 1}, \tag{5.19}$$

which is valid for large N. The black circles in Figure 5.3 are the results from 1000 numerical simulations at each N using Equation 5.13. The agreement between the exact asymptotic result and the numerical simulations is excellent, and the

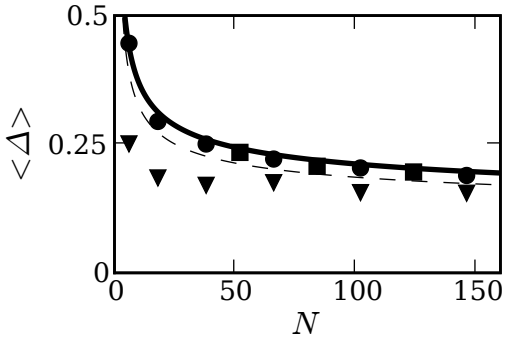


Figure 5.3: Plotted is the mean relative spacing $\langle\Delta\rangle$ between the two cusps with the largest ζ in the adaptive mutation distribution. The solid line is the asymptotically exact result from extreme value theory for spherical fitness functions, while the dashed line is the approximation of Equation 5.19. The circles are numerical simulations for the spherical fitness functions, while the squares and triangles are simulations results for mildly and severely non-spherical fitness functions, respectively. The mean value of Δ declines very slowly with N, suggesting that the cusps will be well-separated for even very large N.

approximate result captures the trend well. Note that $\langle\Delta\rangle$ declines very slowly as a function of N; for a chemotype with $N = 10,000$ relevant parameters the mean Δ is approximately 0.11, a relative fitness difference that is straightforward to measure experimentally. For comparison, Figure 5.2 has $\Delta \approx 0.27$, which is approximately the predicted $\langle\Delta\rangle$ for $N = 30$.

Thus our model predicts, even for a large number of relevant mutating parameters, a substantial range Δ of the most beneficial mutations will all affect the same biochemical parameter.

5.4.4 Non-spherical Fitness Functions

The analytic results in the previous section are all derived for spherical fitness functions and uniform distributions of chemotypic mutation effects. In this section we consider non-spherical fitness landscapes to test the generality of our result

that the largest two cusps in $f_f(Q)$ should be well separated even for large N. Note also that any differences in typical size of chemotypic mutation effects on different parameters can be eliminated by rescaling the parameters z_i, so considering non-spherical fitness functions implicitly also considers different mutation scales amongst the parameters. Spherical fitness functions have all eigenvalues of $\mathbf{S}$ equal, while for non-spherical functions the width of the fitness contour along any given eigenvector of $\mathbf{S}$ is proportional to the square root of the corresponding eigenvalue λ.

For a given distribution of eigenvalues, $\langle\Delta\rangle$ can be calculated numerically from the definition of ζ (Equation 5.12). In the tests described below, each scenario is simulated 1000 times, each instance involving an independent $\mathbf{S}$ and initial chemotype $\vec{z}$. The eigenvectors of $\mathbf{S}$ were random orthogonal vectors and the initial chemotypes were chosen to have a fixed fitness $W(\vec{z})$. We chose the ensemble of fixed $W(\vec{z})$ rather than the ensemble of fixed $|\vec{z}|$ because the fitness is experimentally measurable while $|\vec{z}|$ is not. Additionally the ensemble of fixed $W(\vec{z})$ is invariant under rescaling of the parameters z_i. (The distribution of Δ is independent of the value chosen for $W(\vec{z})$.) Details of the procedure are described in Appendix 5.B.

The black squares in Figure 5.3 result from mildly non-isotropic fitness landscapes corresponding to eigenvalues of $\mathbf{S}$ drawn uniformly from the range $0.4 < \lambda < 3.6$, as in [134]. The deviations of $\langle\Delta\rangle$ from the spherical case are very small.

The black triangles in Figure 5.3 arise from 'sloppy' fitness landscapes [7] (Chapter 2) with the N eigenvalues spaced distributed in the logarithm from 10^6 to 10^{-6}. This corresponds to the narrowest axis of the fitness contours being one-millionth the width of the longest axes. Even with these very anisotropic fitness functions the average spacing Δ remains substantial and comparable to the average

in the spherical case.

5.5 Discussion

We have analyzed a version Fisher's geometric model in which mutations are restricted to changing only one of the N parameters at a time. This condition of minimal pleiotropy is appropriate when the population is described in terms of its chemotype, its biochemical reaction parameters, only one or a few of which will be altered by any given point mutation. We have shown the model predicts that the probability density of fitness effects of adaptive mutations will have cusps, each associated with mutations of a particular chemotypic biochemical parameter. These cusps are particularly prominent in the density of fitness effects of fixed mutations (Figure 5.2). Finally, we have shown that the relative spacing between the two cusps with the highest fitness remains substantial for large N, even for highly non-spherical fitness functions (Figure 5.3), making them experimentally distinguishable.

A key assumption of our model is that each parameter is continuously adjustable throughout the range of possible beneficial mutations. Because the genetic code is discrete, this cannot be strictly true. The distribution of effects of random mutations on biochemical parameters is not well-known, in part because most biochemical studies focus on mutations of large effect. However, studies have shown that random mutations can introduce small but non-zero changes to the enzymatic activity of proteins [135] and the expression driven by promoter sites [136]. These results suggest that our assumption of continuous parameter variation is probably reasonable. (The assumption of a flat probability distribution of mutations of small effect may be questioned, but this assumption is not crucial to the existence or observability of the effects we prediction.)

Implicit in our model is also the assumption that a genetically homogeneous population has a single set of biochemical parameters and a single fitness. Stochastic effects have been shown to be significant in several biochemical networks, which may suggest unavoidable heterogeneity between even genetically identical individuals [137]. The fitness measurements we consider, however, are performed on a population, not an individual, and averaging over a large enough population should mask any intrinsic stochasticity.

The fact that even very non-spherical fitness functions with a range in eigenvalues of 10^{12} yield a qualitatively similar cusp distribution to the spherical function (Figure 5.3) is perhaps surprising. In our simulations we assumed that the eigenvectors, and thus the correlations between the parameters, were random, and this is what leaves the distribution of ζ narrow. On average each parameter contributes about equally to each eigenvector, so the fitness function is similar when projected along each parameter direction. The assumption of random correlation structures is, however, a reasonable approximation to the complicated eigenvectors found in a study of the sensitivity of biochemical networks to parameter changes (Section 2.S1). Although even such very strong anisotropy has little qualitative effect on the probability density of the first fixed mutation, it may play a more important role for adaptive walks of many steps (Section 6.4). Analytically study of steps beyond the first may be difficult, however, because the distribution of $\vec{z}$ after the first step is not simply related by symmetry to the prior distribution of $\vec{z}$, unlike in the Fisher model with maximal pleiotropy [110].

Each cusp corresponds to mutations of a given biochemical parameter, so a substantial Δ also suggests that the mutations conveying the largest fitness benefits will typically all involve a single biochemical parameter. A similar result holds for the mutational landscape model [104, 105]. This is a possible mechanism to explain

the surprising large amount of parallel evolution that can be observed in separate populations exposed to similar environments [138, 139].

Experimental data on the probability density of fitness effects of naturally arising adaptive mutations is sparse, in large part because adaptive mutations are rare. Nevertheless, several groups have studied this density in bacteria and viruses and found that it is consistent with a smooth exponential-like curve similar to the continuum approximation seen in Figure 5.2(a) [107, 140]. The cusps our model predicts, however, are much more prominent in the probability density of fitness of effects of fixed mutations, which are even more rare. This distribution has been studied experimentally in bacteria [132, 141, 131], and those results are consistent with a smooth distribution like the continuum approximation shown in Figure 5.2(b). These studies, however, suffer from a low number of samples. (For example, the study of Barrett et al. isolated only 68 fixed mutations and could not measure relative fitness to a precision needed to resolve the cusps we predict. (better than 1%, see Figure 5.2(b))) This means that they cannot rule out the presence of the cusps our model predicts. When coupled with genetic or biochemical analysis, similar experiments to these should, however, be able to test our other prediction—that all of the most beneficial mutations in a fractional range Δ will involve changes to the same biochemical parameter. Given how slowly $\langle\Delta\rangle$ decreases with N, a relative precision of a few percent will likely be sufficient, which is achievable by averaging repeated assays.

We have studied a version of Fisher's geometrical model in biochemical parameter space, in which pleiotropy is zero. The model predicts cusps in the probability density of fitness effects of fixed mutations, and an extreme value theory analysis suggests that these cusps are likely to be experimentally accessible. Evolution has long been studied in terms of genotype and phenotype and our results show that

considering evolution in terms of biochemical parameters—the chemotype—may offer new insights.

5.6 Acknowledgments

We thank Carl Franck, whose exam question prompted this investigation. We also thank Jason Mezey and Ben Logsdon for helpful discussions relating to population genetics and evolution and Josh Waterfall and Fergal Casey for discussions of the model itself.

5.A Extreme Value Theory for Δ

Δ is a ratio of two values; to calculate its probability density we first calculate the density of $i_1 \equiv \log \zeta_1 - \log \zeta_2$, the spacing between the logarithms of the largest two ζs. Defining

$$\omega \equiv \log \left(\frac{\zeta N}{Q_0} \right) \tag{5.20}$$

and using the asymptotic χ^2 density for ζ (Equation 5.15) yields the asymptotic probability density of ω:

$$f(\omega) = \exp \left[-\frac{1}{2} \big(\exp(\omega) - \omega \big) \right] / \sqrt{2\pi}. \tag{5.21}$$

The corresponding probability distribution $F(\omega) \equiv \int_{-\infty}^{\omega} f(\omega') d\omega'$ is

$$F(\omega) = \operatorname{erf} \left(\exp(\omega/2) / \sqrt{2} \right), \tag{5.22}$$

where erf is the error function. This distribution has exponential-type extreme value statistics [103].

The typical size $u_{1,N}$ of the largest of N samples from the density $f(\omega)$ is given by $F(u_{1,N}) = 1 - \frac{1}{N}$. In our case this is

$$u_{1,N} = 2\log\left(\sqrt{2}\,\mathrm{erf}^{-1}\left(1 - 1/N\right)\right). \tag{5.23}$$

The corresponding scale parameter $\alpha_{1,N}$ is

$$\alpha_{1,N} = Nf\left(u_{1,N}\right), \tag{5.24}$$

and distance between the largest two samples i_1 has probability density[2]

$$f(i_1) = \alpha_{1,N}\exp(-\alpha_{1,N} i_1). \tag{5.25}$$

The distance between the logarithms i_1 is related to Δ by $\Delta \equiv 1 - \zeta_1/\zeta_2 = 1 - \exp(-i_1)$. Thus the probability density for Δ is

$$f(\Delta) = \alpha_{1,N}\left(1 - \Delta\right)^{\left(\alpha_{1,N} - 1\right)}, \tag{5.26}$$

and the average of Δ is

$$\langle\Delta\rangle = \frac{1}{1 + \alpha_{1,N}}. \tag{5.27}$$

A useful approximation for $\alpha_{1,N}$ can be obtained using an asymptotic expansion for erf^{-1} [142]:

$$\sqrt{2}\,\mathrm{erf}^{-1}\left(1 - x\right) \sim \sqrt{\log\left(\frac{2}{\pi x^2}\right) - \log\log\left(\frac{2}{\pi x^2}\right)}. \tag{5.28}$$

Propagating this expansion through $u_{1,N}$ and $\alpha_{1,N}$ and neglecting terms of order $\log\log N$ in the final expression yields

$$\alpha_{1,N} \approx \log N + \frac{1}{2}\log\left(2/\pi\right). \tag{5.29}$$

From this follows the approximate expression for $\langle\Delta\rangle$ in Equation 5.19.

[2]Gumbel's result for this distribution (Equation 5.3.5(4) in Reference [103]) has $\alpha_{2,N}$ in place of $\alpha_{1,N}$. In the limit $N \to \infty$ the two expressions are equal, but $\alpha_{1,N}$ is a better approximation for small N.

5.B Numerical Evaluation of $\langle\Delta\rangle$

A random set of orthogonal vectors $\vec{v}_i$ can be obtained from the eigenvectors of a matrix $\mathbf{G}$ from the Gaussian Orthogonal Ensemble; $\mathbf{G} = \mathbf{H} + \mathbf{H}^{\mathsf{T}}$ where the elements of H are standard normal random numbers. A matrix $\mathbf{S}$ with eigenvalues λ_i can then be constructed via

$$S_{j,k} = \sum_i \lambda_i v_{i,j} v_{i,k}. \tag{5.30}$$

Random chemotypes $\vec{z}$ with specified fitness $Q_0 = -\log W(\vec{z})$ can be obtained using the Cholesky decomposition $\mathbf{A}$ of $\mathbf{S}^{-1}$, defined by $\mathbf{A}\mathbf{A}^{\mathsf{T}} = \mathbf{S}^{-1}$. $\vec{z}$ is then given by

$$\vec{z} = \sqrt{2Q_0}\mathbf{A}\hat{z}, \tag{5.31}$$

where $\hat{z}$ is a random unit vector.

CHAPTER 6

POTPOURRI

6.1 Other Sloppy Systems

Chapter 2 showed that sloppiness is common, and perhaps universal, in complex systems biology models. Sloppiness, however, arises in many other contexts, from fitting interatomic potentials [10] to sums of exponentials [11]. Here we discuss a few other contexts in which sloppiness arises.

A particularly interesting example of a sloppy eigenvalue spectrum comes from Mezey and Houle's study of G matrices in Drosophila wing shape [143]. Mezey and Houle measured 20 quantitative characteristics of wing shape for 16,000 flies, scaling each characteristic by the overall size of the wing and controlling for environmental variation by propagating the flies in the lab for several generations. The G matrix is the correlation matrix for this set of measurements, or equivalently, the inverse of the Hessian approximation we would obtain from Principal Component Analysis of the data. The eigenvalues for Mezey and Houle's G matrix appear sloppy, spanning a range of 10^4 roughly evenly. A particular exciting possibility is that a sloppy G matrix could result from the population equilibrating in a fitness landscape that is itself sloppy (see Sections 5.4.4 and 6.4). However, it must be noted (1) that genealogical simulations show that such a pattern can arise in the absence of selection [144], and (2) that it is unknown whether populations typically have time to equilibrate in their fitness landscape (J. Mezey; personal communication).

6.1.1 Cornell's Proposed Energy Recovery Linac

Cornell is planning to build an energy recovery linear accelerator (ERL) [145] as an enhanced X-ray light source as an extension of the Cornell Electron Storage Ring (CESR) [146]. The design of such a huge machine is in part a large optimization problem; the desired capabilities of the machine must be optimized while constrained by financial cost.

In designing Cornell's ERL these constraints include maximizing use of the existing CESR facility, accommodating the geography of the Cornell campus, and providing the best beam feasible. The beam properties depend on the geometry of the accelerator itself and on the placement and strength of the steering magnets that constrain the beam. Our interest here is the optimization of the magnet arrangement; is it a sloppy optimization problem?

Sloppiness would suggest opportunities in the design of the accelerator. In particular, when built the accelerator will unavoidably have fixed construction errors which must be compensated for by adjusting free parameters. One may be able to take advantage of the sloppiness of the design in most efficiently compensating for these errors. Perhaps the accelerator could even be designed so that expected construction errors have substantial projection along sloppy directions of the system.

To study the accelerator, we use a simulation of a prospective ERL design created by Christopher Mayes. The simulation is implemented in Tao [147] which is built on the Bmad library [148].

The adjustable parameters in this simulation are the position and strength of magnets along the beamline; there are 217 such parameters in the simulation we study. The cost function is a weighted sum of squared residuals reflecting the various design goals and constraints for the accelerator. These include characteristics

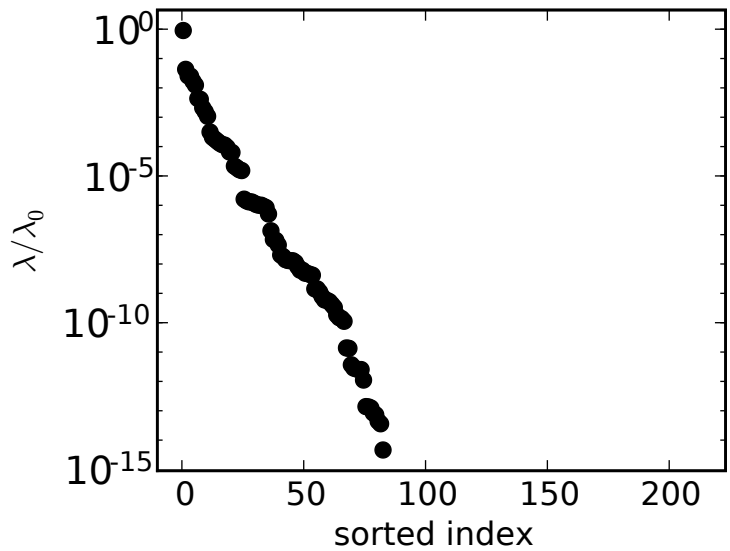


Figure 6.1: The eigenvalues for the ERL model are sloppy; spanning many decades roughly evenly. Many of the eigenvalues are zero, as the model we consider is under-constrained.

such as the beam emittance and dispersion at various points along the beamline.

Tao can output the Jacobian matrix, from which we construct the $J^{\mathsf{T}}J$ approximation to the Hessian matrix (Equation 3.4). The normalized eigenvalues of this approximate Hessian are shown in Figure 6.1. They show the huge range and even spacing characteristic of a sloppy model [7]. Many of the eigenvalues are zero. This reflects the fact that the number of active constraints, 171, is less than the number of parameters. Moreover, many of those constraints were hard walls, with a cost of zero until a given variable left some range, outside of which the cost is infinity.

The stiffest three eigenvectors are shown in Figure 6.2. Note that the parameters are approximately spatially ordered; parameters that affect magnets that are close to each other will be close together in Figure 6.2. The dominant components of the eigenvectors are well-localized, involving only a few close-together magnets at a time. This is very different from the eigenvectors of biochemical networks (Section 2.S1) and ideal sloppy problems [11], which tend to have relatively

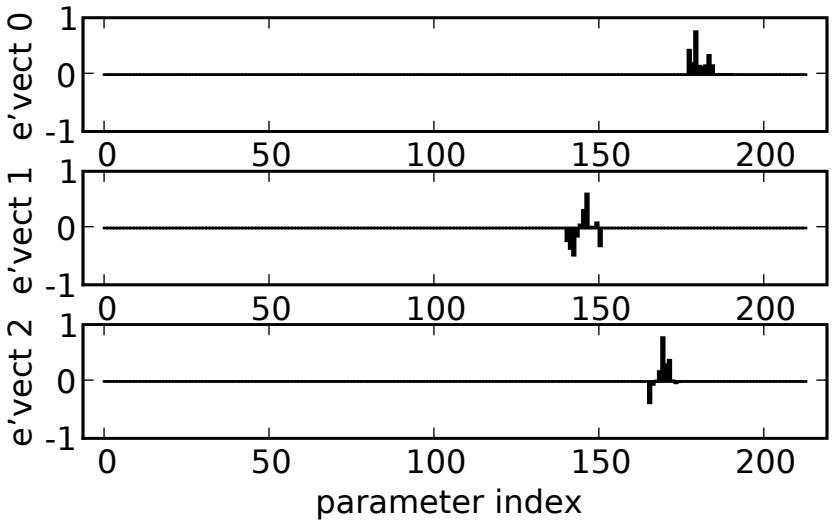


Figure 6.2: The stiffest eigenvectors of the ERL are composed of localized clumps of parameters corresponding to parameters for adjacent magnets.

random-looking eigenvectors. This may suggest that sloppiness manifests itself differently in problems that have explicit spatial features.

6.1.2 Kinematics of Insect Hovering

Berman and Wang's study of insect hovering flight [149] offers an interesting non-network biological optimization problem. Their study revealed that the observed wing motions during hovering for several insects are similar to the motions that would minimize total aerodynamic power expenditure. The potential sloppiness of this model is particularly interesting because we expect power output to be anti-correlated with fitness. Sloppiness in this model would be more circumstantial evidence that evolutionary fitness landscapes are sloppy (Sections 5.4.4 and 6.4).

In hovering flight the path of a wing can be described by 11 parameters, one of which is discrete. Although the model has relatively few parameters, evaluating its sloppiness is tricky because of the number of constrains involved; eight of the

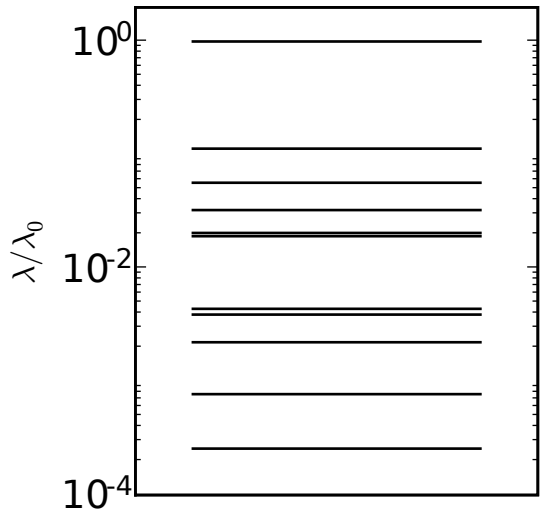


Figure 6.3: Shown is the eigenvalue spectrum resulting from a Principal Components Analysis of the ensemble of fruitfly kinematic parameters. The ensemble is built by penalizing states whose expended during hovering is more than about 10% above the minimal power.

eleven parameters are constrained to physically sensible values, and the lift must be constrained to be greater than one.

The constraints can be dealt with in a Hessian evaluation by restricting our derivatives to the manifold in which the lift is unity (G. Berman, personal communication). Here we study an ensemble of parameter sets, rather than building a Hessian matrix. The free energy is set to infinity for any parameter set which violated any constraint on parameter values or gave a lift less than one; otherwise it is set equal to average power required by the stroke. The ensemble is built without importance sampling, and the discrete parameter is chosen randomly from its two values for each attempted step. The temperature is adjusted such that typical members of the ensemble yielded a power 10% higher than the optimal power.

Figure 6.3 shows the principal component analysis eigenvectors of the ensemble for the model of fruitfly flight. Note that they are sloppy. Figure 6.4 shows the cor-

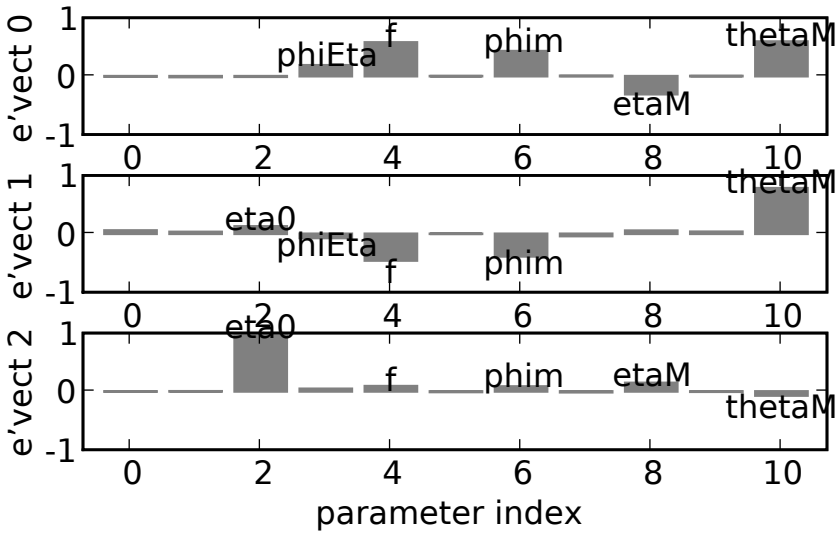


Figure 6.4: Shown are the three stiffest eigenvectors of the fruitfly kinematic parameter ensemble. Like most sloppy models, the eigenvectors are fairly complex combinations of the parameters.

responding three stiffest eigenvectors. Notice that they are complex combinations, much like is seen in the biochemical network models (Section 2.S1). The eigenvectors found in this ensemble approach differ somewhat from the eigenvectors seen in a Hessian-based analysis (G. Berman, personal communication). In particular, the parameter K has large components in the stiffest eigenvectors of the Hessian, but only small components along the stiffest PCA eigenvectors considered here.

Berman and Wang showed that the observed fruitfly wing kinematics are similar to those predicted by their model, and we've shown that the model is sloppy. It would be very interesting to compare the natural variation in kinematics to the stiff and sloppy directions predicted by the model. In particular, does the natural variation fill the basin of kinetic parameters up to some level of power expenditure? If so, it would some evidence that natural populations have enough time to equilibrate amongst all acceptably fit phenotypes.

6.2 Scale Factor Entropy and Priors

When building an ensemble, we want to sample from the distribution of parameter sets compatible with the available data. By Bayes's theorem, the relative probability $P(\theta|D)$ of any given parameter set θ given data D is proportional to $P(D|\theta)P(\theta)$. $P(D|\theta)$ is the probability of the model reproducing the data given parameters θ, and $P(\theta)$ is the *prior* probability placed on the parameters. If the data points d_i have Gaussian uncertainties σ_i, then

$$P(D|\theta) = \sum_i \exp\left[-\frac{(y_i(\theta) - d_i)^2}{2\sigma_i^2}\right]. \tag{6.1}$$

It may be necessary to add additional fit parameters when a model is compared with certain data sets. One common example is data that lacks an absolute scale. For example, the absolute intensity of a band in a Western blot depends on the properties of the antibody used and how the gel was loaded. Thus the intensity of the band only has meaning relative to the intensity of other bands. To fit such data, we must introduce scale factors B_k between theory and data [7] so that Equation 6.1 becomes

$$P(D|\theta, B) = \sum_i \exp\left[-\frac{(B_k y_i(\theta) - d_{k,i})^2}{2\sigma_i^2}\right]. \tag{6.2}$$

The index k groups data points which share a common scale factor.

Conveniently, the optimal scale factors B_k^* for any given θ are straightforward to calculate analytically. The optimal scale factor B_k^* is b_k/a_k where $a_k \equiv \sum_i y_i^2/\sigma_i^2$ and $b_k \equiv \sum_i y_i d_{k,i}/\sigma_i^2$. Thus the scale factors do not need to be considered explicitly while optimizing $P(D|\theta)$ over parameters θ. However, when building an ensemble it is important to consider all possible scale factors so that

$$P(\theta|D) = \int P(\theta|D, B)P(B)dB. \tag{6.3}$$

Here we address the choice of the scale factor priors $P(B)$.

Note that a revealing analogy can be drawn with thermodynamics if we define

$$P\left(\theta|D\right) \equiv \exp\left[-G(\theta,T)/T\right] = \exp\left[-(C(\theta) - TS(\theta,T))/T\right]. \tag{6.4}$$

In analogy with the Boltzmann distribution, this defines the free energy $G(\theta,T)$, the energy (cost) $C(\theta)$, and entropy $S(\theta,T)$. We typically take the cost to be $P(\theta|D,B^*)$, the likelihood given the optimal scale factors. Then the entropy accounts for the contribution of scale factor fluctuations, similar to how the entropy in thermodynamics accounts for fluctuations in variables that have been integrated out of consideration [150].

A straightforward and computationally convenient choice is to take the prior on B to be uniform on the infinite interval [7]. Then

$$P\left(\theta|D\right) = \prod_k \exp\left(b_k^2/2a_kT\right)\sqrt{\frac{2\pi T}{a_k}}. \tag{6.5}$$

This choice, however, is problematic for several reasons.

In practice the infinite uniform prior weights parameter sets more heavily if they lead to small a_k and thus large scale factors. An ensemble can thus evaporate toward regions with large scale factors and correspondingly small theory curves y. This is illustrated in Figure 6.5, where the dark blue curve shows the free energy ($\propto \log P\left(\theta|D\right)$) over the course of an ensemble run for the PC12 model [8]. The free energy diverges toward $-\infty$ while the cost ($\propto \log P\left(D|\theta,B^*\right)$, cyan curve) increases dramatically, indicating that the fit of the model to the data is poor. Additionally, the network equations become much more difficult to integrate when the theory values y differ dramatically in scale. This is reflected by the long stretches of constant free energy and cost seen in Figure 6.5, which indicate that the acceptance ratio is very low, probably because many of the integrations are failing.

Even if the infinite uniform prior did not introduce practical difficulties, the fact that it allows negative scale factors is generally non-physical. Moreover, a

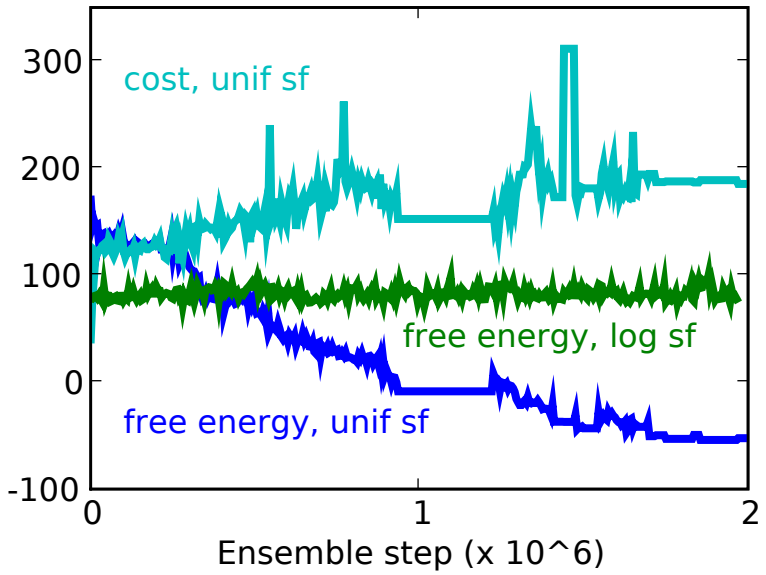


Figure 6.5: Plotted in dark and light blue are the free energy and cost (inverse quality of fit) for an ensemble built for the PC12 model [8] using infinite uniform priors on the scale factors. In green is the free energy in an ensemble built using weak Gaussian priors on the logarithms of the scale factors. This avoids the pathologies of the infinite uniform priors.

uniform scale factor prior is inconsistent with how we treat model parameters that we explicitly optimize. We typically work in the logarithms of the parameters because we recognize that biochemical parameter values can have widely varying scales. Biochemical concentrations and thus scale factors can also vary over a wide range of scales, so it is appropriate to deal with logarithms of scale factors as well.

Unfortunately, a uniform prior on the logarithm of the scale factors yields a divergent $P(\theta)$ because the fit does not become infinitely bad as the scale factors tend to 0. Thus we need to apply a more restrictive prior.

We often work with priors on the explicit parameters that are Gaussian in their logarithm. Taking a similar prior on the logarithm of B_k yields

$$P(\theta) = \int_{-\infty}^{\infty} P(\theta | \log B_k) \, P(\log B_k) \, d \log B_k \tag{6.6}$$

$$= \exp\left(b_k^2 / 2a_k T\right) \int_{-\infty}^{\infty} \exp\left(-a_k/2T \left(\exp(\log B_k) - B_k^*\right)^2\right) P(\log B_k) \, d \log B_k, \tag{6.7}$$

where $P(\log B_k)$ is Gaussian with a specified mean and standard deviation. The above integral cannot be done analytically, but it is numerically well-behaved, and can be computed quickly compared to the typical time for evaluating a model's dynamics.

The green curve in Figure 6.5 shows the free energy over an ensemble run using a Gaussian prior on $\log B$. It lacks the pathologies seen in the uniform prior case, and as long as the prior is chosen quite loosely (large standard deviation), the resulting predictions are expected to be broadly insensitive to it.

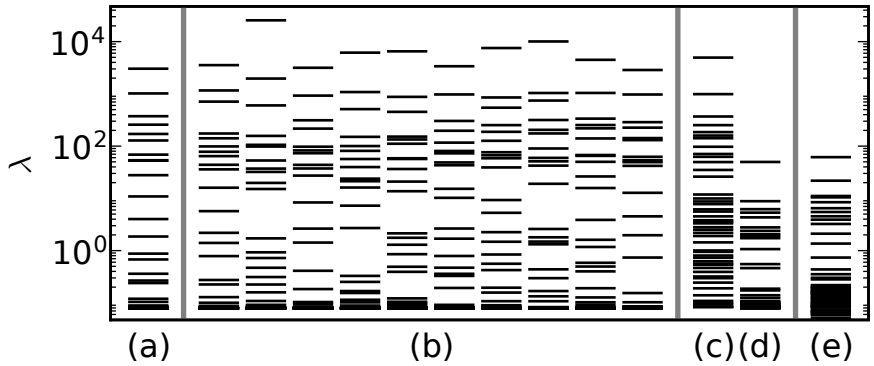


Figure 6.6: Plotted are the eigenvalues of several variants of the Hessian matrix calculated for the PC12 model. (a) $J^{\mathsf{T}}J$ at the best-fit parameters. (b) Ten $J^{\mathsf{T}}J$ matrices from parameter sets distributed along an ensemble. (c) The average of the $J^{\mathsf{T}}J$ matrices along the ensemble. (d) The inverse of the average of the inverses of the $J^{\mathsf{T}}J$ matrices in along the ensemble. (e) $J^{\mathsf{T}}J$ derived from a Principal Component Analysis of the ensemble. Note that at all points in the ensemble the cost landscape is locally sloppy as evidenced by the eigenvalues. The cost basin must be curved, however, as evidenced by how much less stiff the PCA eigenvalues are.

6.3 Faster Monte-Carlo Convergence in Curved Landscapes

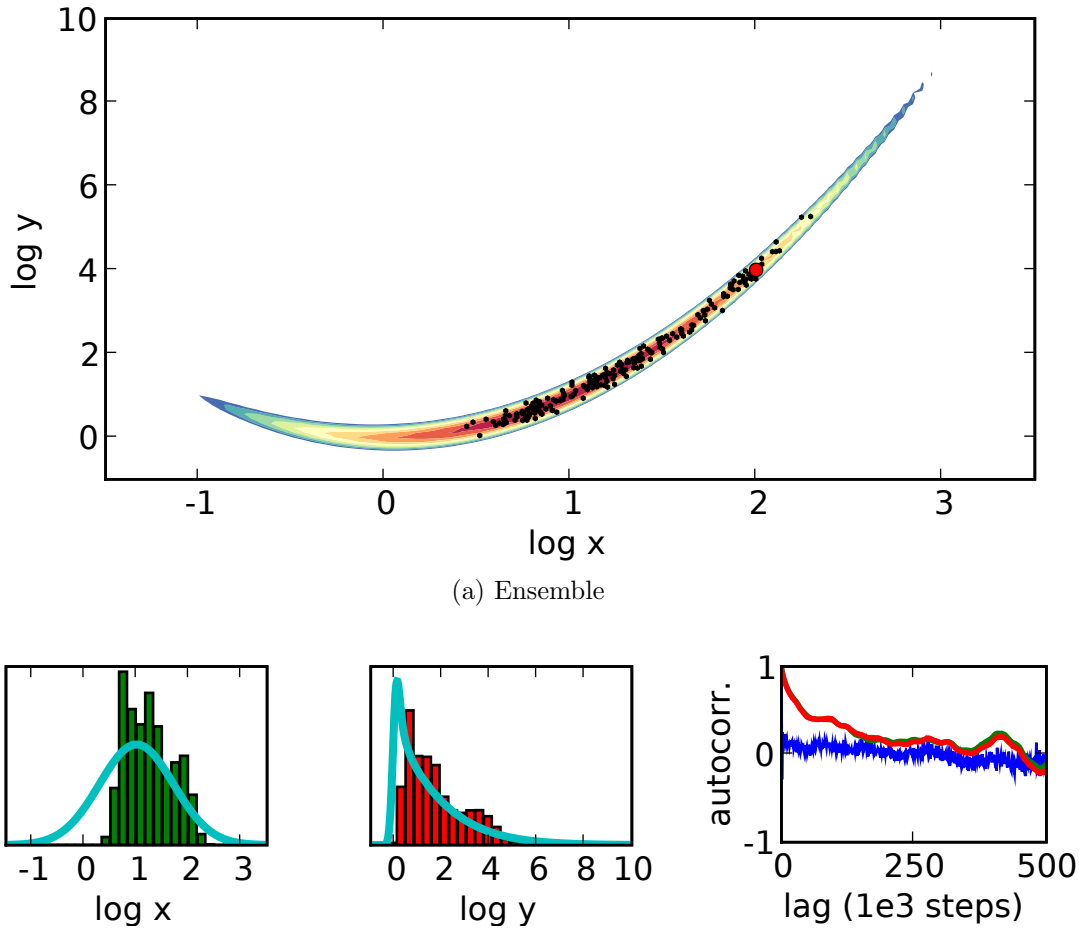
The cost landscapes of the models we study are not only sloppy; they are also curved. This is indicated in Figure 6.6, which shows the eigenvalues of several different Hessian-type matrices for Brown et al.'s PC12 model [8]. Column (a) corresponds to $J^{\mathsf{T}}J$ at the best-fit set of parameters, and the eigenvalues are classically sloppy. The ten columns labeled (b) correspond to $J^{\mathsf{T}}J$ calculated using ten different parameter sets drawn from an ensemble built for the model. All these eigenvalue spectra are sloppy, demonstrating that the cost landscape is everywhere locally sloppy, at least in the region of acceptable model fits.

Although the cost landscape is everywhere sloppy, the stiff and sloppy directions differ between members of the ensemble. This is indicated by columns (c) and (d).

Column (c) shows the eigenvalues of $\left\langle J^{\mathsf{T}}J\right\rangle_{\text{ens}}$, the average of the $J^{\mathsf{T}}J$ matrices calculated over the ensemble. If a parameter combination is stiff in any one $J^{\mathsf{T}}J$ included in the average, it will tend to be somewhat stiff in the final average. Column (d) shows the eigenvalue spectrum of $\left\langle J^{\mathsf{T}}J^{-1}\right\rangle_{\text{ens}}^{-1}$, the inverse of the average of the inverses of the $J^{\mathsf{T}}J$ matrices calculated over the ensemble. In this case, a parameter combination that is sloppy in any one of the averaged $J^{\mathsf{T}}J$ will tend to be somewhat sloppy in the final average. Note that spectrum (c) has more large eigenvalues than any of the $J^{\mathsf{T}}J$ in (a) or (b) while spectrum (d) has more small eigenvalues. This indicates that along the ensemble the stiff and sloppy directions are changing substantially; many more directions are stiff in (c) and many fewer in (d).

The curvature of the cost basin causes computational difficulties when building an ensemble. In general, we need to use *importance sampling*, choosing steps in our random walk guided by some sampling matrix that avoids large steps in stiff directions, because such steps would yield a very low acceptance probability. But if a direction that is sloppy at the best fit is stiff elsewhere, the sampling matrix calculated at the best energy will not help us avoid large steps in that direction. Even more troubling, if the landscape curves so that a stiff direction from the best-fit becomes a sloppy direction, the random walk will explore that direction very slowly.

The difficulties of a curved free energy landscape can be well-illustrated in a two-dimensional example. Figure 6.7 shows results of a 10^6 step ensemble where



(a) Ensemble

(b) $\log x$ projection

(c) $\log y$ projection

(d) Autocorrelations

Figure 6.7: Shown are results from a 10^6 step importance-sampled Markov-Chain Monte Carlo run with a free energy landscape defined by the Rosenbrock function (Equation 6.8) with $R = 30$. (a) The contours trace the free energy up to $G = 4$, and the black dots are every five-thousandth member of the ensemble. The red circle shows where the sampling matrix was calculated. (b) and (c) Histograms of the ensemble projections along $\log x$ and $\log y$ are compared with the expected distributions (cyan curves). (d) The autocorrelation of the free energy (blue), $\log x$ (green), $\log y$ (red) are shown. The correlation times for $\log x$ and $\log y$ are a substantial fraction of the total length of the ensemble.

the free energy was given by the Rosenbrock function with $R = 30$.[1] Even after 10^6 steps this ensemble is nowhere near converged. The fact that the autocorrelation times for $\log x$ and $\log y$ are each about 1/4 the total length of the ensemble (Figure 6.7d) suggests that this ensemble represents at most four independent draws from the desired distribution.

One way to cope with a curved basin is to recalculate the sampling matrix every step. When doing this, the step acceptance probability must be modified to satisfy detailed balance [84]. SloppyCell generate moves from a multidimensional Gaussian matrix whose inverse correlation matrix is $J^\mathsf{T}J$. When considering the move from parameters θ_1 to parameters θ_2, the acceptance probability must now be

$$\alpha(1 \to 2) = \frac{\exp(-G(\theta_2)/T)}{\exp(-G(\theta_1)/T)} \cdot \frac{|J^\mathsf{T}J_2| \exp\left[-\frac{1}{2}(\theta_2 - \theta_1)^\mathsf{T} J^\mathsf{T}J_2(\theta_2 - \theta_1)\right]}{|J^\mathsf{T}J_1| \exp\left[-\frac{1}{2}(\theta_2 - \theta_1)^\mathsf{T} J^\mathsf{T}J_1(\theta_2 - \theta_1)\right]}, \tag{6.9}$$

where $J^\mathsf{T}J_1$ and $J^\mathsf{T}J_2$ are $J^\mathsf{T}J$ matrices calculated at the current and proposed parameter sets, respectively, $G(\theta_1)$ is the free energy at θ_1, and $|J^\mathsf{T}J_1|$ is the determinant of $J^\mathsf{T}J_1$.[2] The second term above reflects the different probabilities for attempting a step from x to y and from y to x.

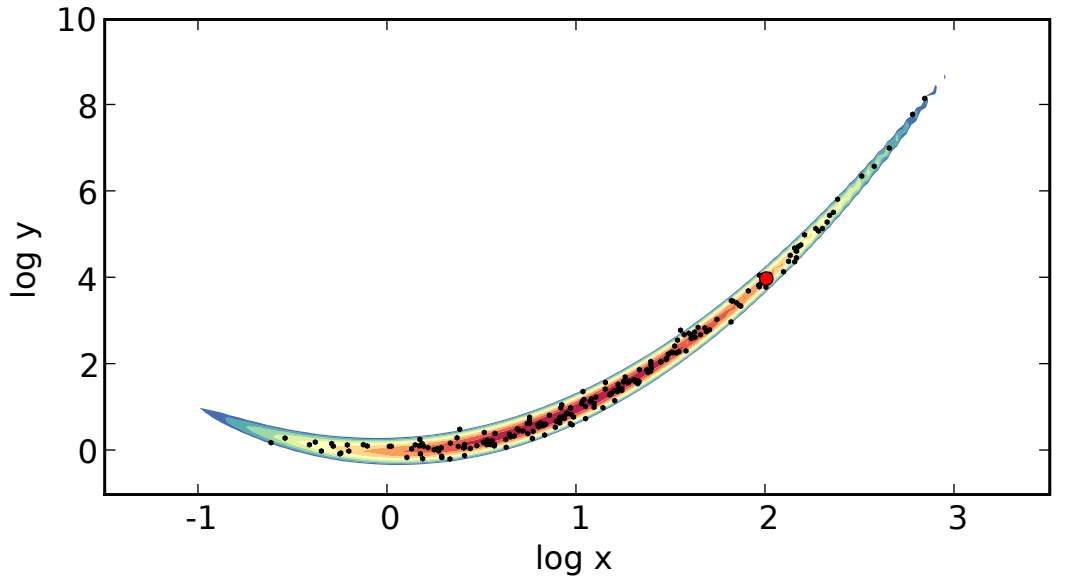
Figure 6.8 shows the results of a 10^4 step ensemble in the Rosenbrock function using the algorithm where $J^\mathsf{T}J$ is recalculated at each step. It has clearly converged much better than the previous ensemble, in approximately 1/25th the computer time.

[1] The Rosenbrock function is a classic test function for optimization algorithms [151] and, in terms of logarithmic variables, has the form:

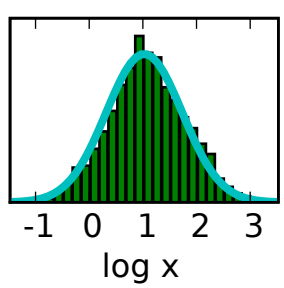
$$G(\log x, \log y) = (1 - \log x)^2 + R\left(\log y - (\log x)^2\right)^2. \tag{6.8}$$

As seen in figure 6.7, the basin for this function is both narrow (controlled by R) and curved. The equilibrium distribution for $\log x$ is $\rho(\log x) \propto \exp\left[-(\log x - 1)^2\right]$ and the distribution for $\log y$ can be calculated numerically.

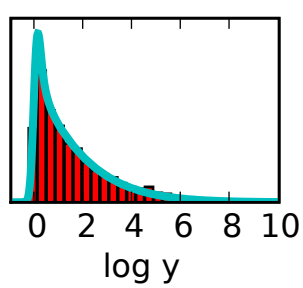
[2] In the version of this thesis submitted to the Cornell graduate college, Eq. 6.9 erroneously omitted the factors of $\frac{1}{2}$ in the $\exp[\ldots]$ terms.



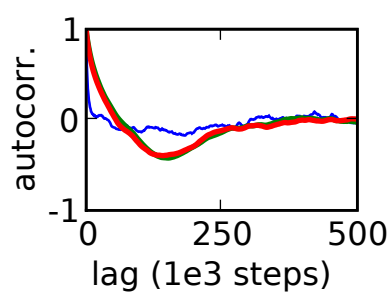
(a) Ensemble



(b) $\log x$ projection

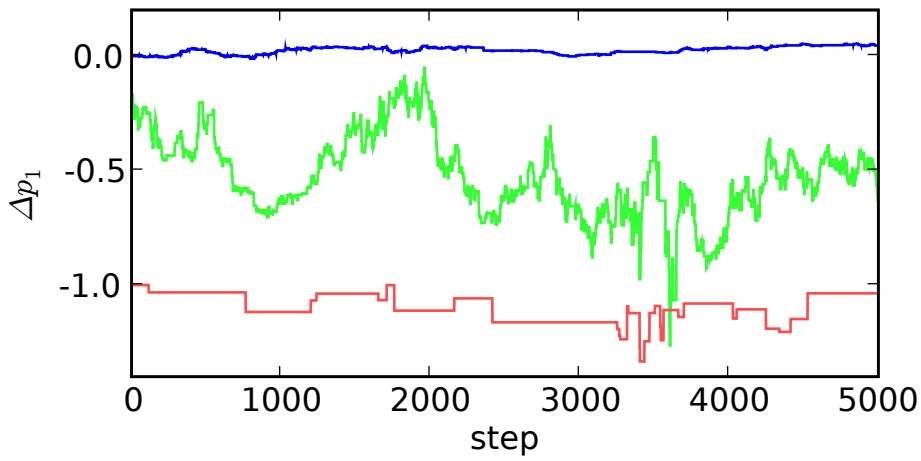


(c) $\log y$ projection

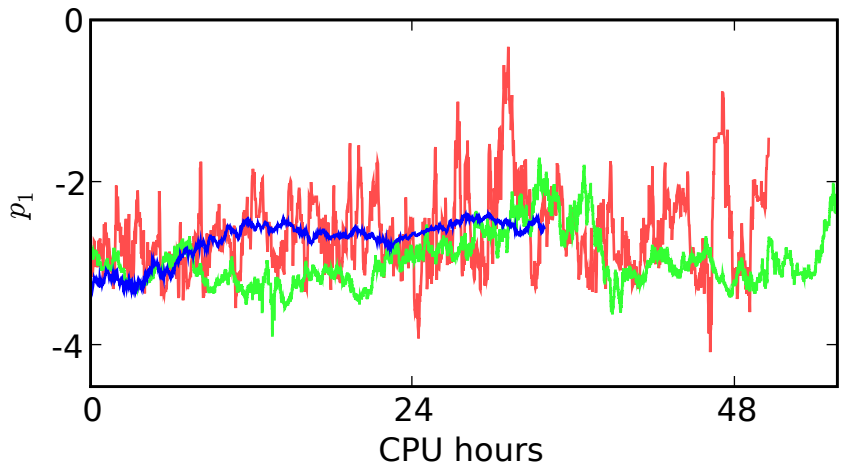


(d) Autocorrelations

Figure 6.8: Shown are results from a 10^4 step Markov-Chain Monte Carlo run over the same free energy landscape as in Figure 6.7, but recalculating the sampling matrix each step. (a) The black dots are every fiftieth member of the ensemble. The red circle shows where the ensemble began. (b) and (c) Histograms of the ensemble projections along $\log x$ and $\log y$ are compared with the expected distributions (cyan curves). (d) The autocorrelation of the free energy (blue), $\log x$ (green), $\log y$ (red) are shown. The correlation times for $\log x$ and $\log y$ are now a small fraction of the total length of the ensemble.



(a) Early steps



(b) Long-term coverage

Figure 6.9: Shown are the projections of ensembles run using three algorithms onto the stiffest eigenvector the the best-fit $J^{\mathsf{T}}J$. In blue is an ensemble built using the best-fit $J^{\mathsf{T}}J$ as the sampling matrix that guides the steps. The green ensemble is built recalculating $J^{\mathsf{T}}J$ every step. The red ensemble uses the PCA hessian of the blue ensemble as its sampling matrix. Note how the green ensemble makes much more progress per step, but the red ensemble is most efficient because each green ensemble step takes so much more CPU time.

For real world problems, recalculating $J^{\mathsf{T}}J$ each step may be very computationally costly. (In Brown et al.'s model [8] calculating $J^{\mathsf{T}}J$ takes 150 times as long as calculating a cost.) An alternative that performs quite well is to use a 'looser' sampling matrix than that suggested by $J^{\mathsf{T}}J$ at the best fit. The question then becomes how to choose that sampling matrix. A practical, if not elegant, method is to build a preliminary ensemble using the best-fit $J^{\mathsf{T}}J$, then use either the Principal Component Analysis hessian of that ensemble or $\left\langle (J^{\mathsf{T}}J)^{-1} \right\rangle_{\text{ens}}^{-1}$ (columns (d) and (e) in Figure 6.6). Using such a looser Hessian performs quite well, as shown by the algorithm comparison in Figure 6.9. Along the y-axis is plotted the projection of each member of the ensemble along the stiffest eigenvector of the best-fit $J^{\mathsf{T}}J$. The blue lines in the figure correspond to an ensemble run using the best-fit $J^{\mathsf{T}}J$, while the green lines correspond to recalculating $J^{\mathsf{T}}J$ each step. Notice how the much farther the $J^{\mathsf{T}}J$ ensemble goes each step in Figure 6.9(a). The red lines in the figure correspond to an ensemble built using the PCA hessian of the blue ensemble as the sampling matrix. The acceptance ratio is much lower, as evidenced by the long stretches in Figure 6.9(a) in which the ensemble does not move. In the long run, however, the red ensemble converges much more quickly than either the blue or the green, as seen in figure 6.9(b).

The cost basins for our sloppy models are quite complex. They are locally sloppy at each point, but they must also be curved. This poses a challenge for Monte-Carlo algorithms, and this challenge can be overcome by recalculating $J^{\mathsf{T}}J$ every step, or by the expediency of using a 'looser' sampling matrix.

6.4 Biochemical Evolution Beyond the First Fixation

Although numerical simulation show that a sloppy fitness landscape leaves little signature on the first step in an adaptive walk (Section 5.4.4), sloppiness may

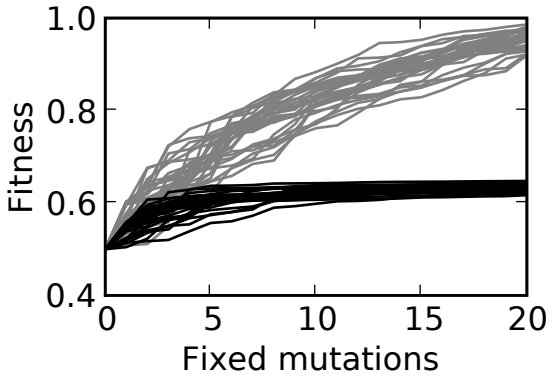
play a larger role in further steps. Here we turn to numerical simulations of longer adaptive walks to search for signatures of sloppiness. Naively simulating the model for many fixed mutations is quite computationally intensive, because adaptive mutations are rare, and they rarely fix. A Continuous-Time Monte-Carlo algorithm (Section 6.4.1), however, dramatically lowers the computational cost.

Figure 6.10 shows trajectories of adaptive walks in both spherical and non-spherical fitness landscapes. Populations in sloppy fitness landscapes seem to generically get 'trapped' at a fitness much lower than the optimum. This phenomenon is illustrated for a two-dimensional landscape in Figure 6.11. As seen in the Figure, the populations tend to become trapped along the ridge of high fitness, since a large change in either parameter will cause a decrease in fitness, and (in an infinite population) deleterious mutations cannot be fixed. Intriguingly, this trapping phenomenon may be much weaker in a smaller population, where deleterious mutations can fixate, perhaps allowing the smaller population to adapt faster. We must be cautious, however; mutations of small fitness effect take longer to fixate, and the approximation that each mutant goes extinct or fixates before the next arises may become invalid, in which case must consider interference between segregating mutations [152].

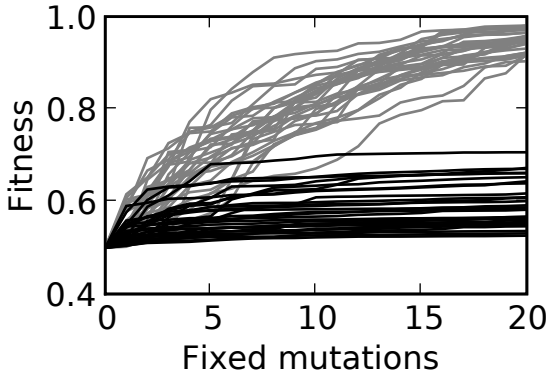
6.4.1 Continuous Time Monte Carlo Simulation

Continuous Time Monte Carlo allows us to simulate the model while tracking only the very rare fixed mutations, thus saving us from explicitly considering the many 'failed' mutations.

The initial steps are very similar to the Gillespie algorithm [96]. For parameter i in the chemotype, we calculate the total probability P_{fi} that the next fixed



(a) Fixed landscape and initial phenotype



(b) Varying landscape and initial phenotype

Figure 6.10: In both panels the gray lines correspond to thirty adaptive walks in a spherical fitness landscape with $N = 20$, while the black lines are thirty walks in an $N = 20$ sloppy landscape with log eigenvalues equally spaced over ten decades. (a) Here the landscape and initial chemotype were fixed for each set of walks. Notice that the population in the sloppy landscape seems trapped at a fitness of approximately 0.6. (b) Here each walk corresponds to a different fitness landscape and initial chemotype. As in (a), the walks in the sloppy landscape all get trapped at relatively low fitness, although there is large variance in the trapping fitness. Note that in both panels the x-axis is fixed mutations, not time. Particularly in the sloppy case, the later fixed mutations take a very long time to arise.

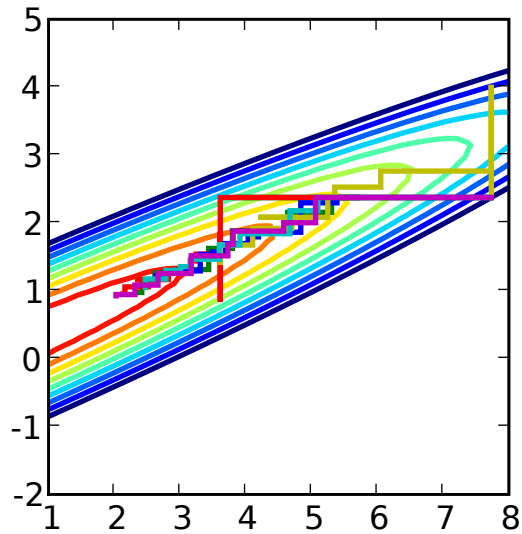


Figure 6.11: The contours show an $N = 2$ sloppy fitness landscape with the two eigenvalues differing by a factor of 100. Several evolutionary trajectories are shown starting from a single initial chemotype. Note how all of the trajectories eventually become 'trapped' along the ridge of high fitness, where a large mutation in either parameter will lead to a decrease in fitness.

mutation involves that parameter:

$$P_{fi} = \int_{-\infty}^{\infty} P_{ri}(r) P_{fix}(W(r)) dr. \tag{6.10}$$

Here r denotes the size of the mutation and $P_{ri}(r)$ is the probability density of mutation effects on parameter i. The probability of fixation for a mutant with fitness $W(r)$ is denoted $P_{fix}(W(r))$ and is a simple formula in the limit of a single mutation segregating at a time [109, 133]. We then choose parameter i to mutate among the N possible, with probability proportional to P_{fi}. Both mutation and fixation are Poisson processes, so the time (in mutations attempted) between each fixation is exponentially distributed with mean $1/\sum_i P_{fi}$; time is incremented based on this total rate.

Once we've chosen which parameter to mutate, we need to determine the magnitude of that mutation. To do so, we choose a uniform random number F^* in the range $(0, P_{fi})$, and solve

$$F^* = \int_{-\infty}^{r*} P_{ri}(r) P_{fix}(W(r)) dr, \tag{6.11}$$

where r^* is the final size of the fixed mutation. To solve the above integral equation, we convert it to a differential equation:

$$\frac{dr}{dF} = \frac{1}{P_{ri}(r) P_{fix}(W(r))} \tag{6.12}$$

and integrate it from $F = 0$ to $F = F^*$.

For finite populations and unbounded P_{ri}, there is a numerical difficulty that $F = 0$ corresponds to $r = -\infty$. To surmount this, the integration can be started at finite F, using Equation 6.11 to set the initial condition for r.

APPENDIX A

SLOPPYCELL USER DOCUMENTATION

Welcome to SloppyCell!

SloppyCell is a software environment for building and analyzing computational models of many-parameter complex systems. To date, our focus has been on biochemical network models based on the Systems Biology Markup Language (SBML) [55]. Many of the techniques and much of the code is, however, applicable to other types of models.

SloppyCell's goal is to provide a flexible environment that facilitates interactive exploratory model development while also remaining efficient for computation-intensive analyses. This goal is facilitated by our use of the computing language Python (`http://www.python.org`) for both writing and driving SloppyCell. SloppyCell scripts are Python programs, but please don't let that frighten you. Python has proved easy to learn and there are numerous tutorials (e.g. `http://wiki.python.org/moin/BeginnersGuide`) and useful books [153]. An excellent introduction to Python in the context of scientific computing is the May/June issue of Computing in Science and Engineering.

SloppyCell's development has been driven by the research interests of the Sethna group. For examples of the types of analysis we do, see our papers [7, 8, 10, 11, 12, 82] and Chapters 2 and 3.

This document opens begins with a high-level overview of SloppyCell's architecture, followed by an example application (Section A.2). We then delve into additional features and installation (Section A.3), and we close with some trou-

bleshooting.

A.1 Overview

A.1.1 Working Interactively

Python is an interpreted language, which means that there is no compile step between writing and running the code. Thus Python can be used interactively, and this is one of its greatest strengths, particularly when used with the enhanced IPython shell [92].

A typical pattern for working with SloppyCell is to have one window running the IPython shell and another open to your favorite text editor. Commands can be directly typed and run in the shell, and when the results are satisfactory they can be recorded into a `.py` script using the text editor. This pattern is particularly powerful when used with IPython's 'magic' `%run` and `%run -i` commands, which allow external scripts to be run as if all their commands were typed into the shell.

The interactive nature of Python is also important for getting help. Information and useful documentation can be accessed about any object using `help(<object>)`, and all the attributes (data and methods) of an object can be listed using `dir(<object>)`. IPython makes such exploration even more powerful, as it offers `<tab>` completion of object attributes. For example, to see all the method of the object `net` that begin with `calc`, one would type `net.calc<tab>`.

Rather than expounding upon the details of every function and method in SloppyCell, this document will focus on a higher-level description. Our aim here is to show you what SloppyCell can do and where to look for that functionality. Once you have that, the interactive help should guide you on how exactly the functions work.

A.1.2 Accessing SloppyCell

To access the tools provided by SloppyCell, most user scripts should have

```
from SloppyCell.ReactionNetworks import *
```

near the top of the script. This imports most of SloppyCell's modules into the top-level namespace where they can be easily accessed.

A.1.3 Networks

At the heart of most SloppyCell projects is a collection of `Network` objects. A `Network` describes a set of chemical species, their interactions, and a particular set of experimental conditions. SloppyCell's `Network`s are based on the SBML Level 2, Version 3 specification (http://sbml.org/documents/). A concise and mostly complete summary of this specification can be found in the paper by Hucka et al. [55] (available from http://sbml.org/documents/).

Briefly, an SBML network consists of *species* (that exist inside *compartments*) whose dynamics are controlled by *reactions* and *rules* (*assignment*, *rate*, *algebraic*, *initial assignment*). A network also describes *parameters* which typically quantify the interactions, and *events* which cause discontinuous changes in model components given specified triggers. A network can also specify mathematical *function definitions* for use in other expressions.

`Network`s are constructed simply as

```
net = Network('example').
```

All the SBML components are added to a network using methods that begin with `add`, e.g.

```
net.add_parameter('kf', 0, is_optimizable=True).
```

This example shows one additional attribute SloppyCell assigns to parameters; if they are `constant`, they can additionally be declared `optimizable`. When

Networks are composed into a `Model`, the `optimizable` parameters are exposed at the `Model` level so they can be tweaked by optimization algorithms or when building an ensemble.

Often one wishes to study several slight modifications to a single `Network`. To that end, `Network`s have a `copy` method.

SloppyCell supports most of the current SBML specification, but there are some exceptions. First, SloppyCell does no processing of units. It is assumed that all numerical quantities in the `Network` have compatible units. Second, we don't support delay elements in math expressions. (Delays in event execution times are supported.) Finally, because we often take analytic derivatives of the equations, using discontinuous functions (e.g. `abs()` or `ceil()`) in the math for reactions or rules will cause integration problems. Discontinuities should be coded using events, and it is supported to use `piecewise` in the event assignments.

A.1.4 Dynamics

The `Dynamics` module contains methods to integrate a `Network`'s equations. The most basic functionality is `Dynamics.integrate` which simply integrates a model's differential equations forward in time and returns a `Trajectory` object containing the result.

Also quite useful is `Dynamics.integrate_sensitivity`, which returns a `Trajectory` object containing the *sensitivity* trajectories. These trajectories are $\partial y(t, \theta)/\partial \theta_i$, the derivatives of a given variable at a given time with respect to a given optimizable variable (indexed by e.g. `(y, theta_i)`). These trajectories are useful for optimizing parameters or experimental designs.

Finally, SloppyCell implements basic fixed-point finding in `Dynamics.dyn_var_fixed_point`.

A.1.5 Models

A `Model` object unites one or more `Networks` with the data contained in one or more `Experiments`:

```
m = Model([<list of expts>], [<list of nets>])
```

A `Model`'s primary task is to calculate the cost C for a set of parameters θ, defined as:

$$C(\theta) \equiv \frac{1}{2} \sum_i \left(\frac{B_i \, y_i(\theta) - d_i}{\sigma_i} \right)^2 + \text{priors}. \tag{A.1}$$

Here $y_i(\theta)$ is the model prediction, given parameters θ, corresponding to data point d_i, and σ_i is the uncertainty of that data point. The B_i are *scale factors* which account for data that are only relative, not absolute, measurements (e.g. Western blots). See Section A.1.6.1 for more on scale factors. The 'priors' term in the costs represents additional components of the cost, often designed to steer parameters in particular directions.

A.1.5.1 Priors

Often it is useful to add additional 'prior' terms to the cost. These may reflect previous direct measurements of a particular parameter, or restrict them to physically reasonable values. Prior terms are added using `m.add_residual(res)` where `res` is a `Residual` object. By far the most common form of additional residual we use is `Residuals.PriorInLog`. Such a residual adds a term to the cost of the form:

$$\frac{1}{2} \left(\frac{\log \theta_i - \log \theta_i^*}{\sigma_{\log \theta_i}} \right)^2 . \tag{A.2}$$

This acts to keep the logarithm of parameter θ_i from deviating much more than $\sigma_{\log \theta_i}$ from $\log \theta_i^*$.

A.1.6 Experiments

An `Experiment` object contains describes a set of data and how it should be compared with a set of `Networks`s.

A.1.6.1 Scale Factors

Each `Experiment` defines a set of measurements in which *all measurements of the same quantity share a scale factor*. Scale factors are important for many forms of biological data which do not give absolute measurements. For example, the intensity of a band in a Western blot is proportional to the concentration of that protein in the sample, but converting it to an absolute value may be very difficult to do reliably. The optimal conversion factor for comparison to a given set of `Network` results is, however, easy to calculate analytically and need not be included as an extra fitting parameter. For historical reasons, by default SloppyCell assumes that *all data involve scale factors that should be optimized*. If you know the absolute scale of your data, use `expt.set_fixed_sf` to specify *fixed* scale factors. Or, if you know that two variables should share a scale factor which needs to be optimized, use `expt.set_shared_sf`.

Because SloppyCell handles the scale factors implicitly, when building an ensemble we must account for their fluctuations by using a *free energy* rather than the previously mentioned `cost`. This free energy depends on prior assumptions about how the scale factors are distributed, and these priors can be changed using `expt.set_sf_priors`. For more on possible effects and subtleties of choosing these priors, see Section 6.2.

```
expt.set_data({'net1':{'X': {2.0: (2.85, 0.29),
                             5.0: (4.9, 0.49),
                             },
                       'Y': {2.0: (1.25, 0.13),
                             5.0: (1.12, 0.13),
                             },
                       },
               'net2':{'X': {2.0: (6.7, 0.3),
                             4.2: (9.8, 0.2),
                             },
                       }
               )
```

Listing A.1: Shown is an example of SloppyCell's data format. This data set contains data on the species with `ids` `'X'` and 'Y' taken under two conditions corresponding to the `Networks` with `ids` `'net1'` and `'net2'`. Importantly, the two conditions by default *share* a floating scale factor for `'X'`.

A.1.6.2 Data Format

SloppyCell's data format is simply a set of nested Python dictionaries or `KeyedLists`. This can be unwieldy to write out by hand, but it provides flexibility for future addition of more complex forms of data, and it can easily be generated from other tables by simple scripts. The first level of nesting in the experimental data is keyed by the `id` of the `Network` whose results the data should be compared with, and the next level is keyed by the variable the data refers to. The final level is a dictionary or `KeyedList` of data-points, mapping times to tuples of `(<value>, <one-sigma uncertainty>)`.

The data format is best illustrated by example; see Listing A.1. This `Experiment` contains data on two variables `'X'` and `'Y'` in two conditions, corresponding to the `Networks` `'net1'` and `'net2'`. Note that, because they are in the same experiment, the two data sets on `'X'` will be fit using the same scale factor. This might be appropriate, for example, if the data came from Western blots using very similar numbers of cells and the same antibody.

A.1.7 Optimization

After the `Model` has been created, it is very common to want to optimize the parameters θ to minimize the cost and thus best fit the data. SloppyCell includes several optimization routines in the `Optimization` module. These include wrappers around SciPy's Nelder-Mead and conjugate gradient routines and SloppyCell's own implementation of Levenberg-Marquardt. The conjugate gradient and Levenberg-Marquardt routines use analytical derivatives calculated via sensitivity integration, and all the routines have versions for both logarithmic and bare parameters.

All the currently implemented methods do only local optimization, but minimizing the cost using any Python-based minimization algorithm is straightforward. To further explore parameter space, consider building a parameter ensemble (Section A.1.9).

A.1.8 Cost and Residual Derivatives

In both optimization and ensemble building, various derivatives of the cost function are very useful. These are all available using methods of the `Model` object, and in each case there are versions for logarithmic and bare parameters. Using sensitivity integration, we can calculate all first derivatives semi-analytically, without reliance on finite-difference derivatives and the resulting loss of precision.

Most basic derivative is the gradient of the cost which is useful for many deterministic optimization algorithms.

A slightly more complicated object is the Jacobian J, which is the derivative matrix of residuals versus parameters: $J_{i,j} \equiv dr_i/d\theta_j$. (The cost we consider (Equation A.1) is a sum of squared residuals: $C(\theta) = \frac{1}{2}\sum_i r_i(\theta)^2$). The Jacobian is useful for understanding which parameters impact which features of the fit and

for clustering parameters into redundant sets [58].

Finally, the Hessian H is the second derivative matrix of the cost: $H_{i,j} \equiv dC(\theta)/d\theta_i d\theta_j$. If calculated at a minimum of the cost, the Hessian describes the local shape of the cost basin. This makes it useful for importance sampling when building an ensemble (Section A.1.9). Note, however, that Hessian calculation relies on finite-difference derivatives, which can be difficult to calculate reliably. For our least-squares cost functions, a very useful approximation to the Hessian is $J^{\mathsf{T}}J$, which can be calculated as:

```
j = m.Jacobian_log_params_sens(log(params))
jtj = dot(transpose(j), j)
```

The approximation becomes exact when the model fits the data perfectly.

A.1.9 Ensembles

To explore the full nonlinear space of parameters that are statistically consistent with the model and the data, we build a Bayesian ensemble where the relative likelihood of any parameter set θ is:

$$P(\theta) \propto \exp\left(-G(\theta, T)/T\right). \tag{A.3}$$

Here $G(\theta, T)$ is the *free energy*, which is the cost plus a possible contribution due to fluctuations in scale factors. The temperature T controls how much we're willing to let the free energy deviate from the optimum. For strict statistical correctness, it should be one, but there are situations in which it is useful to adjust user larger values [10].

The ensemble is built using `Ensembles.ensemble_log_params`, using an importance-sampled Markov-Chain Monte-Carlo algorithm [84]. This algorithm builds the ensemble by taking a random walk through parameter space which, eventually, will converge to the correct probability distribution.

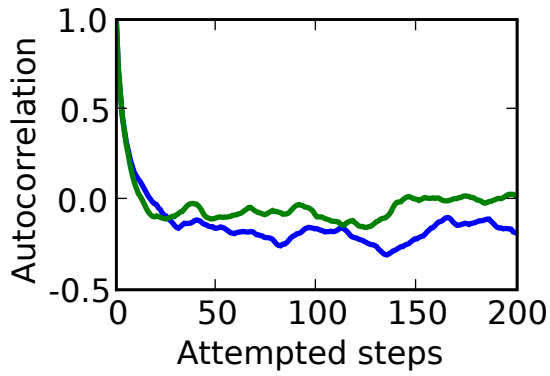


Figure A.1: Shown are autocorrelation functions functions for the cost (blue) and the logarithm of a particular parameter (green) in an ensemble built for the small JAK-STAT example model (Section A.2). The correlation time for both is about 25 steps, suggesting that parameter sets 25 steps apart are statistically independent.

A.1.9.1 Assessing and Speeding Convergence

'Eventually' is a key word in describing ensemble convergence. Because we are taking a walk through parameter space, subsequent members of the ensemble are highly correlated. Generating a thoroughly converged ensemble that independently samples the distribution of parameters many times can be quite computationally intensive for large models.

There are several ways to assess convergence. Most thorough (but computationally expensive) is to start several ensembles from very different initial parameters and see that they give identical answers for your predictions.

Given a single ensemble, one can check for convergence using the autocorrelation function of the cost and logarithms of the parameter values using `Ensembles.autocorrelation`. Figure A.1 shows example autocorrelation functions for a small model. The number of independent samples in an ensemble is approximately the length of the ensemble divided by the *longest* correlation time of any parameter in the ensemble. Scaling arguments suggest that, for the default ensemble parameters, the number of steps in one autocorrelation time is at least

the square of the number of parameters (Section 3.5).

If the correlation time for your ensemble is much longer than the square of the number of parameters, your cost basin is probably substantially curved. For advice on how to deal with this, see Section 6.3.

A.1.9.2 Predicting from an Ensemble

Once your ensemble has converged, you're ready to make predictions. As a first step, you'll probably want to *prune* your ensemble. As mentioned previously, consecutive members of the ensemble are not independent, and it is independent samples that matter for predictions. Once you've estimated the longest correlation time (`corr_steps`), the ensemble can be pruned simply by taking one member per correlation time: `ens_pruned = ens[::corr_steps]`.

To calculate uncertainties for any quantity over the ensemble, simply calculate its value for each member of your pruned ensemble and note the spread in values. SloppyCell includes a few functions to make this easier in some common cases. `Ensembles.ensemble_trajs` will integrate a `Network` for a fixed set of times over all members of an ensemble. `Ensembles.traj_ensemble_quantiles` will calculate quantiles over those integrations, to show, for example, what the 95% confidence bounds are on the trajectory prediction. Similarly, `Ensembles.traj_ensemble_stats` will return trajectories containing the mean and standard deviation of each integration quantity over an ensemble.

A.1.10 Plotting

SloppyCell's plotting functionality is built upon matplotlib [93], also known as `pylab`; a nice tutorial is available at http://matplotlib.sourceforge.net/tutorial.html. SloppyCell's `Plotting` module adds several routines that are

convenient for analyzing fits, ensembles, and Hessians. For example, Figure A.3 shows a plot of the best fit for our example model (Section A.2) with no additional tweaking.

A.1.11 KeyedLists

Numerical algorithms are generally designed to work with arrays of values, while users don't want to remember which parameter was number 97. To solve both issues, SloppyCell uses a custom data type called a `KeyedList` which has properties of both Python lists and dictionaries. Like a normal list, values can be accesses using `[<index>]` notation. Additionally, each entry is associated with a key, so that values can be added and accessed using `get` and `set` methods like a Python dictionary.

A.1.12 Input and Output

SloppyCell's `IO` module includes several functions to facilitate import and export of useful representations of `Network`s.

`IO.from_SBML_file` and `IO.to_SBML_file` allow importing and exporting of SBML files. (Note that SloppyCell will not preserve annotations of an SBML network that has been imported.) `IO.net_DOT_file` generates `dot` files which can be fed to Graphviz (http://www.graphviz.org/) to generate basic network diagrams as seen in Figure A.2. Finally, `IO.eqns_TeX_file` will export a `tex` file that can be used with LaTeXto generate nicely-formatted equations. These are useful both for inclusion in publications and for debugging models.

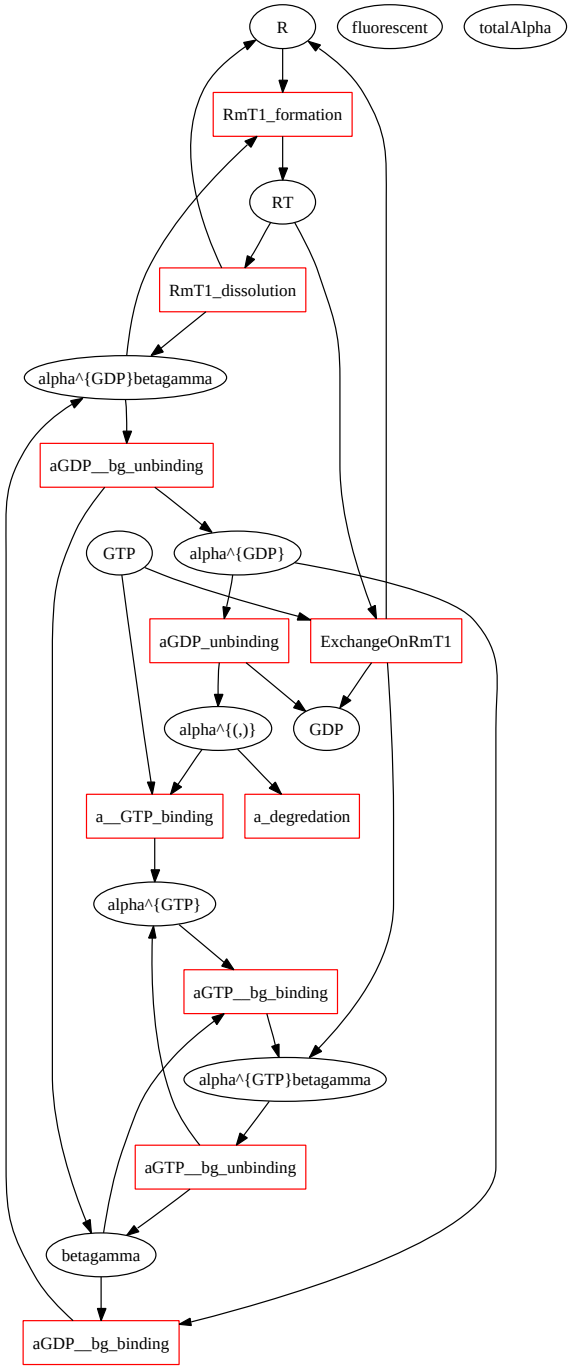


Figure A.2: Shown is a SloppyCell-generated network diagram for the G protein model discussed in Chapter 3. Black ovals denote species and red boxes denote reactions between them.

A.1.13 Miscellaneous Utilities

To conveniently save results for future analysis, you can use `Utility.save` and `Utility.load`. Note that these rely on Python's binary 'pickle' format, so there is a slight danger that upgrades to Python or SloppyCell will render them unreadable. For more robust but less space-efficient saving, dump results to text files using, for example, `scipy.io.write_array`.

A.1.14 Parallelization

SloppyCell implements a relatively simple parallelization paradigm [88] over MPI, using the PyPar (http://sourceforge.net/projects/pypar/) interface. Two major tasks have been parallelized. (1) If evaluating the cost of a `Model` involves integrating several `Networks`, those integrations can be spread across several processors. (2) In `Dynamics.integrate_sensitivity`, the integrations for different parameters can be distributed across processors. Parallel execution is enabled by a simple import:

```
import SloppyCell.ReactionNetworks.RunInParallel as Par.
```

Anything in the script beyond this import will be run only by the master, who can send commands to the slaves. The slaves can be directly controlled using using methods in Par. For example, `Par.statement_to_all_workers("print 'I am processor', my_rank")` will cause each processor to print a message giving its rank.

A.2 JAK-STAT Example

Here we consider an example application using the JAK-STAT model of Swameye et al. [85], which was also used as an example by the SBML Parameter Estimation

Toolkit (SBML-PET) [81].

Using the script shown in Listing A.2, we'll fit the model, build an ensemble, and then estimate some prediction uncertainties, reproducing most of Figure 4.1. (This script is also found in the `Examples/JAK-STAT` directory of the SloppyCell source distribution.)

Lines 1 through 3 `import` code from the Python packages we'll use this session: matplotlib, SciPy, and SloppyCell.

On line 5 we load the model network from the SBML file. (Note that this file has been modified from the one included with SBML-PET to fix some bugs in it and to define the species `'data1'` and `'data2'` which correspond to the measurements we'll fit.) Given the space constraints for listings here, the `Experiment` object is defined in another file, which we `import` on line 8. We use that `Experiment`, along with our previously created `Network`, to create the `Model` object on the following line.

Our initial parameters are defined as a `KeyedList` starting on line 11. We could have specified them as a simple `list`, without the names, but we find things are much clearer when the names are visible as well.

A couple of priors need to be defined to keep model parameters from drifting too far. The prior on `'r3'` (line 14) constrains it (within 95% probability) to lie between 10^{-4} and 10^{4}, while the prior on `'tao'` constrains it to lie between 1 and 16.

The initial cost is about 560, which is very high given that we only have 32 data points. Thus we run several iterations of Levenberg-Marquardt on line 20. (We limit the number of iterations here merely for expediency. This number gets us very close to the actual minimum.) The final cost should be about 18.2. For a perfectly fitting model, we expect a cost of 1/2 the number of data points, so this

```
from pylab import *                                                    #
from scipy import *
from SloppyCell.ReactionNetworks import *                              #

net = IO.from_SBML_file('JAK-STAT_SC.xml', 'net1')                     #
net.set_var_ic('v1', 'v1_0') # Won't be needed once initial assignments work.  #

import JAK_expt                                                        #
m = Model([JAK_expt.expt], [net])

params = KeyedList([('r1', 0.5), ('r3', 2), ('tao', 6.0),              #
                    ('r4_0', 1.35), ('v1_0', 1.19)])

res = Residuals.PriorInLog('r3_prior', 'r3', 0, sqrt(log(1e4)))        #
m.AddResidual(res)
res = Residuals.PriorInLog('tao_prior', 'tao', log(4), sqrt(log(4)))
m.AddResidual(res)                                                     #

print 'Initial cost:', m.cost(params)                                  #
params = Optimization.lm_log_params(m, params, maxiter=20, disp=False) #
print 'Optimized cost:', m.cost(params)
print 'Optimized parameters:', params

# Plot our optimal fit.
figure()
Plotting.plot_model_results(m)                                         #

j = m.jacobian_log_params_sens(log(params))                            #
jtj = dot(transpose(j), j)                                             #

print 'Beginning ensemble calculation.'
ens, gs, r = Ensembles.ensemble_log_params(m, asarray(params), jtj,    #
                                           steps=7500)
print 'Finished ensemble calculation.'

pruned_ens = asarray(ens[::25])                                        #

figure()
hist(log(pruned_ens[:,1]), normed=True)                                #

times = linspace(0, 65, 100)
traj_set = Ensembles.ensemble_trajs(net, times, pruned_ens)            #
lower, upper = Ensembles.traj_ensemble_quantiles(traj_set, (0.025, 0.975))

figure()
plot(times, lower.get_var_traj('frac_v3'), 'g')
plot(times, upper.get_var_traj('frac_v3'), 'g')
plot(times, lower.get_var_traj('frac_v4'), 'b')
plot(times, upper.get_var_traj('frac_v4'), 'b')

show()
```

Listing A.2: This script reproduces some of the results from Chapter 4. For detailed comments see Section A.2.

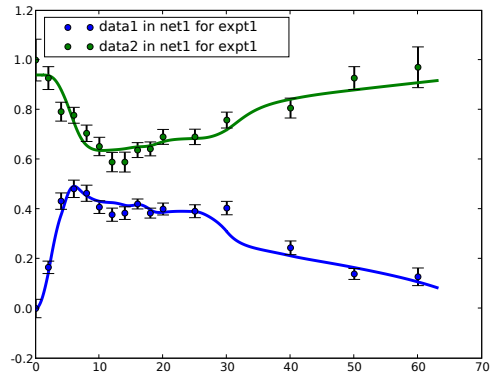


Figure A.3: Plotted is the optimal fit for our example model, generated using `Plotting.plot_model_results(m)` in Listing A.2.

cost indicates a good fit. On line 26 we generate Figure A.3, which compares our best fit with the data.

Our group's philosophy is, however, not to trust solely in the best fit, so we'd like to build an ensemble of parameters. Before we can build an ensemble, we need to build a matrix to guide the sampling. Here we use the $J^{\mathsf{T}}J$ approximation to the Hessian, which we calculate on lines 28 and 29. (As an aside, the eigenvalues and eigenvectors of this $J^{\mathsf{T}}J$ are 'sloppy', as with the models discussed in Chapter 2.)

On line 32 we build a parameter ensemble. We only build a 7500 step ensemble because the model is quite small and well-constrained; with 5 parameters the correlation time should be only about 25 steps. Also, we cast the `params` `KeyedList` to an array in the call; this makes our returned `ens` be composed of arrays rather than `KeyedLists`, which is more memory efficient. Calculating a 7500 member ensemble for this model takes approximately 15 minutes on a modern PC. On line 36 we prune the ensemble; using slicing notation to take every 25th element (25 being the correlation time). We also convert to an array, so that we can use the more powerful slice syntax of arrays.

What does the ensemble actually look like? On line 39, we use matplotlib's

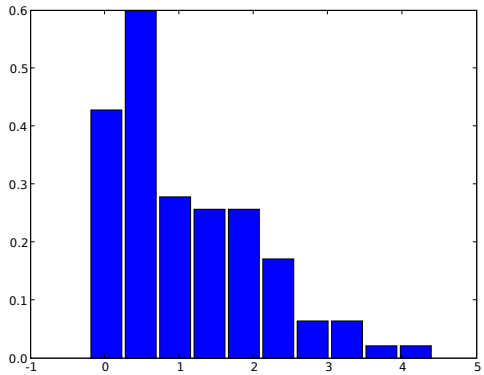


Figure A.4: Shown is a histogram of log `'r3'` for the JAK-STAT ensemble. Note that the value of `'r3'` is bounded from below at about 10^0, but the upper end is only bounded by the prior.

`hist` function to build a histogram of the logarithm of `'r3'` over the ensemble. The result is shown in figure A.4. Note that the upper bound on `'r3'` is set by the prior we added, while the lower bound is constrained by fitting the data.

Once we have our ensemble, we can make some predictions. On line 42 we calculate trajectories for `net` over our pruned ensemble, and on the following line we generate `lower` and `upper` trajectories that bound the central 95% of the values for each variable. We then plot these bounds for the variables `'frac_v3'` and `'frac_v4'`, which have been defined to be `'2*v3/v1'` and `'2*v4/v1'` respectively. These are the fractions of total STAT that are involved in cytoplasmic and nuclear dimers. Figure A.5 shows the resulting figure. Note the relatively large uncertainty of the cytoplasmic dimers (green), which gets very close to zero.

Finally, we end our script with a call to `show()`. This `pylab` command ensures that the plots pop up. It may be unnecessary if you're running the script from within an IPython session started with the `-pylab` command-line option.

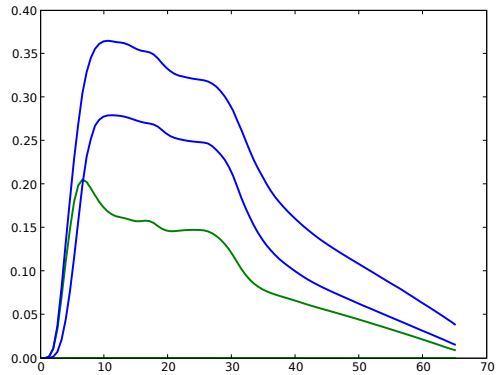


Figure A.5: In green is the 95% uncertainty bound on cytoplasmic dimers, while blue is the 95% bound on nuclear dimers, given the model and data that have been fit.

A.2.1 Other Examples

The JAK-STAT example presented here covers the most basic of SloppyCell's functionality. More extensive examples can be found in the `Examples` directory of the source distribution.

A.3 Installation

A.3.1 Required Dependencies

Python version 2.4 or higher

Available from http://python.org/.

NumPy version 1.0.2 or higher

Available from http://scipy.org/

SciPy version 0.5.2 or higher

Available from http://scipy.org/

matplotlib version 0.90.0 or higher

Available from http://matplotlib.sourceforge.net/

A.3.2 Optional Dependencies

libSBML version 2.3.4

Available from http://www.sbml.org/software/libsbml/.

If libSBML is installed, SloppyCell will be able read and write files encoded in the Systems Biology Markup Language [55]. See section A.1.12.

IPython Available from http://ipython.scipy.org/.

IPython is a dramatic improvement over the standard python shell and we highly recommended it for interactive work.

C compiler For speed, SloppyCell by default generates C versions of the network equations, and this requires a C compiler. If a C compiler is not installed, SloppyCell will run with Python versions of the network equations, which may be up to a factor of 30 slower. This capability has been tested only with the compiler `gcc`, but it should work with others.

Pypar Available from http://sourceforge.net/projects/pypar/.

With Pypar installed, SloppyCell can run several common calculations in parallel over MPI. See section A.1.14 (As of August 21, 2007 the alpha version of Pypar 2.0 is required for compatibility with NumPy.)

A.3.3 On Linux

Installation on a Linux system should be straightforward, as Linux machines are generally well-equipped with the proper compilers.

1. Install the required dependencies

2. Unpack the tarball containing the SloppyCell code. (`tar xzf SloppyCell-XXX.tar.gz`)

3. Descend into the SloppyCell-XXX directory. (`cd SloppyCell-XXX`)

4. Run the installation script to build SloppyCell. (`python setup.py build`). Depending on your system, this may fail due to difficulties with Fortran compilation. The available Fortran compilers can be found with `python setup.py config_fc --help_fcompiler`, and if you contact us we can help with this step.

5. Run the installation script to install. (`python setup.py install`).

   If you prefer to use SloppyCell from the current directory rather than installing it elsewhere, use the `--install-lib` option, `python setup.py install --install-lib=..` . You will have to adjust your `PYTHONPATH` environment variable to include the directory the SloppyCell directory is contained within. This installation option is particularly convenient for staying up-to-date with the CVS repository.

A.3.4 OS X

Note that installation of libsbml with python bindings on Intel macs currently requires some work-arounds. Hopefully these will be resolved soon by the libsbml maintainers. If you're having difficulties, please contact us.

A.3.4.1 Pre-built binary

Binary `mpkgs` are available from the Sourceforge download site. These have been built against the framework builds of Python available from http://www.python.org on OS X 10.4 ('Tiger').

A.3.4.2 From source code

Fortran compilers for OS X are available at http://hpc.sf.net. Once you've installed such a compiler, follow the instructions for installing on Linux.

A.3.5 Windows

Installing `gcc` on Windows requires some effort, but is highly suggested as it will dramatically speed up SloppyCell.

In our experience, the easiest way to get `gcc` is to install `MinGW`. To install it, run `MinGW-5.1.3.exe` from http://sourceforge.net/projects/mingw/. Tell the installer you want `g77` and `g++` in addition to the defaults.

Unfortunately, there is an issue with MinGW-5.1.3 that will cause compilation problems. To fix them, you will need to find `libgcc.a` and run the following command: `ar dvs libgcc.a __main.o` [154]. On our test system `libgcc.a` was in `c:\mingw\lib\gcc\mingw32\3.4.2`.

A.3.5.1 Pre-built binary

Pre-built binaries are available for Python 2.4 and Python 2.5.

A.3.5.2 From source code

Given that you installed `g77` along during the MinGW install, you should be able to follow the Linux install instructions.

A.3.6 Testing the Installation

The SloppyCell source code includes a large package of test routines. To access them: download and unpack `SloppyCell-XXX.tar.gz` or `SloppyCell-XXX.zip`,

descend into the SloppyCell-XXX directory (`cd SloppyCell-XXX`), and run `cd Testing; python test.py`. This will run an extensive series of tests, both with and without C-compilation.

A.4 Troubleshooting

A.4.1 Failing Integrations

Optimization or ensemble construction may explore regions of parameter space for which the model equations become very difficult to integrate, leading to many `daeint` exceptions being raised. One possible solution is to check the 'typical' values that are being assumed for each variable, accessible via `net.get_var_typical_vals()`. These are used to help set the absolute tolerances of the integrations, and if they are very different from the values the variable actually attains, the inconsistency can cause problems. The solution is then to set them to more reasonable values using `net.set_var_typical_val(<variable id>, <value>)`.

APPENDIX B

SLOPPYCELL DEVELOPER DOCUMENTATION

The primary reference for developers is the code itself. This document serves to outline design decisions or particularly tricky parts of the code.

B.1 Test Suite

First, the importance of a good test suite cannot be over-emphasized. When large changes are made to the codebase (for example, incorporating C right-hand-side functions) comprehensive tests make it straightforward to crush many bugs before they effect end users.

SloppyCell's test suite is based on the Python `unittest` framework. The idea of a unit-test is that each test is small and self-contained, testing only a single aspect of the code. This makes it easier to track failures to their source. The `Testing/test.py` file agglomerates all the tests and runs them both with and without C-compilation. Developers should run the test suite with verbose error messages (`python test.py -v`). Make sure your CVS commits don't break the tests! *Every* CVS commit should be preceded by a run of the test suite.

The test suite is, unfortunately, incomplete. We didn't begin automated testing until much of the code was written, particularly at the `Model` level. Nevertheless, all new features should have unit-tests to ensure correctness, and adding new tests to old features is a very valuable contribution.

`test_Misc.py` is a home for tests of bugs that aren't related to specific features. Ideally, if a bug is squashed, a test should be added in this file if it doesn't fit elsewhere.

B.2 Logging

Rather than littering the code with `print` statements, the Python `logging` module should be used, as it allows for much finer control of messaging.

Every module begins by defining a `logger` object which is used for output from that module. Debugging messages should be logged by `logger.debug("My message")`. These will become visible when that module's logger is set to `DEBUG` mode:

```
module.logger.setLevel(module.logging.DEBUG).
```

The output of all loggers can be dumped to a file by starting any SloppyCell script as:

```
python my_scipy.py --debugSC=output_file.log.
```

Logging messages sent through `logger.warn` or `logger.critical` will be visible to the user. They should be used sparingly.

If a set of `logger.debug` statements is useful for tracking down bugs in general, it's fine to leave them in the final commit, as long as they aren't in an inner loop where they will impact performance.

B.3 Integrator

For integration, SloppyCell uses the differential-algebraic equation solver DASKR [94, 95]. DASKR is written in Fortran while the vast majority of SloppyCell is in Python. To interface with between the two languages, we use f2py, which has recently become part of NumPy. The Python interface to DASKR is specified in `ddaskr.pyf`.

One subtlety in the interface is the use of the variable `ipar`. When using DASKR from Fortran, `ipar` would be used to pass extra integer arguments to user-supplied functions. In our wrapper, we use `ipar` to store (1) the dimension of

the system we're solving and (2) the dimension of `rpar`, the array used for passing extra double precision arguments to user-supplied functions. These entries in `ipar` must be set to the proper values by the user of the Python interface. This usage of `ipar` is necessary because the interface code f2py builds between Python and Fortran needs to have the dimension of all arrays specified.

B.4 ExprManip

Many important features in SloppyCell are enabled by the included `ExprManip` package, which includes a slew of methods for manipulating strings representing Python math expressions. These manipulations including extracting all the variables, making substitutions, and taking analytic derivatives. All of these operations begin by building an Abstract Syntax Tree (AST) using the `compiler` module. This tree (a list of lists of lists...) is then manipulated, often recursively.

One caveat with these methods is that they are relatively slow. Perhaps surprisingly, the bottleneck is not in recursively descending the AST, but rather in generating the AST itself. Given that the AST is generated by the standard `compiler` module, it may be difficult to speed this up. Thus it is important to minimize the number of redundant ASTs that are built. A first round of such optimization cut the time to compile large networks by a factor of five to ten.

B.5 Dynamic Functions

To integrate or otherwise analyze a specific `Network`, SloppyCell dynamically generates a number of functions. When possible, C versions of these functions are built and compiled, with Python versions used as a fall-back.

B.5.1 Compilation

The Python and C codes for all the dynamic functions are generated upon a call to `net.compile`. To ensure that the dynamic functions are up-to-date with the structure of the `Network`, `compile` is called before every integration. Generating the code for the dynamic functions is, however, quite slow, so `compile` first checks whether the structure of the `Network` has changed (as defined by `net._get_structure()`) before generating new code.

If `compile` decides it is necessary to regenerate the dynamic functions, all the methods in `Network._dynamic_structure_methods` are called in order. These methods should fill the `KeyedList` `net._dynamic_funcs_python` with the Python code for all the desired dynamic functions. Similarly, these methods can fill `self._prototypes_c` and `self._dynamic_funcs_c` with the function prototypes and code for all the functions that should be compiled in C. Thus to add a new dynamic function to the `Network` class, one simply writes a method in `Network` that will add entries to the appropriate `_dyamic_funcs` lists and adds that method to `Network._dynamic_structure_methods`. If the added function has a C version, the file `f2py_signatures.pyf` will need to be updated with an appropriate interface signature. (Note, because `Network`s are often modified solely by adding or removing events, generating the dynamic functions relating to events is handled separately but similarly.)

B.5.2 Execution

The final step in `net.compile` is to call `net.exec_dynamic_functions`, which will execute all the dynamic function code into callable functions.

First `net.exec_dynamic_functions` checks a cache to see whether it really needs to re-create all the functions. If it does, it first constructs Python functions

from all the code in `net._dynamic_funcs_python`. To construct the functions, we `exec` the code for each function and attach the resulting object to the `Network` `self`. Note that these are functions, not methods; they don't take an implicit `self` argument and can't access attributes of the `Network` they are attached to. This is primarily for consistency with the C versions, which cannot be sensibly made into methods.

One additional wrinkle exists for large models. It turns out that Python has an in-built limit to the complexity of a module it can `import` (or, equivalently, string it can `exec`). The resulting error is `SystemError: com_backpatch: offset too large`. Including logical control statements (`if`, `for`, `while`) in our dynamic functions is thus dangerous [155]. This is why the current versions generates individual functions for $\partial y/\partial p$ rather than using a large `if` statement as in previous versions.

To construct C functions `net.exec_dynamic_functions` writes `.c` and `.pyf` files containing the code and interface for all the C dynamic functions. To help ensure uniqueness the base filenames are assigned based on current system time and the MPI rank of the current node. These files are compiled into an python module by spawning an external f2py process. The resulting module is imported and stored in the cache, and the dynamic functions in that module are assigned to the appropriate attributes of the `Network`, overwriting the Python versions. (Generating the module with a unique name circumvents the fact that C extension modules cannot be reloaded. If a network is changed many times during program execution, however, the import of all these modules may cause excessive memory usage, as the garbage collector cannot free unused imported C modules.)

In general the C code for a dynamic function is a straightforward translation of the Python code. One important difference is that functions that are passed

to DASKR should take in the full argument list expected by DASKR, even if the f2py wrapper hides some of them. By passing a `._cpointer` from the function to DASKR we can then get direct C to Fortran communication, avoiding any Python-induced overhead. The other subtlety is that ostensibly two-dimensional arrays are passed in from Fortran functions as flat one-dimensional arrays, so indexing is more complicated. (One can cast a one-dimensional array to a two-dimensional array in C via `double (*arr2d)[N] = (double(*)[N])(&arr1d);`. In testing this seemed to cause problems when interfacing with DASKR.)

B.6 Sensitivity Integration

Much of our research revolves around how changes in parameter values affect the dynamics $y(t;p)$ of a network, thus we are often interested in the *sensitivities* $\frac{dy(t;p)}{dp}$ of those dynamics. Such sensitivities can be obtained via finite-difference derivatives as

$$\frac{dy(t;p)}{dp} = \frac{y(t;p+\Delta p) - y(t;p)}{\Delta p}. \tag{B.1}$$

This procedure is, however, not very well-behaved numerically. We can do much better using SloppyCell's ability to take analytical derivatives of Python math expressions.

In a normal integration, we're evaluating:

$$y(t;p) = \int_0^t \frac{dy}{dt'}\, dt'. \tag{B.2}$$

If we take d/dp of this equation, we obtain

$$dy(t;p)/dp = \int_0^t \left[\frac{\partial}{\partial p}\frac{dy}{dt'} + \sum_{y'} \frac{\partial}{\partial y'}\frac{dy}{dt'}\frac{dy'}{dp} \right] dt'. \tag{B.3}$$

Essential to this procedure is the fact that analytic Python math expressions for $\frac{\partial}{\partial p}\frac{dy}{dt'}$ and $\frac{\partial}{\partial y'}\frac{dy}{dt'}$ can be obtained using the analytic differentiation capabilities of

the `ExprManip` module (Section B.4). This set of equations must be integrated simultaneously with normal right-hand-side (Equation B.2), so our system now has twice as many equations. This does slow down the integration somewhat, but our experience suggests that calculating sensitivities this way is not much slower than calculating them via finite differences and is much better behaved.

In SloppyCell, the right-hand-side function for the sensitivity integration for a `Network` object is `net.sens_rhs`. The optimizable variable to return derivatives with respect to is specified by the last entry in the `constants` argument to `sens_rhs`.

B.6.1 Handling Events

Perhaps the trickiest part of the sensitivity integration is dealing correctly with the SBML event model. The SBML event model is relatively complex and perhaps not intuitive. An event *fires* when the function defining its triggering function T transitions from False to True. At that *firing time* t_f new values are calculated for all variables with event assignments. The effects of the event may *delayed* by some time t_d which may be a function D of the variables at the firing time. The event thus *executes* at a time $t_e = t_f + t_d$, and the values calculated when the event fired are assigned to the appropriate variables.

B.6.1.1 Time Sensitivities

First we calculate the derivative of the event firing time with respect to p. The event fires when $\mathrm{T}(y(t_f, p), p, t_f) = 0$, and taking the derivative yields:

$$\frac{d}{dp}\mathrm{T}(y(t_f,p),p,t_f) = \frac{\partial \mathrm{T}}{\partial p} + \sum_y \frac{\partial \mathrm{T}}{\partial y}\frac{dy}{dp} + \frac{\partial \mathrm{T}}{\partial t}\frac{dt_f}{dp} + \sum_y \frac{\partial \mathrm{T}}{\partial y}\frac{dy}{dt}\frac{dt_f}{dp} = 0. \quad \text{(B.4)}$$

Solving for $\frac{dt_f}{dp}$ we obtain

$$\frac{dt_f}{dp} = \frac{\frac{\partial \mathrm{T}}{\partial p} + \sum_y \frac{\partial \mathrm{T}}{\partial y}\frac{dy}{dp}}{\frac{\partial \mathrm{T}}{dt} + \sum_y \frac{\partial \mathrm{T}}{\partial y}\frac{dy}{dt}}. \tag{B.5}$$

All quantities in this derivative are, of course, evaluated at the time the event fires. One subtlety with $\frac{dt_f}{dp}$ is that events may be *chained*; the execution of one event may cause the firing of another event. In that case $\frac{dt_f}{dp}$ of the fired event is equal to $\frac{dt_e}{dp}$ of the event whose execution caused the current event to fire.

Next we need the derivative of the delay time t_d which may be calculated by a function $\mathrm{D}(y(t_f), p, t_f)$. This is straightforward to calculate as:

$$\frac{dt_d}{dp} = \frac{\partial \mathrm{D}}{\partial p} + \sum_y \frac{\partial \mathrm{D}}{\partial y}\frac{dy}{dp} + \frac{\partial \mathrm{D}}{\partial t}\frac{dt_f}{dp} + \sum_y \frac{\partial \mathrm{D}}{\partial y}\frac{dy}{dt}\frac{dt_f}{dp}, \tag{B.6}$$

where again all variables are evaluated at the firing time.

Finally, the sensitivity of the event execution time is just

$$\frac{dt_e}{dp} = \frac{dt_f}{dp} + \frac{dt_d}{dp}. \tag{B.7}$$

B.6.1.2 Variable Sensitivities

Now let us calculate the sensitivities of variables after event execution. First consider a variable y whose value is not changed by the event. Note that $\frac{dy}{dt}$ may be different before and after the event executes because of changes to other variables. Then the perturbation to its sensitivity $\frac{dy}{dp}$ is given by

$$\left.\frac{dy}{dp}\right|_{>t_e} = \left.\frac{dy}{dp}\right|_{<t_e} + \left.\frac{dy}{dt}\right|_{<t_e}\frac{dt_e}{dp} + \left.\frac{dy}{dt}\right|_{>t_e}\frac{dt_e}{dp}, \tag{B.8}$$

where $|_{<t_e}$ denotes values prior to event execution and $|_{>t_e}$ denotes values after event execution. The additional terms involving $\frac{dy}{dt}$ can be thought of as changes in y that do or do not happen because of the shift execution time.

Now let y be a variable whose value is changed by the event, as determined by the function A. The sensitivity of y after event execution is:

$$\left.\frac{dy}{dp}\right|_{>t_e} = \left.\frac{d\,\mathrm{A}}{dp}\right|_{t_f} - \left.\frac{dy}{dt}\right|_{>t_e} \frac{dt_e}{dp}, \tag{B.9}$$

where $|_{t_f}$ is a reminder of the fact that A is calculated at the firing time, so only the variable values at that time can matter for that term. The sensitivity of the assigned value is

$$\frac{d\,\mathrm{A}}{dp} = \frac{\partial\,\mathrm{A}}{\partial p} + \sum_y \frac{\partial\,\mathrm{A}}{\partial y}\frac{dy}{dp} + \frac{\partial\,\mathrm{A}}{\partial t}\frac{dt_f}{dp} + \sum_y \frac{\partial\,\mathrm{A}}{\partial y}\frac{dy}{dt}\frac{dt_f}{dp}, \tag{B.10}$$

where all variables are evaluated at the time the event fires.

B.6.1.3 Implementation

To calculate all the sensitivities, we need $\left.\frac{dy}{dt}\right|_{t_f}$, $\left.\frac{dy}{dt}\right|_{<t_e}$, $\left.\frac{dy}{dt}\right|_{>t_e}$, $\left.\frac{dy}{dp}\right|_{t_f}$, and $\left.\frac{dy}{dp}\right|_{<t_e}$. In particular, note that we need $\left.\frac{dy}{dt}\right|_{>t_e}$ which is a quantity only available *after* the event executes. To deal with this, each sensitivity integration begins with a normal integration of the network. The resulting trajectory stores a list of `event_info` objects under `traj.events_occurred`, and the relevant quantities are available as attributes of these objects. For example, if `e` is an `event_info` object, then `e.yp_pre_exec` holds $\left.\frac{dy}{dt}\right|_{<t_e}$.

During sensitivity integration a copy of normal trajectory's `events_occurred` list is made, and the `event_info` objects are updated with the necessary sensitivity information (e.g. $\left.\frac{dy}{dp}\right|_{t_f}$) as the network is integrated. All computations related to events and sensitivities outlined above are performed in `Network_mod.executeEventAndUpdateSens`. Chained events are handled by storing a reference to the `event_info` object prior to the current event in the chain.

B.6.2 Adjoint Method

The above sensitivity analysis requires solving $2N_e$ equations for each parameter, where N_e is the number of equations in a normal integration of the model. For calculating a Jacobian this many integrations is probably be unavoidable. In optimization, however, we're often interested in the gradient of a single function of the model variables, the cost. So-called 'adjoint' sensitivity methods are designed for just this case, where we're interested in the derivative of one or a few quantities with respect to many variables [156, 157]. Essentially, an adjoint calculation involves two integrations, one forward and one backward, of an augmented systems of equations. Importantly, the size of this augmented system does not depend on the number of parameters one is considering.

An early version of SloppyCell included adjoint calculation of the gradient of the cost. Performance was somewhat disappointing; the method was not faster than calculating the forward sensitivities as above. This was not, however, a failure of the method itself, but rather our implementation. As mentioned above, the adjoint method requires an integration backwards in time which must refer to values calculated on the forward integration. To access those values we used the SciPy's spline interpolation routines, and it was calls to these routines that killed performance. An implementation that had direct access to the forward integration's approximation to the trajectory could be much faster.

Additionally, we did not work out how to propagate the adjoint system across events. As seen above, this can be quite complicated even in forward sensitivity analysis.

B.7 Parallel Execution

For communication between nodes in a parallel job, SloppyCell uses `PyPar`, a relatively Pythonic wrapper for MPI. To minimize code complexity (particularly in exception handling), SloppyCell uses a simple master/slave architecture enabled by Python's object serialization and dynamic code execution capabilities [88].

Upon import of `RunInParallel.py`, all nodes with `rank != 0` enter a `while` loop and wait for commands. The master node sends commands of the form `(command, args)`. `command` is a string specifying the command to execute, while `args` is a dictionary mapping names of arguments to the appropriate objects. The slave `eval`s the command and sends the return value back to the master. If an exception is raised during the function evaluation, the slave instead sends back the exception object. This architecture ensures that the execution path is very simple on the slaves, minimizing difficulties in synchronization. It does markedly increase the communication overhead, but many of our common calculations can be parallelized into large, coarse operations.

The arguments and return values sent between nodes can be any Python object that can be "pickled", serialized into a string representation. This requires some finesse for our `Network` objects, as dynamically generated functions cannot themselves be pickled. When pickled, an object returns its state as a dictionary from `self.__getstate__`. When unpickled that dictionary is used by `self.__setstate__` to restore the object. To enable pickling (and copying) of `Network` objects, they overload the default `__getstate__` and `__setstate__` methods. In `Network.__getstate__` all dynamic functions are removed from the return dictionary, and in `Network.__setstate__` `self.exec_dynamic_functions` is called to recreate them. (The code for the dynamic functions is pickled with the Network, so it need not be regenerated.)

One source of danger in the current implementation is that it will fail if the slaves call some function that is itself parallelized, as all parallelized functions assume they are being called on the master. Bypassing this limitation may be difficult, as messages can only be sent to workers who are in the 'receive' state. As more parts of the code our parallelized, we may need options to choose a which level parallelization happens.

BIBLIOGRAPHY

[1] Baltimore D (2001) Our genome unveiled. Nature 409:814–816. back

[2] Bruggeman FJ, Westerhoff HV (2007) The nature of systems biology. Trends Microbiol 15:45–50. back

[3] Tyson JJ, Chen K, Novak B (2001) Network dynamics and cell physiology. Nat Rev Mol Cell Biol 2:908–916. back

[4] Lazebnik Y (2002) Can a biologist fix a radio?—Or, what I learned while studying apoptosis. Cancer Cell 2:179–182. back

[5] Yates FE (1978) Good manners in good modeling: mathematical models and computer simulations of physiological systems. Am J Physiol 234:R159–R160. back

[6] Cumming G, Fidler F, Vaux DL (2007) Error bars in experimental biology. J Cell Biol 177:7–11. back

[7] Brown KS, Sethna JP (2003) Statistical mechanical approaches to models with many poorly known parameters. Phys Rev E 68:021904. back

[8] Brown KS, Hill CC, Calero GA, Myers CR, Lee KH, et al. (2004) The statistical mechanics of complex signaling networks: nerve growth factor signaling. Phys Biol 1:184–195. back

[9] Brodersen R, Nielsen F, Christiansen JC, Andersen K (1987) Characterization of binding equilibrium data by a variety of fitted isotherms. Eur J Biochem 169:487–495. back

[10] Frederiksen SL, Jacobsen KW, Brown KS, Sethna JP (2004) Bayesian ensemble approach to error estimation of interatomic potentials. Phys Rev Lett 93:165501. back

[11] Waterfall JJ, Casey FP, Gutenkunst RN, Brown KS, Myers CR, et al. (2006) Sloppy-model universality class and the Vandermonde matrix. Phys Rev Lett 97:150601. back

[12] Casey FP, Baird D, Feng Q, Gutenkunst RN, Waterfall JJ, et al. (2007) Optimal experimental design in an epidermal growth factor receptor signalling and down-regulation model. IET Syst Biol 1:190–202. back

[13] Kuczenski RS, Hong KC, García-Ojalvo J, Lee KH (2007) PERIOD-TIMELESS interval timer may require an additional feedback loop. PLoS Comput Biol 3:e154. back

[14] Buckheit JB, Donoho DL (1995) Wavelets in statistics, Springer-Verlag, chapter WaveLab and Reproducible Research. pp. 55–82. back

[15] Dobzhansky T (1964) Biology, molecular and organismic. Am Zool 4:443–452. back

[16] Dobzhansky T (1973) Nothing in biology makes sense except in the light of evolution. Am Biol Teach 35:125–129. back

[17] Shalizi CR, Tozier WA (1999). A simple model for the evolution of simple models of evolution. arXiv:adap-org/9910002. back

[18] Baxter SM, Day SW, Fetrow JS, Reisinger SJ (2006) Scientific software development is not an oxymoron. PLoS Comput Biol 2:e87. back

[19] Kitano H (2002) Computational systems biology. Nature 420:206–210. back

[20] Locke JCW, Southern MM, Kozma-Bognr L, Hibberd V, Brown PE, et al. (2005) Extension of a genetic network model by iterative experimentation and mathematical analysis. Mol Syst Biol 1:0013. back

[21] Voit E, Neves AR, Santos H (2006) The intricate side of systems biology. Proc Natl Acad Sci USA 103:9452–9457. back

[22] Aldridge BB, Burke JM, Lauffenburger DA, Sorger PK (2006) Physicochemical modelling of cell signalling pathways. Nat Cell Biol 8:1195–1203. back

[23] Ingram PJ, Stumpf MPH, Stark J (2006) Network motifs: structure does not determine function. BMC Genomics 7:108. back

[24] Mayo AE, Setty Y, Shavit S, Zaslaver A, Alon U (2006) Plasticity of the cis-regulatory input function of a gene. PLoS Biol 4:e45. back

[25] Sachs K, Perez O, Pe'er D, Lauffenburger DA, Nolan GP (2005) Causal protein-signaling networks derived from multiparameter single-cell data. Science 308:523–529. back

[26] Maerkl SJ, Quake SR (2007) A systems approach to measuring the binding energy landscapes of transcription factors. Science 315:233–237. back

[27] Minton AP (2001) The influence of macromolecular crowding and macromolecular confinement on biochemical reactions in physiological media. J Biol Chem 276:10577–10580. back

[28] Teusink B, Passarge J, Reijenga CA, Esgalhado E, van der Weijden CC, et al. (2000) Can yeast glycolysis be understood in terms of in vitro kinetics of the constituent enzymes? Testing biochemistry. Eur J Biochem 267:5313–5329. back

[29] Mendes P, Kell D (1998) Non-linear optimization of biochemical pathways: applications to metabolic engineering and parameter estimation. Bioinformatics 14:869–883. back

[30] Jaqaman K, Danuser G (2006) Linking data to models: data regression. Nat Rev Mol Cell Biol 7:813–819. back

[31] Cho KH, Shin SY, Kolch W, Wolkenhauer O (2003) Experimental design in systems biology, based on parameter sensitivity analysis using a Monte Carlo method: A case study for the TNFα-mediated NF-κB signal transduction pathway. Simulation 79:726–739. back

[32] Rodriguez-Fernandez M, Mendes P, Banga JR (2006) A hybrid approach for efficient and robust parameter estimation in biochemical pathways. Biosystems 83:248–265. back

[33] Fell D (1997) Understanding the Control of Metabolism. Ashgate Publishing, 300 pp. back

[34] Wiback S, Famili I, Greenberg HJ, Palsson BØ (2004) Monte Carlo sampling can be used to determine the size and shape of the steady-state flux space. J Theor Biol 228:437–447. back

[35] Famili I, Mahadevan R, Palsson BØ (2005) k-Cone analysis: determining all candidate values for kinetic parameters on a network scale. Biophys J 88:1616–1625. back

[36] Bailey JE (2001) Complex biology with no parameters. Nat Biotechnol 19:503–504. back

[37] Faller D, Klingmuller U, Timmer J (2003) Simulation methods for optimal experimental design in systems biology. Simulation 79:717–725. back

[38] Zak DE, Gonye GE, Schwaber JS, Doyle FJ III (2003) Importance of input perturbations and stochastic gene expression in the reverse engineering of genetic regulatory networks: insights from an identifiability analysis of an in silico network. Genome Res 13:2396–2405. back

[39] Gadkar KG, Varner J, Doyle FJ III (2005) Model identification of signal transduction networks from data using a state regulator problem. IEEE Syst Biol 2:17–30. back

[40] Tyson JJ (1991) Modeling the cell division cycle: cdc2 and cyclin interactions. Proc Natl Acad Sci USA 88:7328–7332. back

[41] Zwolak JW, Tyson JJ, Watson LT (2005) Globally optimised parameters for a model of mitotic control in frog egg extracts. Syst Biol (Stevenage) 152:81–92. back

[42] Goldbeter A (1991) A minimal cascade model for the mitotic oscillator involving cyclin and cdc2 kinase. Proc Natl Acad Sci USA 88:9107–9111. back

[43] Vilar JMG, Kueh HY, Barkai N, Leibler S (2002) Mechanisms of noise-resistance in genetic oscillators. Proc Natl Acad Sci USA 99:5988–5992. back

[44] Edelstein SJ, Schaad O, Henry E, Bertrand D, Changeux JP (1996) A kinetic mechanism for nicotinic acetylcholine receptors based on multiple allosteric transitions. Biol Cybern 75:361–379. back

[45] Kholodenko BN (2000) Negative feedback and ultrasensitivity can bring about oscillations in the mitogen-activated protein kinase cascades. Eur J Biochem 267:1583–1588. back

[46] Lee E, Salic A, Krüger R, Heinrich R, Kirschner MW (2003) The roles of APC and axin derived from experimental and theoretical analysis of the Wnt pathway. PLoS Biol 1:e10. back

[47] Leloup JC, Goldbeter A (1999) Chaos and birhythmicity in a model for circadian oscillations of the PER and TIM proteins in drosophila. J Theor Biol 198:445–459. back

[48] von Dassow G, Meir E, Munro EM, Odell GM (2000) The segment polarity network is a robust developmental module. Nature 406:188–192. back

[49] Ueda HR, Hagiwara M, Kitano H (2001) Robust oscillations within the interlocked feedback model of Drosophila circadian rhythm. J Theor Biol 210:401–406. back

[50] Curto R, Voit EO, Sorribas A, Cascante M (1998) Mathematical models of purine metabolism in man. Math Biosci 151:1–49. back

[51] Chassagnole C, Noisommit-Rizzi N, Schmid JW, Mauch K, Reuss M (2002) Dynamic modeling of the central carbon metabolism of Escherichia coli. Biotechnol Bioeng 79:53–73. back

[52] Chen KC, Calzone L, Csikasz-Nagy A, Cross FR, Novak B, et al. (2004) Integrative analysis of cell cycle control in budding yeast. Mol Biol Cell 15:3841–3862. back

[53] Sasagawa S, Ozaki Y, Fujita K, Kuroda S (2005) Prediction and validation of the distinct dynamics of transient and sustained ERK activation. Nat Cell Biol 7:365–373. back

[54] Le Novère N, Bornstein B, Broicher A, Courtot M, Donizelli M, et al. (2006) BioModels Database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. Nucleic Acids Res 34:D689–691. back

[55] Hucka M, Finney A, Sauro HM, Bolouri H, Doyle JC, et al. (2003) The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. Bioinformatics 19:524–531. back

[56] Gutenkunst RN, Casey FP, Waterfall JJ, Atlas JC, Kuczenski RS, et al. SloppyCell. http://sloppycell.sourceforge.net. back

[57] Mehta ML (2004) Random Matrices. Academic Press, 3rd edition. back

[58] Waterfall JJ (2006) Universality in Multiparameter Fitting: Sloppy Models. Ph.D. thesis, Cornell University. back

[59] Takeda S, Kadowaki S, Haga T, Takaesu H, Mitaku S (2002) Identification of G protein-coupled receptor genes from the human genome sequence. FEBS Lett 520:97–101. back

[60] Wise A, Gearing K, Rees S (2002) Target validation of G-protein coupled receptors. Drug Discov Today 7:235–46. back

[61] Park PH, Filipek S, Wells J, Palczewski K (2004) Oligomerization of G protein-coupled receptors: past, present, and future. Biochemistry 43:15643–15656. back

[62] Levitzki A, Klein S (2002) G-protein subunit dissociation is not an integral part of G-protein action. Chembiochem 3:815–818. back

[63] Preininger AM, Hamm HE (2004) G protein signaling: insights from new structures. Sci STKE 2004:re3. back

[64] Ceruso MA, Periole X, Weinstein H (2004) Molecular dynamics simulations of transducin: interdomain and front to back communication in activation and nucleotide exchange. J Mol Biol 338:469–481. back

[65] Oldham WM, Eps NV, Preininger AM, Hubbell WL, Hamm HE (2006) Mechanism of the receptor-catalyzed activation of heterotrimeric G proteins. Nat Struct Mol Biol 13:772–777. back

[66] Majumdar S, Ramachandran S, Cerione RA (2004) Perturbing the linker regions of the α-subunit of transducin: a new class of constitutively active GTP-binding proteins. J Biol Chem 279:40137–40145. back

[67] Pereira R, Cerione RA (2005) A switch 3 point mutation in the α subunit of transducin yields a unique dominant-negative inhibitor. J Biol Chem 280:35696–35703. back

[68] Majumdar S, Ramachandran S, Cerione RA (2006) New insights into the role of conserved, essential residues in the GTP binding/GTP hydrolytic cycle of large G proteins. J Biol Chem 281:9219–9226. back

[69] Pugh ENJ, Lamb TD (2000) Handook of Biological Physics, Elsevier, volume 3, chapter Phototransduction in Vertebrate Rods and Cones: Molecular Mechanisms of Light Amplification, Recovery and Light Adaptation. pp. 183–255. back

[70] Arshavsky VY, Lamb TD, Pugh EN (2002) G proteins and phototransduction. Annu Rev Physiol 64:153–187. back

[71] Ramachandran S, Cerione RA A conserved threonine in the switch 1 region of

large G-proteins plays important roles in the GTP-binding/GTP hydrolytic cycle. In preparation. back

[72] Skiba NP, Bae H, Hamm HE (1996) Mapping of effector binding sites of transducin α-subunit using $G\alpha_t/G\alpha_{i1}$ chimeras. J Biol Chem 271:413–24. back

[73] Phillips WJ, Cerione RA (1988) The intrinsic fluorescence of the α subunit of transducin. Measurement of receptor-dependent guanine nucleotide exchange. J Biol Chem 263:15498–505. back

[74] Guy PM, Koland JG, Cerione RA (1990) Rhodopsin-stimulated activation-deactivation cycle of transducin: kinetics of the intrinsic fluorescence response of the α subunit. Biochemistry 29:6954–64. back

[75] Jones E, Oliphant T, Peterson P, et al. (2001–). SciPy: open source scientific tools for Python. http://www.scipy.org/. back

[76] Oliphant TE (2007) Python for scientific computing. Comput Sci Eng 9:10–20. back

[77] Kass RE, Raftery AE (1995) Bayes factors. J Am Stat Assoc 90:773–795. back

[78] Schmidt H, Jirstrand M (2006) Systems Biology Toolbox for MATLAB: a computational platform for research in systems biology. Bioinformatics 22:514–515. back

[79] Bergmann FT, Sauro HM (2006) SBW—a modular framework for systems biology. In: WSC '06: Proceedings of the 38th conference on Winter simulation. Winter Simulation Conference, pp. 1637–1645. back

[80] Hoops S, Sahle S, Gauges R, Lee C, Pahle J, et al. (2006) COPASI—a COmplex PAthway SImulator. Bioinformatics 22:3067–3074. back

[81] Zi Z, Klipp E (2006) SBML-PET: a Systems Biology Markup Language-based parameter estimation tool. Bioinformatics 22:2704–2705. back

[82] Gutenkunst RN, Casey FP, Waterfall JJ, Myers CR, Sethna JP (2007) Extracting falsifiable predictions from sloppy models. In: Stolovitsky G, Califano A, Collins J, editors, Reverse Engineering Biological Networks: Oppor-

tunities and Challenges in Computational Methods for Pathway Inference. New York Academy of Sciences. In press, arXiv:0704.3049. back

[83] Mosegaard K, Tarantola A (1995) Monte Carlo sampling of solutions to inverse problems. J Geophys Res 100:12431–12447. back

[84] Chib S, Greenberg E (1989) Understanding the Metropolis-Hastings algorithm. Amer Statistician 49:327–335. back

[85] Swameye I, Muller TG, Timmer J, Sandra O, Klingmuller U (2003) Identification of nucleocytoplasmic cycling as a remote sensor in cellular signaling by databased modeling. Proc Natl Acad Sci USA 100:1028–1033. back

[86] Machné R, Finney A, Müller S, Lu J, Widder S, et al. (2006) The SBML ODE Solver Library: a native API for symbolic and fast numerical analysis of reaction networks. Bioinformatics 22:1406–1407. back

[87] Gunawan R, Taylor SR, Doyle FJ III (2005) Sensitivity analysis in biological modeling: an application in the model development of staphylococcal enterotoxin B pre-apoptotic pathways. In: Proceedings of the AIChE Annual Meeting. back

[88] Myers CR, Gutenkunst RN, Sethna JP (2007) Python unleashed on systems biology. Comput Sci Eng 9:34–37. back

[89] Olivier BG, Rohwer JM, Hofmeyr JHS (2002) Modelling cellular processes with Python and Scipy. Mol Biol Rep 29:249–254. back

[90] Olivier BG, Rohwer JM, Hofmeyr JHS (2005) Modelling cellular systems with PySCeS. Bioinformatics 21:560–561. back

[91] Poolman MG (2006) ScrumPy: metabolic modelling with Python. IEE Proc Syst Biol 153:375–378. back

[92] Pérez F, Granger BE (2007) IPython: a system for interactive scientific computing. Comput Sci Eng 9:21–29. back

[93] Hunter JD (2007) Matplotlib: a 2D graphics environment. Comput Sci Eng 9:90–95. back

[94] Brown PN, Hindmarsh AC, Petzold LR (1994) Using Krylov methods in the

solution of large-scale differential-algebraic systems. SIAM J Sci Comput 15:1467–1488. back

[95] Brown PN, Hindmarsh AC, Petzold LR (1998) Consistent initial condition calculation for differential-algebraic systems. SIAM J Sci Comput 19:1495–1512. back

[96] Gillespie DT (1977) Exact stochastic simulation of coupled chemical reactions. J Phys Chem-US 81:2340–2361. back

[97] Funahashi A, Morohashi M, Kitano H, Tanimura N (2003) CellDesigner: a process diagram editor for gene-regulatory and biochemical networks. BIOSILICO 1:159–162. back

[98] Kutschera U, Niklas KJ (2004) The modern theory of biological evolution: an expanded synthesis. Naturwissenschaften 91:255–276. back

[99] Orr HA (2005) Theories of adaptation: what they do and don't say. Genetica 123:3–13. back

[100] Orr HA (2005) The genetic theory of adaptation: a brief history. Nat Rev Genet 6:119–127. back

[101] Elena SF, Lenski RE (2003) Evolution experiments with microorganisms: the dynamics and genetic bases of adaptation. Nat Rev Genet 4:457–469. back

[102] Gillespie JH (1984) Molecular evolution over the mutational landscape. Evolution 38:1116–1129. back

[103] Gumbel EJ (1958) Statistics of Extremes. Columbia University Press. back

[104] Orr HA (2002) The population genetics of adaptation: the adaptation of DNA sequences. Evolution 56:1317–1330. back

[105] Orr HA (2003) A minimum on the mean number of steps taken in adaptive walks. J Theor Biol 220:241–247. back

[106] Rokyta DR, Joyce P, Caudle SB, Wichman HA (2005) An empirical test of the mutational landscape model of adaptation using a single-stranded DNA virus. Nat Genet 37:441–444. back

[107] Kassen R, Bataillon T (2006) Distribution of fitness effects among beneficial mutations before selection in experimental populations of bacteria. Nat Genet 38:484–488. back

[108] Fisher RA (1930) The genetical theory of natural selection. Oxford Univ Press. back

[109] Kimura M (1983) The neutral theory of molecular evolution. Cambridge Univ. Press. back

[110] Orr HA (1998) The population genetics of adaptation: the distribution of factors fixed during adaptive evolution. Evolution 52:935–949. back

[111] Orr HA (1999) The evolutionary genetics of adaptation: a simulation study. Genet Res 74:207–214. back

[112] Hartl DL, Taubes CH (1998) Towards a theory of evolutionary adaptation. Genetica 102-103:525–533. back

[113] Poon A, Otto SP (2000) Compensating for our load of mutations: freezing the meltdown of small populations. Evolution 54:1467–1479. back

[114] Orr HA (2000) Adaptation and the cost of complexity. Evolution 54:13–20. back

[115] Welch JJ, Waxman D (2003) Modularity and the cost of complexity. Evolution 57:1723–1734. back

[116] Tenaillon O, Silander OK, Uzan JP, Chao L (2007) Quantifying organismal complexity using a population genetic approach. PLoS ONE 2:e217. back

[117] Martin G, Elena SF, Lenormand T (2007) Distributions of epistasis in microbes fit predictions from a fitness landscape model. Nat Genet 39:555–560. back

[118] Orr HA (2006) The distribution of fitness effects among beneficial mutations in Fisher's geometric model of adaptation. J Theor Biol 238:279–285. back

[119] Clarke B, Arthur W (2000) What constitutes a 'large' mutational change in phenotype? Evol Dev 2:238–240. back

[120] Orr HA (2001) The "sizes" of mutations fixed in phenotypic evolution: a response to Clarke and Arthur. Evol Dev 3:121–3. back

[121] Arthur W (2001) Why imperfect steps in the right direction attract criticism. Evol Dev 3:125–126. back

[122] Fong SS, Marciniak JY, Palsson BØ (2003) Description and interpretation of adaptive evolution of Escherichia coli K-12 MG1655 by using a genome-scale in silico metabolic model. J Bacteriol 185:6400–6408. back

[123] Franois P, Hakim V (2004) Design of genetic networks with specified functions by evolution in silico. Proc Natl Acad Sci USA 101:580–585. back

[124] Adam GC, Sorensen EJ, Cravatt BF (2002) Proteomic profiling of mechanistically distinct enzyme classes using acommon chemotype. Nat Biotech 20:805–809. back

[125] Hillig KW, Mahlberg PG (2004) A chemotaxonomic analysis of cannabinoid variation in Cannabis (Cannabaceae). Am J Bot 91:966–975. back

[126] Zubova S, Ivanov A, Prokhorenko I (2007) Relations between the chemotype of Rhodobacter capsulatus strains and the cell electrophoretic properties. Microbiology 76:177–181. back

[127] Peck JR, Barreau G, Heath SC (1997) Imperfect genes, Fisherian mutation and the evolution of sex. Genetics 145:1171–1199. back

[128] Martin G, Lenormand T (2006) The fitness effect of mutations across environments: a survey in light of fitness landscape models. Evolution 60:2413–2427. back

[129] Waxman D, Welch JJ (2005) Fisher's microscope and Haldane's ellipse. Am Nat 166:447–457. back

[130] Eyre-Walker A, Keightley PD (2007) The distribution of fitness effects of new mutations. Nat Rev Genet 8:610–618. back

[131] Perfeito L, Fernandes L, Mota C, Gordo I (2007) Adaptive mutations in bacteria: high rate and small effects. Science 317:813–815. back

[132] Rozen DE, de Visser JAGM, Gerrish PJ (2002) Fitness effects of fixed beneficial mutations in microbial populations. Curr Biol 12:1040–1045. back

[133] Sella G, Hirsh AE (2005) The application of statistical physics to evolutionary biology. Proc Natl Acad Sci USA 102:9541–9546. back

[134] Waxman D (2007) Mean curvature versus normality: a comparison of two approximations of Fisher's geometrical model. Theor Popul Biol 71:30–36. back

[135] Bloom JD, Romero PA, Lu Z, Arnold FH (2007) Neutral genetic drift can alter promiscuous protein functions, potentially aiding functional evolution. Biol Direct 2:17. back

[136] Landry CR, Lemos B, Rifkin SA, Dickinson WJ, Hartl DL (2007) Genetic properties influencing the evolvability of gene expression. Science 317:118–121. back

[137] Rosenfeld N, Young JW, Alon U, Swain PS, Elowitz MB (2005) Gene regulation at the single-cell level. Science 307:1962–1965. back

[138] Wood TE, Burke JM, Rieseberg LH (2005) Parallel genotypic adaptation: when evolution repeats itself. Genetica 123:157–170. back

[139] Pelosi L, Khn L, Guetta D, Garin J, Geiselmann J, et al. (2006) Parallel changes in global protein profiles during long-term experimental evolution in Escherichia coli. Genetics 173:1851–1869. back

[140] Imhof M, Schlotterer C (2001) Fitness effects of advantageous mutations in evolving escherichia coli populations. Proc Natl Acad Sci USA 98:1113–1117. back

[141] Barrett RDH, MacLean RC, Bell G (2006) Mutations of intermediate effect are responsible for adaptation in evolving Pseudomonas fluorescens populations. Biol Lett 2:236–238. back

[142] Abramowitz M, Stegun IA (1964) Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables. Dover, 10th edition. back

[143] Mezey JG, Houle D (2005) The dimensionality of genetic variation for wing shape in Drosophila melanogaster. Evolution 59:1027–1038. back

[144] Griswold CK, Logsdon B, Gomulkiewicz R (2007) Neutral evolution of multiple quantitative characters: a genealogical approach. Genetics 176:455–466. back

[145] Gruner SM, Bilderback D, Bazarov I, Finkelstein K, Krafft G, et al. (2002) Energy recovery linacs as synchrotron radiation sources. Rev Sci Intrum 73:1402–1406. back

[146] Hoffstaetter GH, Bazarov IV, Bilderback D, Billing M, Gruner S, et al. (2004) ERL upgrade of an existing X-ray facility: CHESS at CESR. In: Proceedings of the European Particle Accelerator Conference. pp. 497–499. back

[147] Sagan D, Smith J (2005) The TAO accelerator simulation program. In: Proceedings of the Particle Accelerator Conference. pp. 4159–4161. back

[148] Urban J, Fields L, Sagan D (2005) Linear accelerator simulations with Bmad. In: Proceedings of the Particle Accelerator Conference. pp. 1937–1939. back

[149] Berman GJ, Wang ZJ (2007) Energy-minimizing kinematics in hovering insect flight. J Fluid Mech 582:153–168. back

[150] Sethna JP (2006) Statistical Mechanics: Entropy, Order Parameters and Complexity. Oxford University Press. back

[151] Rosenbrock HH (1960) An automatic method for finding the greatest or least value of a function. Computer J 3:175–184. back

[152] Desai MM, Fisher DS, Murray AW (2007) The speed of evolution and maintenance of variation in asexual populations. Curr Biol 17:385–394. back

[153] Lutz M, Ascher D (2003) Learning Python. O'Reilly, 2nd edition. back

[154] Smith D (2005). Re: [Mingw-users] __EH_FRAME_BEGIN_ - a simpler demonstration of the problem. Mingw-users mailing list, March 13. back

[155] Reedy T (2006). Re: Too many if statements? comp.lang.python newsgroup, Feb 10. back

[156] Errico RM (1997) What is an adjoint model? B Am Meteorol Soc 78:2577–2591. back

[157] Cao Y, Li S, Petzold L, Serban R (2002) Adjoint sensitivity analysis for differential-algebraic equations: the adjoint DAE system and its numerical solution. SIAM J Sci Comput 24:1076–1089. back