UNIVERSITY OF CALIFORNIA, SAN DIEGO

A Path Integral Approach to Data Assimilation in Stochastic Nonlinear Systems

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Physics

by

John C. Quinn

Committee in charge:

Professor Henry D. I. Abarbanel, Chair
Professor Clifford Surko, Co-Chair
Professor Philip Gill
Professor Katja Lindenberg
Professor Tom Murphy

2010

The dissertation of John C. Quinn is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____
Co-Chair

_____
Chair

University of California, San Diego

2010

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank my advisor, Professor Henry Abarbanel, for his guidance, support, and patience. His deep knowledge of and enthusiasm for Physics, with a capital P, was an inspiration to me.

I would also like to acknowledge Paul Bryant for many interesting discussions, and for setting a good example of how to be thorough and precise in scientific pursuits.

I appreciate the valuable time that each of my dissertation committee members, Professors Philip Gill, Katja Lindenberg, Tom Murphy, and Clifford Surko, has devoted.

I would like to acknowledge the Nonneutral Plasma Physics group at UCSD, headed by Professors Tom O'Neil, Fred Driscoll, and Dan Dubin, for their support and guidance.

I would like to thank all my fellow students in the research group: Dan Creveling, Sallee Klein, Leif Gibb, Reza Farsian, Mark Kostuk, Bryan Toth, Chris Knowlton, and William G. Whartenby for making things more interesting and enjoyable. Thanks to Marius Buibas for stimulating discussions and help with CUDA programming, and to Chris Schroeder for help with Monte Carlo methods.

The completion of this dissertation would not have been possible without emotional support from family and friends. I especially appreciate the friendship of fellow graduate students and roommates, Mike Anderson and Toby Weber. Most of all, I would like to thank my Aunt Erin and Uncle Rod for taking me to many amazing places, and for providing countless delicious meals and crunchy snacks over the past seven years.

VITA

2003	B. S., Rutgers University
2005	M. S. in Physics, University of California, San Diego
2003-2010	Graduate Student, University of California, San Diego
2010	Ph. D. in Physics, University of California, San Diego

PUBLICATIONS

J.C. Quinn, H.D.I. Abarbanel, "State and parameter estimation using Monte Carlo evaluation of path integrals", (accepted to the Quarterly Journal of the Royal Meteorological Society, July 2010)

J.C. Quinn, P.H. Bryant, D.R. Creveling, S.R. Klein, H.D.I. Abarbanel, "Parameter and state estimation of experimental chaotic systems using synchronization", Physical Review E **80** (2009)

T. Driscoll, J. Quinn, S. Klein, H.T. Kim, B.J. Kim, Yu. V. Pershin, M. Di Ventra, D.N. Basov, "Memristive adaptive filters", Applied Physics Letters **97** (2010)

CONFERENCE POSTERS

J.C. Quinn, H.D.I. Abarbanel, "Statistical data assimilation using Monte Carlo evaluation of path integrals", Dynamics Days, Evanston 2010

J.C. Quinn, D.H.E. Dubin, "Spatial Landau Damping of Diocotron Modes in Nonneutral Disc Plasmas", APS Division of Plasma Physics, Orlando 2007

J.C. Quinn, D.H.E. Dubin, "Diocotron Modes in Nonneutral Disc Plasmas", APS Division of Plasma Physics, Philadelphia 2006

ABSTRACT OF THE DISSERTATION

A Path Integral Approach to Data Assimilation in Stochastic Nonlinear Systems

by

John C. Quinn

Doctor of Philosophy in Physics

University of California, San Diego, 2010

Professor Henry D. I. Abarbanel, Chair
Professor Clifford Surko, Co-Chair

In this dissertation the problem of data assimilation in stochastic nonlinear systems is formulated using path integrals. Each path represents a time evolution of the model states, and the time independent model parameters. In the path integral, every possible path is integrated over with each path weighted by $P(\mathbf{X}|\mathbf{Y}) \propto \exp[-A_0(\mathbf{X},\mathbf{Y})]$, where $A_0(\mathbf{X},\mathbf{Y})$ is the action, which quantifies how likely it is that the given path $\mathbf{X}$ was actually realized in the experiment which produced the observed time series $\mathbf{Y}$.

The goal of data assimilation is to combine information from a measurement time series with a dynamical model to make statistical estimates or predictions of model states and parameters. Both the measurements and the dynamical model may be noisy, and this fact is incorporated by using a probabilistic formulation for $P(\mathbf{X}|\mathbf{Y})$, the posterior path distribution conditioned on the observed time series.

With an expression for $P(\mathbf{X}|\mathbf{Y})$ it is possible to express expectation values, conditioned upon the the observations, of any function of the path as a path integral over all possible paths. The path integrals can then be numerically approximated using a Markov chain Monte Carlo method such as the Metropolis method. This method is

discussed and applied to two example systems: the Colpitts oscillator circuit, and the Lorenz 96 toy atmosphere model.

By studying the characteristics of the action as a function of the path, properties of the data assimilation problem can be deduced. For instance, if the surface in path space defined by the action is rough with many local minima with similar values of action, then the data assimilation problem is not well-defined. If more observations are made which rule out regions of path space that were previously likely, then the surface may become smoother with a single minimum. By examining the shape of the action, the question of how many measurements are needed to fully reconstruct the model state can be answered. It is also important to examine the shape of the action in the vicinity of the global minimum to find the level of uncertainty in state and parameter estimates. These ideas are illustrated with the Lorenz 96 system as an example.

Chapter 1

Introduction

1.1 Goal of Data Assimilation

A common strategy in many areas of science is to combine theoretical models with experimental observations in order to learn something about a dynamical system. The dynamical model may be created based on ideas about what the underlying mechanisms are, or simply to reproduce observed phenomena. In either case there are usually parameters in the model that are not known from first principles. In addition, the models often have dynamical state variables that cannot directly be measured, but which are useful to estimate as a function of time, either to learn how the unobservable states evolved or to predict how an observable will evolve. Data assimilation is the process of combining experimental observations with models to produce statistical estimates of the model states and parameters that cannot be directly measured.

When dealing with real systems there is always some degree of noise or uncertainty. The function that relates the true state to the observation is typically not deterministic, and the time evolution of the state is not perfectly described by the model. Data assimilation methods account for these facts by formulating the dynamics and the measurement process in a probabilistic way, and by generating probabilistic state and parameter estimates.

There are many established approaches to this problem [26, 16, 57, 12, 59, 45, 31, 40, 35, 48, 28, 13], some which are discussed in Chapter 5. This dissertation develops an approach to the problem of data assimilation based on a formulation in terms of path integrals [4, 55, 6, 3]. The paths in question represent every possible time evolution of

the model state. Each path is assigned a probability based on how consistent it is with the model and with the data. The path integrals then are integrals over every possible path, each one weighted by its probability. This formulation is developed in Chapter 2.

Even after the problem is formulated, there still remains the challenge of actually evaluating the path integrals. Chapter 3 illustrates how the Metropolis [41] path integral Monte Carlo (PIMC) method can be used for this purpose. In Chapter 4 the structure of the conditional probability distribution in path space is examined for some specific examples, and the role of observations is illuminated. The question of the number of observations needed to reconstruct the state is also addressed.

1.2 Example Systems

Two particular dynamical systems are used throughout this dissertation to demonstrate application of the PIMC data assimilation method. One of them is the Colpitts oscillator circuit [30] and one is the toy atmosphere model of Lorenz [36, 37].

1.2.1 Colpitts Oscillator

The Colpitts oscillator is the simple circuit shown in Fig. 1.1, which consists of two resistors, two capacitors, an inductor, and a bipolar junction transistor (BJT). It is a damped resonant circuit with the transistor providing an occasional driving force via nonlinear switching action. The state of the circuit can be fully described by three dynamical variables: $V_{CE}(t)$ the voltage at the collector relative to the emitter, $V_E(t)$ voltage at the emitter relative to ground, and $I_L(t)$ the current through the inductor. The state can be thought of as a point in a three-dimensional phase space, and the dynamics is represented as a trajectory in phase space.

This version of the circuit has one adjustable parameter, R, which is controlled with a potentiometer. When R is large the system is at a fixed point: there is no motion in phase space. As R decreases the system begins oscillating periodically. As R decreases further the system undergoes a series of period doubling bifurcations, and then the oscillations become chaotic. This is illustrated in the bifurcation diagram in Fig. 1.2 as a function of the resistance R. This is a plot of $V_{CE}(t_n)$ where t_n are the times when $V_E = -0.6$ V and $\frac{dV_E}{dt} > 0$.

The Colpitts oscillator is often used for practical purposes in the periodic regime, but here we are mainly interested in the chaotic behavior to use the circuit as a chal-

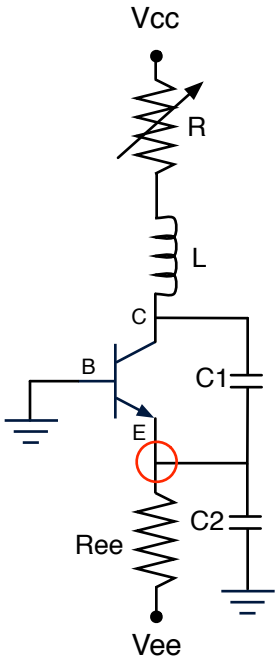

Figure 1.1: Schematic of the Colpitts oscillator circuit.

lenging test system to apply data assimilation methods to. The chaotic dynamics makes the data assimilation problem challenging, because the model output is highly sensitive to the parameter settings and initial conditions. The circuit is a good test system for several reasons: it can be easily built, all state variables can be measured, it can be accurately modeled, and it can behave chaotically.

The circuit dynamics can be described by three coupled first-order differential equations which can be found by a straightforward application of Kirchoff's laws. They are:

$$\begin{aligned} C_1 \frac{dV_{CE}}{dt} &= I_L(t) - I_C(V_E) \\ C_2 \frac{dV_E}{dt} &= I_L(t) - \frac{V_E(t) - V_{EE}}{R_{EE}} + I_B(V_E) \\ L \frac{dI_L}{dt} &= V_{CC} - V_E(t) - V_{CE}(t) - RI_L(t), \end{aligned} \tag{1.1}$$

where $I_C(V_E)$ and $I_B(V_E)$ are the currents into the collector and base of the transistor respectively. To describe the transistor behavior we use a simplified version of the Ebers-Moll model [15, 39] for the transistor:

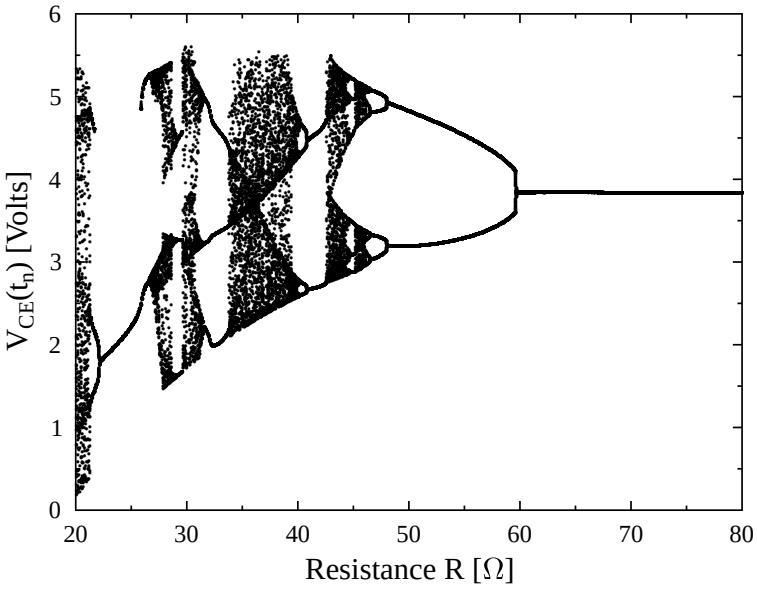

Figure 1.2: Bifurcation diagram for the Colpitts oscillator model as a function of the control parameter R. Period doubling bifurcations and windows of chaos are visible.

$$I_C(V_E) = (1\text{ mA})\exp\left(\frac{-V_E - V_0}{V_{th}}\right)$$
$$I_B(V_E) = \frac{I_C(V_E)}{\beta_F}. \tag{1.2}$$

These equations specify the transistor currents as a function of V_E, and in terms of two transistor properties: the forward current gain β_F, and a voltage offset V_0. The other parameter is the thermal voltage $V_{th} = kT/e$.

Our goal is to estimate the parameters R, V_0, L, C_2, β, and V_{th}, as well as the unobserved state variables $V_{CE}(t)$ and $I_L(t)$ from a time series recording of $V_E(t)$. An example of $V_E(t)$ time series data recorded at a sampling rate of $\Delta t = 0.01$ ms from the circuit is shown in Fig. 1.3. The other two state variables, $V_{CE}(t)$ and $I_L(t)$, were also recorded but only to compare to the state estimation results at the end of the data assimilation procedure. In this example the circuit parameters may be directly measured by taking the circuit apart and measuring the individual circuit elements, but there are many systems where it is difficult or impossible to directly measure all the parameters. In Chapter 3 we will see the results of using the PIMC data assimilation method using data recorded from an actual Colpitts oscillator circuit.

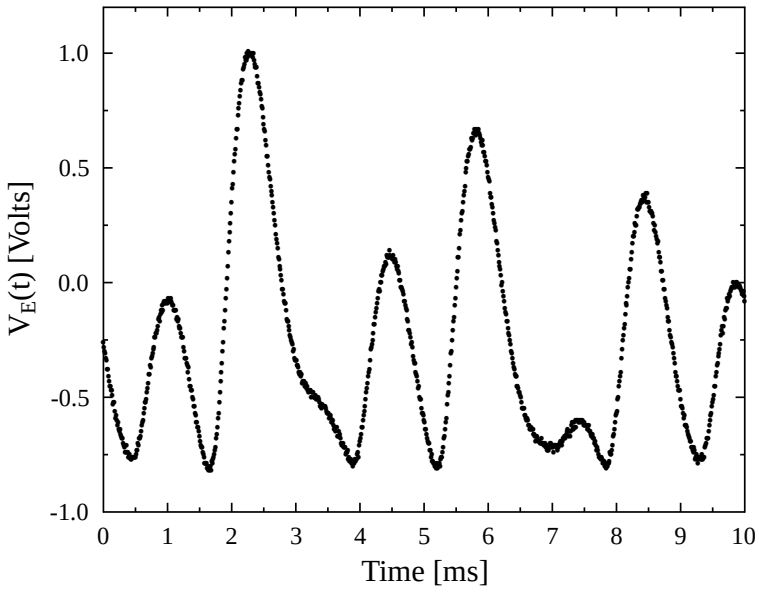

Figure 1.3: Voltage time series $V_E(t)$ recorded from a Colpitts circuit operating in the chaotic regime.

1.2.2 Lorenz 96

The model of Lorenz [36, 37] is the set of coupled differential equations

$$\frac{dw_l(t)}{dt} = w_{l-1}(t)\left[w_{l+1}(t) - w_{l-2}(t)\right] - w_l(t) + f_0, \tag{1.3}$$

with the state variables labeled by $l = 0, 1, \ldots, D-1$. The state variables are arranged on a ring, so $w_D = w_0$, $w_{-1} = w_{D-1}$, and $w_{-2} = w_{D-2}$. The state variables represent some unspecified scalar quantity at a location on the ring labeled by l. This model was created as a toy version of an atmospheric model. The quadratic terms represent advection of the quantity around the ring, and conserve $(w_0^2 + \ldots + w_{D-1}^2)$. The linear term, $-w_l(t)$, represents damping, and the constant term, $+f_0$, represents an external driving force.

This model was originally invented by Lorenz as an example system that could be used to address the question of how many measurements are required to be able to fully reconstruct the state of the system. The index l can be thought of as labeling longitudes around the globe. The observations will come from weather stations positioned sparsely at various longitudes. With this picture in mind, the question becomes: how many weather stations are needed and where should they be positioned in order to be able

to sufficiently estimate all the state variables of the model? Another way of asking the question is: given a model and a set of observations over time from different locations, how accurately can we estimate the current state of the atmosphere, and thus predict the future states?

In Chapter 3 we will see how the Monte Carlo evaluation of the data assimilation path integral can be applied to the Lorenz 96 system using simulated data taken within an observation window, $0 \leq t \leq t_M$. In this 'twin experiment' we get to decide how many and which state variables to measure. We also can pick a data sampling rate and a distribution of simulated noise of some amplitude. From the artificial measurements we can estimate the state variables and the associated uncertainties at each time step, as well as project the state distribution at $t = t_M$ forward in time to make predictions.

In Chapter 4 we again study the Lorenz 96 model, this time looking more closely at the structure of the action function (as formulated in Chapter 2) associated with this problem. We will see that when not enough measurements are available the surface of the action is rough with many local minima with similar values of action, but as the number of measurements are increased the surface becomes smooth with only one minimum.

Chapter 2

Probabilistic Formulation

Real physical systems are never perfectly described by models, and real measurements are never without noise. This suggests formulating the problem of data assimilation in terms of probability which is done in [58, 48, 11, 22, 5] and by many others. The goal is to find a probability distribution for all the states of the model as a function of time from noisy measurements of some property of the system over time.

The inputs into this formulation will be a model of the system and a time series of observations, together with uncertainties associated with these two ingredients. The model will be in the form of Markov transition probabilities, which describe how the model state evolves in time. The transition probabilities may come from a discrete time map or a system of ordinary differential equations that are then discretized. In this chapter, we will show how these ingredients are combined to form a probability distribution that is a function of the time history of the model state conditioned on the observations.

The first ingredient is the model. To make a model, we first represent the state of the system at each time step $n \in \{0, 1, \ldots, M\}$ as a D-dimensional vector $\mathbf{x}_n$. We assume the dynamical model is Markov, and so we can represent the time evolution using a transition probability $P(\mathbf{x}_{n+1}|\mathbf{x}_n)$ which depends only on $\mathbf{x}_n$ and not any previous states. This tells us the probability of transitioning from state $\mathbf{x}_n$ at time step n to state $\mathbf{x}_{n+1}$ at time step $n+1$. This may be a deterministic transition in which case $P(\mathbf{x}_{n+1}|\mathbf{x}_n)$ will be a delta function, or a stochastic transition in which case $P(\mathbf{x}_{n+1}|\mathbf{x}_n)$ will be something more spread out than a delta function. The D dimensions of the model may include any time-independent parameters, by promoting the parameters to

state variables with time evolution given by $\dot{\mathbf{p}} = 0$.

The second ingredient is a time series of measurements. At each time step labeled by $n \in \{1, 2, \ldots, M\}$ the measurement is an L-dimensional vector $\mathbf{y}_n$ which is a known function of the D-dimensional state vector $\mathbf{x}_n$. Typically in practice $L < D$, so there are 'hidden states'. To simplify notation without loss of generality, we can assume that the measurement function is simply $y_{n,l} = x_{n,l}$+noise for $l \in \{1, 2, \ldots, L\}$. The noise term will be discussed in more detail later. The first subscript is the time index and the second subscript is the vector component index. In general the measurements may be complicated functions of the state vector $\mathbf{x}_n$, but we may transform the model into a coordinate system where the measurements are simply projections along the first L coordinate axes of the state space. The time series of measurements is represented by $\mathbf{Y} = \mathbf{y}_{1:M} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_M\}$.

We then consider the discrete time evolution of the model state by defining a path variable $\mathbf{X} = \mathbf{x}_{0:M} = \{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_M\}$. This describes a trajectory through the the D-dimensional state space. This formulation considers the entire path simultaneously, assuming all the observations have already been collected. In signal processing language this method would be called a 'smoother', as opposed to a 'filter' which only uses observations from the past. It is also sometimes useful to think of $\mathbf{X}$ as identifying a single point in path space which has dimension $D \times (M + 1)$.

The 'true path' is defined as the sequence of states that describes what the actual evolution of the system was. We call the true path $\mathbf{W} = \{\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_M\}$, where each $\mathbf{w}_n$ is a D-dimensional vector. The observation time series $\mathbf{Y}$ is produced from the true path by making noisy measurements of only some of the states. From this limited information we wish to find $P(\mathbf{X}|\mathbf{Y})$, a probability distribution as a function of $\mathbf{X}$ conditioned on the specific data $\mathbf{Y}$. The true path $\mathbf{W}$ which when measured produced $\mathbf{Y}$ can be thought of as a specific realization of the random path variable $\mathbf{X}$ drawn from the distribution $P(\mathbf{X}|\mathbf{Y})$.

Since the process is stochastic, we cannot say what $\mathbf{W}$ was with certainty given $\mathbf{Y}$. The best we can do is to find the distribution $P(\mathbf{X}|\mathbf{Y})$ from which $\mathbf{W}$ was drawn. This will allow us to find an estimate for $\mathbf{W}$ and an associated uncertainty.

2.1 Bayesian Formulation of the Conditional Probability Distribution

We would like to find $P(\mathbf{X}|\mathbf{Y})$, the probability distribution of paths conditioned on the time series data. It should turn out that this probability is high for paths that are consistent with both the observations and with model, and low for paths that are inconsistent with the observations or inconsistent with the model. We now show how this comes out of an application of Bayesian probability.

As a first step we calculate $P(\mathbf{x}_n|\mathbf{y}_{1:n})$ in terms of $P(\mathbf{x}_n|\mathbf{y}_{1:n-1})$, so that we can see how the newest measurement $\mathbf{y}_n$ modifies the the probability of $\mathbf{x}_n$. This is the where the observations enter into the formulation. We use the definition of conditional probability $P(A|B) = P(A,B)/P(B)$ and the fact that $P(\mathbf{y}_{1:n}) = P(\mathbf{y}_n, \mathbf{y}_{1:n-1})$ to write

$$\begin{aligned} P(\mathbf{x}_n|\mathbf{y}_{1:n}) &= \frac{P(\mathbf{x}_n, \mathbf{y}_n, \mathbf{y}_{1:n-1})}{P(\mathbf{y}_n, \mathbf{y}_{1:n-1})} \\ &= \frac{P(\mathbf{x}_n, \mathbf{y}_n|\mathbf{y}_{1:n-1})}{P(\mathbf{y}_n|\mathbf{y}_{1:n-1})} \\ &= \frac{P(\mathbf{y}_n|\mathbf{x}_n, \mathbf{y}_{1:n-1})}{P(\mathbf{y}_n|\mathbf{y}_{1:n-1})} P(\mathbf{x}_n|\mathbf{y}_{1:n-1}). \end{aligned} \tag{2.1}$$

The identity $P(A, B|C) = P(A|C)P(B|A, C)$ was used on the last step.

The other piece that is needed comes from the dynamics prescribed by the model. The model gives us a way to advance the state in time: it lets us calculate $P(\mathbf{x}_n|\mathbf{y}_{1:n-1})$ from $P(\mathbf{x}_{n-1}|\mathbf{y}_{1:n-1})$. This is done by using the Markov transition probabilities specified by the model to advance the state forward one time step. When the identity,

$$\begin{aligned} P(\mathbf{x}_n|\mathbf{y}_{1:n-1}) &= \int d\mathbf{x}_{n-1} P(\mathbf{x}_n, \mathbf{x}_{n-1}|\mathbf{y}_{1:n-1}) \\ &= \int d\mathbf{x}_{n-1} P(\mathbf{x}_n|\mathbf{x}_{n-1}, \mathbf{y}_{1:n-1}) P(\mathbf{x}_{n-1}|\mathbf{y}_{1:n-1}), \end{aligned}$$

is combined with the assumption that the model is Markov, meaning that the state at the next time step depends only on the current state, which implies $P(\mathbf{x}_n|\mathbf{x}_{n-1}, \mathbf{y}_{1:n-1}) = P(\mathbf{x}_n|\mathbf{x}_{n-1})$, we get the Chapman-Kolmogorov equation [44]

$$P(\mathbf{x}_n|\mathbf{y}_{1:n-1}) = \int d\mathbf{x}_{n-1} P(\mathbf{x}_n|\mathbf{x}_{n-1}) P(\mathbf{x}_{n-1}|\mathbf{y}_{1:n-1}). \tag{2.2}$$

This gives us a way to calculate $P(\mathbf{x}_n|\mathbf{y}_{1:n-1})$ from $P(\mathbf{x}_{n-1}|\mathbf{y}_{1:n-1})$ by using the model to predict one time step forward, and can be combined with Eq. (2.1) to yield

$$P(\mathbf{x}_n|\mathbf{y}_{1:n}) = \frac{P(\mathbf{y}_n|\mathbf{x}_n, \mathbf{y}_{1:n-1})}{P(\mathbf{y}_n|\mathbf{y}_{1:n-1})} \int d\mathbf{x}_{n-1} P(\mathbf{x}_n|\mathbf{x}_{n-1}) P(\mathbf{x}_{n-1}|\mathbf{y}_{1:n-1}). \tag{2.3}$$

This equation gives a way to compute $P(\mathbf{x}_n|\mathbf{y}_{1:n})$ given $P(\mathbf{x}_{n-1}|\mathbf{y}_{1:n-1})$ by incorporating the model prediction and the information from the most recent measurement. This is the key equation that is used by all recursive Bayesian estimation methods. It is the starting point for particle filter methods [7] which use this equation to perform the forecast step to advance the probability distribution, as approximated by a weighted particle distribution, and the analysis step to incorporate measurements. Particle filtering methods, which are discussed in more detail in Chapter 5, process the measurements and update the model state sequentially. Here we take a different approach and consider the entire model evolution and the whole time series of observations simultaneously.

To write $P(\mathbf{x}_M|\mathbf{y}_{1:M})$ in a non-recursive way we first look at the first time step by plugging $n = 1$ into Eq. (2.3) to get

$$P(\mathbf{x}_1|\mathbf{y}_{1:1}) = \frac{P(\mathbf{y}_1|\mathbf{x}_1)}{P(\mathbf{y}_1)} \int d\mathbf{x}_0 P(\mathbf{x}_1|\mathbf{x}_0)P(\mathbf{x}_0),$$

and we note that $\mathbf{y}_{1:0}$ means there are no measurements to condition the probabilities by. We then apply Eq. (2.3) iteratively $M - 1$ additional times to get

$$P(\mathbf{x}_M|\mathbf{y}_{1:M}) = \int \prod_{n=1}^{M} d\mathbf{x}_{n-1} \frac{P(\mathbf{y}_n|\mathbf{x}_n, \mathbf{y}_{1:n-1})}{P(\mathbf{y}_n|\mathbf{y}_{1:n-1})} P(\mathbf{x}_n|\mathbf{x}_{n-1})P(\mathbf{x}_0).$$

This is the marginal distribution of the state at the last time point, $\mathbf{x}_M$. We can use the definition of a marginal distribution

$$P(\mathbf{x}_M|\mathbf{y}_{1:M}) = \int d\mathbf{x}_0 \dots d\mathbf{x}_{M-1} P(\mathbf{x}_{0:M}|\mathbf{y}_{1:M}),$$

to express the conditional probability distribution as a function of the entire path $\mathbf{x}_{0:M}$ as

$$P(\mathbf{x}_{0:M}|\mathbf{y}_{1:M}) = \prod_{n=1}^{M} \left[\frac{P(\mathbf{y}_n|\mathbf{x}_n, \mathbf{y}_{1:n-1})}{P(\mathbf{y}_n|\mathbf{y}_{1:n-1})} \right] P(\mathbf{x}_n|\mathbf{x}_{n-1})P(\mathbf{x}_0). \tag{2.4}$$

To calculate the marginal distribution at a particular time step $0 \leq n \leq M$ we integrate over the state variables at all the other time steps

$$P(\mathbf{x}_n|\mathbf{y}_{1:M}) = \int d\mathbf{x}_0 \dots d\mathbf{x}_{n-1} d\mathbf{x}_{n+1} \dots d\mathbf{x}_M P(\mathbf{x}_{0:M}|\mathbf{y}_{1:M}).$$

Note that this formulation for $P(\mathbf{x}_n|\mathbf{y}_{1:M})$ includes the information gained from the whole time series of measurements $\mathbf{y}_{1:M}$ and not just the the past measurements $\mathbf{y}_{1:n}$ which is the case in filtering methods.

2.1.1 Information Theory Interpretation

The conditional mutual information of $\mathbf{x}_n$ and $\mathbf{y}_n$ given $\mathbf{y}_{1:n-1}$ is defined as [19]

$$\mathrm{MI}(\mathbf{x}_n, \mathbf{y}_n|\mathbf{y}_{1:n-1}) = \log\left[\frac{P(\mathbf{x}_n, \mathbf{y}_n|\mathbf{y}_{1:n-1})}{P(\mathbf{x}_n|\mathbf{y}_{1:n-1})P(\mathbf{y}_n|\mathbf{y}_{1:n-1})}\right], \tag{2.5}$$

and is a measure of the amount of information that is learned about the state $\mathbf{x}_n$ by observing $\mathbf{y}_n$, given all the previous observations $\mathbf{y}_{1:n-1}$. By using the concept of conditional mutual information, we will explicitly see what the role of the measurements is in the formulation of $P(\mathbf{x}_n|\mathbf{y}_{1:M})$.

Note that this is a different definition from what is usually called the conditional mutual information: $\sum_{x,y,z} P(x,y,z)\log(\frac{P(x,y|z)}{P(x|z)P(y|z)})$ which is the conditional mutual information averaged of over all possible states [10].

We can use the definitions of conditional probability and conditional mutual information, Eq. (2.5), to rewrite the term in square brackets in Eq. (2.4) as [3]

$$\begin{aligned}\frac{P(\mathbf{y}_n|\mathbf{x}_n, \mathbf{y}_{1:n-1})}{P(\mathbf{y}_n|\mathbf{y}_{1:n-1})} &= \frac{P(\mathbf{x}_n, \mathbf{y}_n|\mathbf{y}_{1:n-1})}{P(\mathbf{x}_n|\mathbf{y}_{1:n-1})P(\mathbf{y}_n|\mathbf{y}_{1:n-1})} \\ &= \exp\left[\mathrm{MI}(\mathbf{x}_n, \mathbf{y}_n|\mathbf{y}_{1:n-1})\right].\end{aligned}$$

We can now rewrite Eq. (2.4) using the concept of conditional mutual information

$$P(\mathbf{x}_{0:M}|\mathbf{y}_{1:M}) = \prod_{n=1}^{M} \exp\left[\mathrm{MI}(\mathbf{x}_n, \mathbf{y}_n|\mathbf{y}_{1:n-1})\right] P(\mathbf{x}_n|\mathbf{x}_{n-1})P(\mathbf{x}_0). \tag{2.6}$$

With the conditional path distribution written in this way, we see explicitly how the measurements provide information to modify $P(\mathbf{x}_{0:M}|\mathbf{y}_{1:M})$.

It is useful to do some manipulations to illuminate the role of mutual information. From the definition of mutual information and from the fact that $P(\mathbf{y}_{1:n}) = P(\mathbf{y}_n, \mathbf{y}_{1:n-1})$ we get

$$\begin{aligned}\mathrm{MI}(\mathbf{x}_n, \mathbf{y}_{1:n}) &= \log\left[\frac{P(\mathbf{x}_n, \mathbf{y}_{1:n})}{P(\mathbf{x}_n)P(\mathbf{y}_{1:n})}\right] \\ &= \log\left[\frac{P(\mathbf{x}_n, \mathbf{y}_n, \mathbf{y}_{1:n-1})}{P(\mathbf{x}_n)P(\mathbf{y}_n, \mathbf{y}_{1:n-1})}\right].\end{aligned}$$

Again from the definition of mutual information we have

$$\mathrm{MI}(\mathbf{x}_n, \mathbf{y}_{1:n-1}) = \log\left[\frac{P(\mathbf{x}_n, \mathbf{y}_{1:n-1})}{P(\mathbf{x}_n)P(\mathbf{y}_{1:n-1})}\right].$$

The previous two equations can be combined to find

$$\begin{aligned}\mathrm{MI}(\mathbf{x}_n, \mathbf{y}_{1:n}) - \mathrm{MI}(\mathbf{x}_n, \mathbf{y}_{1:n-1}) &= \log\left[\frac{P(\mathbf{x}_n, \mathbf{y}_n|\mathbf{y}_{1:n-1})}{P(\mathbf{x}_n|\mathbf{y}_{1:n-1})P(\mathbf{y}_n|\mathbf{y}_{1:n-1})}\right] \\ &= \mathrm{MI}(\mathbf{x}_n, \mathbf{y}_n|\mathbf{y}_{1:n-1}),\end{aligned} \tag{2.7}$$

by using the definition of conditional probability. This tells us that $\mathrm{MI}(\mathbf{x}_n, \mathbf{y}_n|\mathbf{y}_{1:n-1})$ can be interpreted as the change in mutual information due to making one additional observation $\mathbf{y}_n$.

We can also use the concept of relative entropy also known as the Kullback-Leibler divergence. The relative entropy between two distributions P and Q is defined as [10]

$$D_{KL}(P||Q) = \int dx P(x) \log \frac{P(x)}{Q(x)}.$$

It is a theorem that $D_{KL}(P||Q) \geq 0$ with equality only when $P = Q$. We can calculate the relative entropy as a way of comparing the distributions $P(\mathbf{x}_n|\mathbf{y}_{1:n})$ and $P(\mathbf{x}_n|\mathbf{y}_{1:n-1})$ using Eq. (2.1). We get

$$D_{KL}(P(\mathbf{x}_n|\mathbf{y}_{1:n})||P(\mathbf{x}_n|\mathbf{y}_{1:n-1})) = \int d\mathbf{x}_n P(\mathbf{x}_n|\mathbf{y}_{1:n})\mathrm{MI}(\mathbf{x}_n, \mathbf{y}_n|\mathbf{y}_{1:n-1}).$$

We see that if $\mathbf{y}_n$ and $\mathbf{x}_n$ are independent ($P(\mathbf{x}_n, \mathbf{y}_n) = P(\mathbf{x}_n)P(\mathbf{y}_n)$), then the mutual information term is zero, and the relative entropy between $P(\mathbf{x}_n|\mathbf{y}_{1:n})$ and $P(\mathbf{x}_n|\mathbf{y}_{1:n-1})$ is zero. This means that the two distributions are the same– clearly nothing is learned from the measurement $\mathbf{y}_n$ if it is independent of $\mathbf{x}_n$. If the previous equation is averaged over all possible measurement histories $\mathbf{y}_{1:n}$, by multiplying by $P(\mathbf{y}_{1:n})$ and integrating over $\mathbf{y}_{1:n}$, we get

$$\int d\mathbf{x}_n d\mathbf{y}_{1:n} P(\mathbf{x}_n, \mathbf{y}_{1:n})\mathrm{MI}(\mathbf{x}_n, \mathbf{y}_n|\mathbf{y}_{1:n-1}),$$

which is the average conditional mutual information.

2.1.2 Uncorrelated Measurement Errors

If we assume that the measurement errors are uncorrelated in time, then we can say that measurement at time n only depends on the state at time n and not on the previous measurement history, so $P(\mathbf{y}_n|\mathbf{x}_n, \mathbf{y}_{1:n-1}) = P(\mathbf{y}_n|\mathbf{x}_n)$. With this we can rewrite the term in square brackets in Eq. (2.4) as

$$\exp\left[\mathrm{MI}(\mathbf{x}_n, \mathbf{y}_n|\mathbf{y}_{1:n-1})\right] = \frac{P(\mathbf{y}_n|\mathbf{x}_n, \mathbf{y}_{1:n-1})}{P(\mathbf{y}_n|\mathbf{y}_{1:n-1})} \propto P(\mathbf{y}_n|\mathbf{x}_n). \tag{2.8}$$

We drop the normalization factor in the denominator, because it is not a function of the path, and we do not need to keep track of normalization constants, as we will see later. It is not necessary to make this assumption, but it is a convenient simplification. This will not be a good assumption if the autocorrelation time of the measurement noise is larger than the data sampling rate.

The term $P(\mathbf{y}_n|\mathbf{x}_n)d\mathbf{y}_n$ is the probability of a measurement being in the range $\mathbf{y}_n$ to $\mathbf{y}_n + d\mathbf{y}_n$ given that the state is $\mathbf{x}_n$. The particular form of this probability depends on the noise inherent in the measurement process which we will discuss later. We can now express the conditional probability Eq. (2.4) as

$$P(\mathbf{X}|\mathbf{Y}) \propto \prod_{n=1}^{M} P(\mathbf{y}_n|\mathbf{x}_n)P(\mathbf{x}_n|\mathbf{x}_{n-1})P(\mathbf{x}_0). \tag{2.9}$$

The distribution of initial states $P(\mathbf{x}_0)$ is propagated forward by the dynamical transition probabilities $P(\mathbf{x}_n|\mathbf{x}_{n-1})$ and also modified by the measurement terms $P(\mathbf{y}_n|\mathbf{x}_n)$. A schematic representation of this equation is given in Fig. 2.1. We will examine each of these terms in more detail in later sections of this chapter.

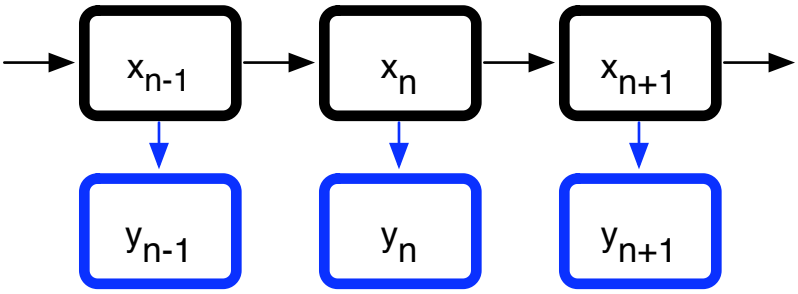

Figure 2.1: Schematic description of the formula for $P(\mathbf{X}|\mathbf{Y})$, with the assumption of Markov dynamics and the assumption that measurements only depend on the current state. Each black arrow represents a transition probability $P(\mathbf{x}_n|\mathbf{x}_{n-1})$ and each blue arrow represents measurement probability $P(\mathbf{y}_n|\mathbf{x}_n)$. The conditional path probability distribution $P(\mathbf{X}|\mathbf{Y})$ is a product of all these terms and $P(\mathbf{x}_0)$.

2.1.3 The Action

We can then use Eq. (2.6) to write down the conditional probability as a function of the entire path $\mathbf{X} = \mathbf{x}_{0:M}$ conditioned on the entire time series of observations $\mathbf{Y} = \mathbf{y}_{1:M}$ as

$$P(\mathbf{X}|\mathbf{Y}) \propto \exp[-A_0(\mathbf{X}, \mathbf{Y})], \tag{2.10}$$

by defining the action as

$$\begin{aligned} A_0(\mathbf{X},\mathbf{Y}) = &-\sum_{n=1}^{M} \mathrm{MI}(\mathbf{x}_n,\mathbf{y}_n|\mathbf{y}_{1:n-1}) \\ &-\sum_{n=1}^{M} \log P(\mathbf{x}_n|\mathbf{x}_{n-1}) \\ &-\log P(\mathbf{x}_0). \end{aligned} \tag{2.11}$$

We are free to drop additive constants to $A_0(\mathbf{X},\mathbf{Y})$, because we defined $\exp[-A_0(\mathbf{X},\mathbf{Y})]$ to be proportional to $P(\mathbf{X}|\mathbf{Y})$ but not necessarily equal. We will now think of the action as a function of the path $\mathbf{X}$ given a fixed time series of observations $\mathbf{Y}$, and so we will refer to it as $A_0(\mathbf{X})$.

2.2 Conditional Expectation Values as Path Integrals

Now that we have $P(\mathbf{X}|\mathbf{Y})$, we can calculate conditional expectation values of any function of the path $\chi(\mathbf{X})$ by using the definition of expectation value,

$$\begin{aligned} \langle \chi(\mathbf{X}) \rangle &= \int d\mathbf{X}\ \chi(\mathbf{X}) P(\mathbf{X}|\mathbf{Y}) \\ &= \frac{\int d\mathbf{X}\ \chi(\mathbf{X}) \exp[-A_0(\mathbf{X})]}{\int d\mathbf{X}\ \exp[-A_0(\mathbf{X})]}, \end{aligned} \tag{2.12}$$

where $\int d\mathbf{X}$ means integrate over all paths. This equation says to find the expected value of $\chi(\mathbf{X})$ we must average over all possible paths with each path weighted by a factor $\exp[-A_0(\mathbf{X})]$. This is very difficult to do in a direct fashion since the dimension of $\mathbf{X}$ is typically very high, and so the volume that needs to be integrated over is very large. We will see in Chapter 3 that there are better ways of doing this integral which take advantage of the fact that $\exp[-A_0(\mathbf{X})]$ is typically essentially zero everywhere except in localized regions of path space. The idea is to use a large number of sample paths generated by a Markov chain Monte Carlo (MCMC) method, which will be more concentrated in regions of higher probability, to approximate the distribution $P(\mathbf{X}|\mathbf{Y})$. We can then use the sample paths to approximate properties of $P(\mathbf{X}|\mathbf{Y})$ such as moments of the path components.

Equation (2.12) together with definition of the action Eq. (2.11) allow us, at least in principle, to calculate the expectation values of any arbitrary function of the path $\chi(\mathbf{X})$, conditioned by the entire sequence of measurements $\mathbf{Y}$. Why is this a useful thing to do?

One of our goals is to find a value $\mu_{n,l}$ to serve as the 'best' estimate of the state $x_{n,l}$. In this probabilistic setting, $x_{n,l}$ is thought of as a random variable that is drawn from the marginal distribution $P(x_{n,l}|\mathbf{Y})$, which is $P(\mathbf{X}|\mathbf{Y})$ integrated over all states except $x_{n,l}$. If we consider the squared error between the estimate and the state, $d_{n,l}^2 = (\mu_{n,l} - x_{n,l})^2$, and calculate its expectation we find

$$\left\langle d_{n,l}^2 \right\rangle = (\mu_{n,l} - \langle x_{n,l} \rangle)^2.$$

We see that if our criterion is to minimize the expectation value of the error, then our best estimate will be $\mu_{n,l} = \langle x_{n,l} \rangle$, the minimum variance estimate [34, 58]. This is the motivation for calculating the first moment of each component of the path. This can be accomplished by choosing $\chi(\mathbf{X}) = x_{n,l}$ in Eq.(2.12), for $n = 0, 1, \ldots, M$ and $l = 1, 2, \ldots, D$.

We would also like to know how accurate the state estimate $\mu_{n,l}$ is. This can be quantified by calculating the the expectation value of $(x_{n,l} - \mu_{n,l})^2$,

$$\left\langle (x_{n,l} - \mu_{n,l})^2 \right\rangle = \left\langle x_{n,l}^2 \right\rangle - \langle x_{n,l} \rangle^2 .$$

This tells us that we should also compute the second moment of each component in the path by choosing $\chi(\mathbf{X}) = x_{n,l}^2$. We define the RMS variation about the mean as

$$\sigma_{n,l} = \sqrt{\langle (x_{n,l} - \langle x_{n,l} \rangle)^2 \rangle},$$

and use it to quantify the accuracy of the state estimate. We will also see in Chapter 3 that higher moments of the components of the path can be useful to compute.

2.3 Approximations to the Action

We now discus each term in the action Eq. (2.11) in more detail, and show approximations that can be made to get the action in a simple and useful form.

2.3.1 Distribution of Initial Conditions

The distribution $P(\mathbf{x}_0)$ represents the distribution of initial states, and parameters since we are counting parameters as time-independent states. We are free to choose this distribution in any reasonable way. It should be based on knowledge of the system being studied, which may allow us to constrain the initial states and parameters to be

within certain ranges. This information could be found from previous experiments, or from theoretical reasoning. The simplest, although uninformative, choice that we will make is to assume a uniform distribution, so $P(\mathbf{x}_0)$ is a constant and can be neglected. We are free to ignore any constant added to $A_0(\mathbf{X})$, because they become multipliers to $\exp[-A_0(\mathbf{X})]$ which appear in the numerator and denominator in Eq. (2.12). Another possibility is to use the output of a previous data assimilation which ended at time $t = 0$ as the $P(\mathbf{x}_0)$.

2.3.2 Transition Probabilities 1

The Markov transition probability $P(\mathbf{x}_{n+1}|\mathbf{x}_n)$ represents the probability of transitioning from state $\mathbf{x}_n$ at time step n to state $\mathbf{x}_{n+1}$ at time step $n+1$. This is exactly what a Markov model of the system will tell us. The model may come in many different forms, but here we will focus on how to find $P(\mathbf{x}_{n+1}|\mathbf{x}_n)$ from a model in the form of a system of first order ordinary differential equations

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{F}(\mathbf{x}(t)) + \boldsymbol{\eta}(t), \tag{2.13}$$

where $\boldsymbol{\eta}(t)$ is a noise term. We discretize this equation in time using the implicit rule

$$\frac{\mathbf{x}_{n+1} - \mathbf{x}_n}{\Delta t} = \frac{\mathbf{F}(\mathbf{x}_{n+1}) + \mathbf{F}(\mathbf{x}_n)}{2} + \boldsymbol{\eta}(\frac{t_n + t_{n+1}}{2}).$$

This method is an average of forward (explicit) and backward (implicit) Euler integration rules, and it is accurate to order Δt^2. It was shown in [52, 27] that this way of discretizing leads to the correct form of the path integral in the limit of continuous time. This discretization scheme is known as the second order Adams-Moulton or the trapezoidal rule, and is an implicit method. We can write this as

$$g_l(\mathbf{x}_n, \mathbf{x}_{n+1}) = \Delta t \; \eta_l(\frac{t_n + t_{n+1}}{2}),$$

where we defined

$$g_l(\mathbf{x}_n, \mathbf{x}_{n+1}) = x_{n+1,l} - x_{n,l} - \frac{\Delta t}{2}\left[F_l(\mathbf{x}_{n+1}) + F_l(\mathbf{x}_n)\right]. \tag{2.14}$$

In the case of a deterministic model the transition probabilities are

$$P(\mathbf{x}_{n+1}|\mathbf{x}_n) = \delta^D(\mathbf{g}(\mathbf{x}_n, \mathbf{x}_{n+1})). \tag{2.15}$$

This says that if the state $\mathbf{x}_n$ is known, then the state $\mathbf{x}_{n+1}$ is also known with certainty. We can modify this to allow stochastic dynamics by broadening the delta function. There are several ways to do this, but here we use a Gaussian distribution,

$$\delta^D(\mathbf{g}(\mathbf{x}_n, \mathbf{x}_{n+1}) \to \sqrt{\frac{R_f}{(2\pi)^D}} \exp\left[-\frac{R_f}{2}\mathbf{g}(\mathbf{x}_n, \mathbf{x}_{n+1})^2\right], \tag{2.16}$$

where $R_f = \sigma_f^{-2}$, and σ_f is a parameter that sets the width of the distribution, and thus the dynamical noise level. As $\sigma_f \to 0$ we get back the delta function, and deterministic dynamics. Putting it all together, the transition probability is

$$P(\mathbf{x}_{n+1}|\mathbf{x}_n) = \sqrt{\frac{R_f}{(2\pi)^D}} \prod_{l=1}^{D} \exp\left[-\frac{R_f}{2} g_l(\mathbf{x}_n, \mathbf{x}_{n+1})^2\right] \tag{2.17}$$

2.3.3 Transition Probabilities 2

We now arrive at this same result for transition probabilities, $P(\mathbf{x}_n|\mathbf{x}_{n-1})$, in a different way which clarifies the meaning of R_f, and makes the approximations more apparent. The type of model we consider here is expressed as a system of stochastic differential equations with additive noise terms. We further assume that the noise is Gaussian and uncorrelated in time and among the different components of the state.

Consider a one dimensional model of the form

$$\frac{dx}{dt} = V(x(t)) + \eta(t), \tag{2.18}$$

where $\eta(t)$ is a Gaussian noise term with $\langle\eta(t)\rangle = 0$ and $\langle\eta(t)\eta(t')\rangle = 2\Gamma\delta(t-t')$. If we have an ensemble of particles with some initial distribution $f(x, t=0)$, and each one is evolved in time by the Langévin equation Eq. (2.18), then the time evolution of the distribution will be given by the Fokker-Plank equation [29]

$$\frac{\partial f(x,t)}{\partial t} = -\frac{\partial}{\partial x}(V(x)f(x,t)) + \Gamma\frac{\partial^2 f(x,t)}{\partial x^2}. \tag{2.19}$$

We would like to find the transition probability $P(x_{n+1}|x_n)$ which says how the distribution evolves from time t_n to time $t_{n+1} = t_n + \Delta t$. To approximate this transition probability in a time symmetric fashion, we divided the time step into two equal pieces. The intermediate time is called $t_h = t_n + \tau$ where $\tau = \Delta t/2$. We divide the transition into a forward part from (x_n, t_n) to (x_h, t_h) and a backwards part from (x_{n+1}, t_{n+1}) to (x_h, t_h), and integrate over all possible intermediate positions x_h. We are finding the approximate propagator for particle to go from (x_n, t_n) to (x_{n+1}, t_{n+1}) when the particle's motion is prescribed by Eq. (2.18).

First the forward transition is

$$f_+(x_h, t = t_h) = \int dx' P_\tau(x_h|x') f(x', t = t_n)$$
$$= P_\tau(x_h|x_n).$$

The second equality is true if we set the initial distribution to be $f(x', t = t_n) = \delta(x' - x_n)$. If we make the approximation that $V(x) \approx V(x_n)$, we can write down the solution to Eq. (2.19) for the appropriate delta function initial condition:

$$P_\tau(x_h|x_n) = \sqrt{\frac{1}{4\pi\Gamma\tau}} \exp\left[-\frac{(x_h - x_n - \tau V(x_n))^2}{4\Gamma\tau}\right].$$

And similarly for the backwards transition with the final condition $f(x', t = t_{n+1}) = \delta(x' - x_{n+1})$

$$f_-(x_h, t = t_h) = \int dx' P_{-\tau}(x_h|x') f(x', t = t_{n+1})$$
$$= P_{-\tau}(x_h|x_{n+1}).$$

With the approximation that $V(x) \approx V(x_{n+1})$ the solution for the backwards transition is

$$P_{-\tau}(x_h|x_{n+1}) = \sqrt{\frac{1}{4\pi\Gamma\tau}} \exp\left[-\frac{(x_h - x_{n+1} + \tau V(x_{n+1}))^2}{4\Gamma\tau}\right]$$

Then integrate the product of forwards and backwards transitions over x_h

$$P(x_{n+1}|x_n) = \int dx_h P_\tau(x_h|x_n) P_{-\tau}(x_h|x_{n+1})$$
$$= \sqrt{\frac{1}{8\pi\Gamma\tau}} \exp\left[-\frac{(x_{n+1} - x_n - \tau V(x_n) - \tau V(x_{n+1}))^2}{8\Gamma\tau}\right]. \quad (2.20)$$

The expression on the right comes from expanding the terms in the exponentials, completing the square for x_h, doing the Gaussian integral over x_h, and then factoring the result. In D dimensions this generalizes to the same formula as before for the transition probabilities, Eq. (2.17), if we set $R_f = 1/(2\Gamma\Delta t)$ and remember that $\tau = \Delta t/2$.

The crucial approximation that $V(x) \approx V(x_n)$ is valid when τ is small enough so that $V(x)$ does not change much over the region of significant probability in one time step. The approximation is valid when

$$\frac{dV(x_n)}{dx}\left(\tau V(x_n) + \sqrt{4\Gamma\tau}\right) \ll V(x_n).$$

2.3.4 Mutual Information

We again make the assumption that measurement errors are uncorrelated in time, so the mutual information term simplifies as shown in Eq. (2.8). We can then make assumptions about the form of the measurement noise. Here we assume

$$P(\mathbf{y}_n|\mathbf{x}_n) = P_{noise}(|y_{n,1} - x_{n,1}|, |y_{n,2} - x_{n,2}|, \ldots, |y_{n,L} - x_{n,L}|).$$

We further assume that the noise has a multivariate Gaussian distribution

$$P(\mathbf{y}_n|\mathbf{x}_n) \propto \exp\left[-\frac{1}{2}\sum_{l,l'=1}^{L}(y_{n,l} - x_{n,l})(\mathbf{R_m})_{ll'}(y_{n,l'} - x_{n,l'})\right],$$

where $\mathbf{R_m}$ is the inverse of the covariance matrix, a symmetric positive-definite $L \times L$ matrix which characterizes the the measurement noise distribution. If the measurements of each component of $\mathbf{x}_n$ are independent, $\mathbf{R_m}$ is a diagonal matrix. For illustration purposes we assume that this is the case, and further assume that each measurement has the same noise magnitude, so $\mathbf{R_m} = \mathbf{I}R_m$, where R_m is a scalar. With these assumptions the total mutual information (neglecting additive constants) becomes

$$\sum_{n=1}^{M} \mathrm{MI}(\mathbf{x}_n, \mathbf{y}_n) = -\frac{R_m}{2}\sum_{n=1}^{M}\sum_{l=1}^{L}(y_{n,l} - x_{n,l})^2. \tag{2.21}$$

This has a straightforward interpretation as a least squares cost function comparing the model output to the data. The closer to zero this term is, the more consistent the path $\mathbf{X}$ is with the measurements $\mathbf{Y}$.

2.4 Illustrative Examples

It is instructive to pause at this point and work out two simple examples that can be solved analytically.

2.4.1 Repeated Measurements of Constant

Suppose we take repeated measurements of some quantity that we believe is constant in time. To connect to the formalism presented so far, we model this as a dynamical system with one state variable $x(t)$ having the trivial differential equation $\frac{dx}{dt} = 0$. If we assume the model is deterministic, so that $x(t)$ is exactly constant, then

the transition probabilities are simply

$$P(x_{n+1}|x_n) = \delta(x_{n+1} - x_n).$$

For illustration we assume that two uncorrelated measurements are taken at two different times, one of x_1 and one of x_2, called y_1 and y_2 respectively. From Eq. (2.9) integrated over x_0 we get

$$P(x_1, x_2|y_1, y_2) \propto \int dx_0 P(y_2|x_2)P(y_1|x_1)P(x_2|x_1)P(x_1|x_0)P(x_0). \tag{2.22}$$

We can use this to find an estimate of the true state x and an uncertainty in that estimate. To do this we calculate

$$\langle x_1 \rangle = \langle x_2 \rangle = \frac{1}{Z} \int dx_1 dx_2 x_2 P(x_1, x_2|y_1, y_2),$$
$$\left\langle x_1^2 \right\rangle = \left\langle x_2^2 \right\rangle = \frac{1}{Z} \int dx_1 dx_2 x_2^2 P(x_1, x_2|y_1, y_2),$$

where Z is the normalization factor. We also assume the measurements have Gaussian noise distributions given by

$$P(y_n|x_n) \propto \exp\left[-\frac{R_n}{2}(x_n - y_n)^2\right],$$

where we allow the uncertainties $\sigma_n = R_n^{-1/2}$ to be different for each measurement. For simplicity we also assume that $P(x_0)$ is uniform and so it can be pulled out of the integrals and canceled from the numerator and denominator of the expectation values.

Plugging in the transition probability and the measurement terms into Eq. (2.22), we do all but one of the integrals and eliminate the delta functions. Then complete the square and do the Gaussian integrals. The result can be generalized to many measurements and get the familiar results

$$\langle x \rangle = \frac{\sum_n R_n y_n}{\sum_n R_n}$$
$$\sigma^2 = \left\langle x^2 \right\rangle - \langle x \rangle^2 = \frac{1}{\sum_n R_n}.$$

The expected value of x is a weighted average of the measurements with more weight given to the measurements that have less uncertainty. The uncertainty σ decreases as more measurements are made like $1/\sqrt{N}$, where N is the number of measurements of the same quantity.

2.4.2 Brownian Motion

Consider the motion of a particle whose position is given by

$$\frac{dx}{dt} = \eta(t), \tag{2.23}$$

where $\eta(t)$ is a Gaussian white noise term with $\langle \eta(t) \rangle = 0$ and $\langle \eta(t)\eta(t') \rangle = 2\Gamma\delta(t-t')$. We assume that no measurements are available, but that we exactly know the initial position is at $x_0 = 0$. This means that the initial distribution is $P(x_0) = \delta(x_0)$. We can find the transition probability from Eqs. (2.20) by plugging in $V(x) = 0$:

$$P(x_{n+1}|x_n) = \sqrt{\frac{1}{4\pi\Gamma\Delta t}} \exp\left[-\frac{(x_{n+1}-x_n)^2}{4\Gamma\Delta t}\right].$$

We then use the path distribution from Eq. (2.9), but in this case there are no measurement terms

$$P(x_{0:n}) \propto \prod_{j=1}^{n} P(x_j|x_{j-1})P(x_0).$$

If we integrate over all state variables except the last one x_n, we get the probability distribution of the particle's position at time n

$$P(x_n) \propto \int dx_0 \dots dx_{n-1} P(x_0) \exp\left[-\frac{(x_1-x_0)^2}{4\Gamma\Delta t}\right] \dots \exp\left[-\frac{(x_n-x_{n-1})^2}{4\Gamma\Delta t}\right].$$

We can do the integrals starting with x_0 to eliminate the delta function. Then complete the square to get the term involving x_1 in the form of

$$\exp\left[-\frac{(x_1-\frac{1}{2}x_2)^2}{2\Gamma\Delta t}\right] \exp\left[-\frac{x_2^2}{8\Gamma\Delta t}\right],$$

and then do the Gaussian integral over x_1. Repeat this for each integral up to x_{n-1}. We are then left with

$$P(x_n) \propto \exp\left[-\frac{x_n^2}{4\Gamma n\Delta t}\right].$$

If we identify $n = t/\Delta t$ we get the usual solution of the simple diffusion equation $\frac{\partial P(x,t)}{\partial t} = \Gamma\frac{\partial^2 P(x,t)}{\partial x^2}$, with $P(x,0) = \delta(x)$, an expanding Gaussian

$$P(x,t) = \sqrt{\frac{1}{4\pi\Gamma t}} \exp\left[-\frac{x^2}{4\Gamma t}\right].$$

As $t \to 0$ we get back the initial delta function distribution. This equation gives the density of an ensemble of particles all initially at $x = 0$ and diffusing according to Eq. (2.23).

This example shows how the Markov transition probability $P(x_n|x_{n-1})$ evolves the distribution forward in time. In this simple case $P(x_n)$ simply becomes wider and it remains centered at zero, because the RHS of the differential equation averages to zero. Without any measurements, our knowledge of the position of the particle decreases in time.

We could also imagine doing an experiment where we take occasional noisy measurements of the position of the particle, still assuming the motion of the particle is modeled by Eq. (2.23). Then the probability distribution for the particle's position at time n, $P(x_n|y_{1:n})$, would include a term for each measurement $P(y_n|x_n)$ in addition to the transition probability terms shown above. The measurement terms would modify $P(x_n|y_{1:n})$ so that it no longer uniformly expands in time, but instead is guided by what we learn from the measurements. Each measurement provides information about the particle's position, and so it has the ability to narrow the distribution $P(x_n|y_{1:n})$, letting us make a more accurate estimate of the position of the particle $\langle x_n \rangle$.

2.5 Summary

We have seen how the conditional probability distribution in path space can be expressed in terms of the action using $P(\mathbf{X}|\mathbf{Y}) \propto \exp[-A_0(\mathbf{X})]$. This allows us to calculate conditional expectation values of any function of the path with the path integral in Eq. (2.12). With all the simplifying assumptions made in the previous section, the action (neglecting additive constants) becomes

$$\begin{aligned} A_0(\mathbf{X}) &= \frac{R_m}{2} \sum_{n=1}^{M} \sum_{l=1}^{L} (x_{n,l} - y_{n,l})^2 \\ &+ \frac{R_f}{2} \sum_{n=1}^{M} \sum_{l=1}^{D} g_l(\mathbf{x}_{n-1}, \mathbf{x}_n)^2, \end{aligned}$$

where $g_l(\mathbf{x}_{n-1}, \mathbf{x}_n)$ is defined in Eq. (2.14), and represents the amount of deviation from the trajectory predicted by the deterministic model, $\mathbf{F}(\mathbf{x})$. The first sum becomes large for paths that are inconsistent with the data. The second sum becomes large for paths that are inconsistent with the deterministic model. The smallest value the action can take on is zero, and that can only happen if there is no noise in the measurements, and the deterministic model is exactly describes the system evolution. We have assumed that there are observations available at every time step for notational convenience, but any observations that are missing can simply be excluded from the first sum.

In this simplest case, we have two constants R_m and R_f which set the importance of the measurements and the importance of the model, respectively. In a real experiment there will be some estimate of the uncertainty in the measurements σ_m, and so R_m can be set using $R_m = \sigma_m^{-2}$. The other term, R_f, characterizes the noise or uncertainty in the model, and is harder to set *a priori*, since we may not know how accurately the model describes the dynamics. We will see what the effect of this setting is in later chapters, and show how it can be varied to test the accuracy of the model.

Chapter 3

Monte Carlo Evaluation of Path Integrals

3.1 Introduction to Monte Carlo Sampling

We saw in Chapter 2 that the process of doing data assimilation can be reduced to doing path integrals of the form

$$\begin{aligned}\langle \chi(\mathbf{X}) \rangle &= \int d\mathbf{X}\ \chi(\mathbf{X}) P(\mathbf{X}|\mathbf{Y}) \\ &= \frac{\int d\mathbf{X}\ \chi(\mathbf{X}) \exp[-A_0(\mathbf{X})]}{\int d\mathbf{X}\ \exp[-A_0(\mathbf{X})]},\end{aligned} \tag{3.1}$$

where $A_0(\mathbf{X})$ is the action, and $\chi(\mathbf{X})$ is some arbitrary function of the path. The action is a positive function of the path $\mathbf{X}$ which depends on the entire time series of measurements, $\mathbf{Y}$, as well as a model of the dynamical system, and is defined in the previous chapter. The integral is over every possible path, each which represents a possible time evolution of the D-dimensional state variable, consistent with the model or not. Each possible path is weighted by the factor $\exp[-A_0(\mathbf{X})]$ which represents how likely the path is, which depends on how close the path is to the data and how close the path is to what is predicted by the model. Since time is discretized into M steps, and the state vector has dimension D, the dimension of the integral is $N = (M+1) \times D$ which is typically large.

One approach to approximating the integral is to divide path space into cells. To do this, we would have to pick a finite region of path space where the contribution to the integral is non-negligible, say of size x_L in each of the N directions, and pick a grid

resolution Δx. We could then evaluate the integrand at each of these grid points and sum the results. The trouble with this naive approach is that the number of grid points would be $(x_L/\Delta x)^N$ which is a huge number since N is large. This would be impractical. It is also wasteful, because in the cases of interest where the model and measurements provide useful information, most of the huge number of possible paths are extremely unlikely. In a well-defined data assimilation problem there should be a relatively small region of path space that has non-negligible contribution to the integral, and these are the paths we are interested in.

The goal of this chapter is to discuss and demonstrate ways of actually numerically evaluating the path integrals. The key idea is that instead of trying to uniformly sample all possible paths, we sample the paths based on their importance. The most important paths are the ones where the weight factor $\exp[-A_0(\mathbf{X})]$ comes out largest, which is where $A_0(\mathbf{X})$ is smallest.

If we could generate a series of paths $\{\mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(J)}\}$ that are distributed in path space according to $P(\mathbf{X}|\mathbf{Y}) \propto \exp[-A_0(\mathbf{X})]$, then we can use those paths to approximate the distribution with

$$P(\mathbf{X}|\mathbf{Y}) \approx \frac{1}{J}\sum_{j=1}^{J} \delta(\mathbf{X} - \mathbf{X}^{(j)}).$$

We can then calculate expectation values of any function of the path with

$$\begin{aligned} \langle \chi(\mathbf{X}) \rangle &= \int d\mathbf{X}\, \chi(\mathbf{X}) P(\mathbf{X}|\mathbf{Y}) \\ &\approx \frac{1}{J}\sum_{j=1}^{J} \chi(\mathbf{X}^{(j)}). \end{aligned} \tag{3.2}$$

This approach has the benefit that we do not need to sample all of the path space, but only the regions of path space that have a significant contribution to the path integral. Another benefit is that we do not need to explicitly calculate the normalization factor in the denominator of the conditional expectation value Eq. (3.1). The paths generated by this process can be thought of as representing many possible time evolutions of the system state that could have produced the observed data when measured. The paths that are more likely to have produced the observed data will be generated more times than the paths which are less likely to have produced the observed data.

Now all that is left to do is to find a way to generate a series of paths that are distributed according to $\exp[-A_0(\mathbf{X})]$. There are several path integral Monte Carlo methods, such as Metropolis or Hybrid Monte Carlo, that do exactly this [38, 42, 4, 55, 6].

3.2 Path Integral Monte Carlo Methods

3.2.1 Metropolis Monte Carlo

One of the simplest and oldest approaches is the Metropolis Monte Carlo method [41]. This method works by generating a sequence of paths $\{\mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(J)}\}$ that we will call Monte Carlo paths by a random walk through path space. The method is an example of a Markov chain Monte Carlo method, because the the paths are generated in sequence and the next path is generated only from the current path in a stochastic way. The random walk is biased in a particular way as described below so that the sequence of paths that are generated come out distributed according to $\exp[-A_0(\mathbf{X})]$.

The method works by generating a new path $\mathbf{X}^{(n+1)}$ from the current path $\mathbf{X}^{(n)}$ with a two step procedure. First a candidate path $\mathbf{X}'$ is selected by adding an unbiased random displacement to the current path $\mathbf{X} = \mathbf{X}^{(n)}$. The displacement may be to only one component or all the components, and the random number may be drawn from any type of distribution, as long as it is unbiased: $\mathbf{X} \to \mathbf{X}'$ is as likely to occur as $\mathbf{X}' \to \mathbf{X}$.

Next the candidate path is either accepted ($\mathbf{X}^{(n+1)} = \mathbf{X}'$) or rejected ($\mathbf{X}^{(n+1)} = \mathbf{X}$). The probability for acceptance is

$$P_{accept}(\mathbf{X}', \mathbf{X}) = \min(1, \exp[-\Delta A_0(\mathbf{X}', \mathbf{X})]), \tag{3.3}$$

where $\Delta A_0(\mathbf{X}', \mathbf{X}) = A_0(\mathbf{X}') - A_0(\mathbf{X})$ is the change in action which would be caused by changing $\mathbf{X}$ to $\mathbf{X}'$. This says that if a proposed change will lower the action it should be accepted, and if the proposed change increases the action it should only be accepted with probability $\exp[-\Delta A_0(\mathbf{X}', \mathbf{X})]$. Note that only the change in action is required, so the full action never needs to be computed. This means we do not need to keep track of additive constants to the action, and can save a lot of computation time by only computing the terms in the action that will be changed by the update.

Now we will see why this Metropolis path update rule will generate paths distributed in the appropriate way. If we start with an ensemble of paths with density $\rho_s(\mathbf{X}) \propto \exp[-A_0(\mathbf{X})]$ and update all the paths according to the Metropolis rule the density of the ensemble will be invariant. This is because the detailed balance condition

$$\rho_s(\mathbf{X}) P(\mathbf{X} \to \mathbf{X}') = \rho_s(\mathbf{X}') P(\mathbf{X}' \to \mathbf{X})$$

is satisfied for the stationary density $\rho_s(\mathbf{X}) \propto \exp[-A_0(\mathbf{X})]$. The Metropolis method

defines the transition probability to be

$$P(\mathbf{X} \to \mathbf{X}') = P_{can}(|\mathbf{X}' - \mathbf{X}|) \min(1, \exp[-\Delta A_0(\mathbf{X}', \mathbf{X})]),$$

where $P_{can}(|\mathbf{X}' - \mathbf{X}|)$ is the candidate selection probability, and the second term is the acceptance probability as defined in Eq. (3.3). It is straightforward to show that the stationary distribution $\rho_s(\mathbf{X}) \propto \exp[-A_0(\mathbf{X})]$ with the Metropolis update rule satisfies the detailed balance condition.

It can also be shown that any initial distribution of paths that are updated according to the Metropolis rule will approach the stationary distribution [41].

We can then use the sequence of paths generated by the Metropolis procedure to approximate the distribution $P(\mathbf{X}|\mathbf{Y})$ and from this calculate conditional expectation values of any function of the path as shown in Eq. (3.2).

3.2.2 Error Estimates

It is important to be able to estimate how accurately the Monte Carlo calculation approximates the true value of the path integral. We would like to estimate the error in the MC estimate of some quantity $z_j = \chi(\mathbf{X}^{(j)})$ defined by some function of the path $\chi(\mathbf{X})$. The MC estimate is the average over paths

$$\bar{z} = \frac{1}{J} \sum_{j=1}^{J} z_j.$$

The variance of this quantity is

$$\sigma_z^2 = \frac{1}{J} \sum_{j=1}^{J} (z_j - \bar{z})^2.$$

The error in the estimate of $\langle z \rangle$ is

$$\Delta z = \bar{z} - \langle z \rangle \approx \sigma_z \sqrt{\frac{J_{ac}}{J}}.$$

The $J^{-1/2}$ dependence comes from the fact that the estimate is based on the sum over many random steps. The J_{ac} factor enters because the paths generated by the MC procedure are not independent, and so the effective number of independent steps is less than J. J_{ac} can be estimated by calculating the number of path updates needed for the autocorrelation function to decay to $1/e$. This is a rough approximation of the number

of updates that must be done to a given path to produce a new path that is independent [14].

The autocorrelation function,

$$f_{ac}(n) = \frac{1}{\sigma_z^2(J-n)} \sum_{j=1}^{J-n} (z_j - \bar{z})(z_{j+n} - \bar{z}), \tag{3.4}$$

is normalized so that $f_{ac}(0) = 1$, and should decay to zero when n becomes large enough. When this happens, enough path updates have been done to wipe out correlations. The autocorrelation function goes roughly as $e^{-n/J_{ac}}$. We will compute this function for a specific example in a later section, as a way of estimating J_{ac}. Another method of error estimation is discussed in Appendix A.

3.3 Monte Carlo Predictions

It is possible to extend the path to include times beyond the end of the observation window, $0 \leq t \leq t_m$, where measurements are not available. The action is formulated in the same way as before, but the unavailable measurements are simply excluded from the mutual information term in the same way that was done for unobserved variables. The path integral can then be done in the same way as before to calculate moments of the path, but now part of the path can be interpreted as a predication.

However, this makes the MC evaluation very difficult if the model is chaotic and if the prediction window is long compared to $1/\lambda_{max}$, where λ_{max} is the largest Lyapunov exponent of the model. Even if the likely states (including parameters as state variables) at $t = t_m$ are confined to a small region of state space, the chaotic nature of the model, without the guiding influence of measurements, will cause the small likely region to spread out and eventually cover the entire attractor. This means very many paths will need to be generated to adequately sample the likely regions of path space. The problem may be even worse when the likely regions of state space form islands of high probability separated by regions of low probability.

A way to deal with this problem is to not include future times in the path, but to generate the predications by directly integrating the model equations forward in time. This can be done by integrating each path that is generated in the MC process forward into the prediction window. This is essentially what is done in the prediction step of particle filter methods [7] (the other step in particle filter methods is the update

step which is how the measurements are incorporated). The integration is done using the states (and parameters) at $t = t_m$ as initial conditions (and parameters). The integration can be done without noise, or with noise added to the integration. In this way we get a large number of trajectories in the prediction window which we can calculate statistics of, typically the mean and standard deviation as a function of time, to make predictions. The trajectories will typically start out well-confined at $t = t_m$ and spread out as time progresses because of chaos and because of the noise in the dynamics (if explicitly added to the integration). At a certain point the mean predictions will not be very useful, but that will be apparent: since we calculate an ensemble of paths instead of just one path we can see the spread in paths.

3.4 Monte Carlo Implementation

Now we discuss the details of the particular path integral Monte Carlo implementation used to do the calculations shown in this chapter. This section contains an overview of the process, more details about the Monte Carlo Data Assimilation code are presented in Appendix A.

We include N_p unknown parameters in the path by treating them as state variables with the differential equation $\dot{p} = 0$. We treat the parameters as exactly constant in time, so we do not add noise terms to the differential equation for the parameters. This means that each parameter can be represented as a single component of the path. Another possibility is to relax the assumption that the parameters are exactly constant in time, and to treat them in exactly the same way as the state variables. This increases the dimension of the problem, but may be a very useful thing to do in some cases.

The path $\mathbf{X}$ is represented by a one-dimensional array `x[N]` with size $N = (M + 1)D + N_p$. The array is indexed as follows: the state $x_{n,l}$, where $n \in \{0, 1, \ldots, M\}, l \in \{0, 1, \ldots, D-1\}$, has index $i = nD+l$, and the parameter p_l, where $l \in \{0, 1, \ldots, N_p-1\}$, has index $i = (N-N_p)+l$. This indexing system is relevant only because the components of the path are updated the sequence `x[0],...,x[N-1]`.

We use the Metropolis Monte Carlo algorithm in a form similar to what is shown in Fig. 3.1. The proposed changes to the path are made one component at a time in order. The proposed changes are drawn from uniform distribution in the interval $(-\delta_i, \delta_i)$. The psuedo-random numbers are generated using the double precision SIMD oriented Fast Mersenne Twister (dSFMT) code by M. Saito and M. Matsumoto [56]. The size of the

random walk step δ_i is allowed to be different for each component of the path. The δ_i's are set automatically using a procedure described in Appendix A, with the goal of getting 50% of the proposals accepted per path component. The goal in setting the δ_i's is to have the path space be sufficiently explored with a minimal number of path updates. If a δ is set too large the acceptance rate will be very low, and if it is set too small the path will not change very much per update. In either case the path space will not be explored efficiently.

After each component of the path has had one opportunity to be updated, the new path is used to add onto the running sum for the first through fourth moments of each component of the path. It some cases it may be useful to give some or all of the components of the path multiple opportunities to update before using the path to calculate moments, but that is not done here. The full path is usually not stored, since the information we are interested is contained in the moments. At the end of the process of generating paths, the moments are then used to compute mean, standard deviation, skewness, and kurtosis of every component of the path.

Another important detail is that the initial `Ninit` number of paths are disregarded during the initialization phase. This is done so that the initial guess of a path does not contaminate the statistics, which should be independent of the initial guess. It takes many updates for the influence of the initial guess path to be wiped out. The number of paths to disregard, `Ninit`, and the total number of paths, `NIt`, must be determined by running the code and looking at how some quantity of interest $z_j = \chi(\mathbf{X}^{(j)})$ changes as the paths are updated. Ideally z_j should come to an equilibrium value before `Ninit` number of updates and then continue to fluctuate around the equilibrium.

We assume that measurement noise is Gaussian and uncorrelated in time and among the different components of the state. We also make the same assumptions about noise in the dynamics, and so the action takes the form

$$\begin{aligned} A_0(\mathbf{X}) = & \frac{1}{2}\sum_{n=0}^{M}\sum_{l=0}^{D-1} \mathrm{R^m}_{n,l}\left(x_{n,l} - \mathrm{y}_{n,l}\right)^2 \\ & + \frac{\beta}{2}\sum_{n=0}^{M-1}\sum_{l=0}^{D-1} \mathrm{R^f}_{l}\left[x_{n+1,l} - x_{n,l} - \frac{\Delta t}{2}\left(F_l(\mathbf{x}_{n+1},\mathbf{p}) + F_l(\mathbf{x}_n,\mathbf{p})\right)\right]^2. \end{aligned} \tag{3.5}$$

We take the initial distributions $P(\mathbf{x}(0))$ and $P(\mathbf{p})$ to be uniform, and so they are additive constants to the action that can be neglected. The deterministic part of the model differential equation is $\frac{d\mathbf{x}}{dt} = \mathbf{F}(\mathbf{x},\mathbf{p})$ which is assumed to be known. The constants

```
Set x[] to initial guess path
Set mean[] to 0
for it = 0 to NIt -1 {
    for i = 0 to N-1 {
        xp = x[i] + delta[i]*Uniform(-1,1)
        dA = Change in action caused by replacing x[i] with xp
        Paccept = min(1, exp(-dA))
        if( Uniform(0,1) < Paccept )
            x[i] = xp
     }
     if( it >= Ninit ){
         for i = 0 to N-1
             mean[i] = mean[i] + x[i]/(NIt-Ninit-1)
     }
}
```

Figure 3.1: Pseudo-code for Metropolis Monte Carlo

$\mathrm{R}^{\mathrm{f}}, \mathrm{R}^{\mathrm{m}}$ are the confidence in the model and the measurements respectively, and may be different at different points in the (l, n) grid. The measurements are $\mathrm{y}_l(n)$ where $l \in \{0, 1, \ldots, D-1\}$ are known constants read in from data files. Note that now the sum over measurements is over all D components, but this is just for convenience and not a real change. Since measurements are not typically available at every point in the (l, n) grid, the $\mathrm{R}^{\mathrm{m}}{}_l(n)$'s associated with missing measurements are set to zero. The parameters are p_l where $l = 0, 1, \ldots, N_p - 1$, and they are considered part of the path $\mathbf{X}$.

The parameter β in Eq. (3.5) is used to do a type of simulated annealing. It may be set to start out at $\beta = \beta_0 < 1$ and gradually increased up to $\beta = 1$ during the initialization phase. The idea behind this is that it may be beneficial to have the action initially be dominated by the nice smooth Gaussian terms and gradually increase the influence of the possibly complicated and nonlinear model. This turns out to be very important, particularly in the low dynamical noise setting, and is explored in more detail in Chapter 4 where we examine the shape of $A_0(\mathbf{X})$.

One additional practical detail is that we may wish to put bounds on the state variables and parameters, instead of allowing them to range from $-\infty$ to ∞. A simple way to accomplish this is to reject any proposed moves that would take a state or parameter out of bounds. This is equivalent to saying that the action becomes infinite if any of the components of the path go out of bounds, which means that such a path has zero probability and should not be sampled. Often the bounds can be set based on what makes sense in the model, or from other previous measurements.

3.5 Application to Example Systems

We now show how the path integral Monte Carlo (PIMC) method just described can be applied to three different problems. The first example is an ensemble of damped harmonic oscillators with stochastic forcing. In this example measurements are only provided at $t = 0$, so this is not really an example of data assimilation, but it is a simple test case with a known solution. It is an example of how a stochastic differential equation with a given initial distribution can be solved using PIMC. The next example uses simulated data generated with the Lorenz 96 model, and demonstrates state estimation and prediction. The last example uses real data recorded from the Colpitts oscillator circuit. The point of this example is to show that the method can be used when the model is not known exactly, which is the case in any real system, and when there are

several unknown parameters.

3.5.1 Damped Harmonic Oscillator with Noisy Forcing

We now consider the problem of a particle of mass m confined to one dimension by a linear restoring force and immersed in a viscous fluid at temperature T. The particle feels a viscous drag when it moves through the fluid, and also experiences random forces from the thermal motion of the fluid. This problem was originally solved analytically in [9], and also discussed in [33]. Here we study it as a demonstration of the path integral Monte Carlo technique, and compare the results to the exact answer.

The dynamics of the particle can be modeled as a damped harmonic oscillator with random forcing. The random variables, $X(t)$ for position, and $V(t)$ for velocity will evolve according to the stochastic difference equations (for small Δt)

$$\Delta X = V(t)\Delta t, \tag{3.6}$$

$$\Delta V = -\omega^2 X(t)\Delta t - \gamma V(t)\Delta t + \sqrt{2\Gamma\Delta t}N_t^{t+\Delta t}(0,1) \tag{3.7}$$

where $\Delta V = V(t+\Delta t) - V(t)$, $\Delta X = X(t+\Delta t) - X(t)$, and $N_t^{t+\Delta t}(0,1)$ represents an independent sample drawn at each time step from a Gaussian distribution with zero mean and unit variance. The fluctuation strength Γ and the damping coefficient γ are related to the temperature by the fluctuation-dissipation relation, $\Gamma/\gamma = kT/m$. As in Brownian motion, the random step has size has to be proportional to $\Delta t^{1/2}$, to get self-consistancy when the time step is divided in half and two normal distributions are added, since according to the normal sum theorem the variances of the two normal distributions add together [33].

We now consider an ensemble of particles that initially at $t = 0$ have a Gaussian distribution of position and velocity. Since this system is linear, the distributions of position and velocity will remain Gaussian at all times, so we only need to keep track of the 1st and 2nd moments: $\langle X(t)\rangle, \langle V(t)\rangle, \langle X(t)^2\rangle, \langle V(t)^2\rangle, \langle X(t)V(t)\rangle$. We can find a system of five ordinary differential equations for the moments by manipulating the equations of motion and averaging over the ensemble.

The evolution of the means $\langle X\rangle$ and $\langle V\rangle$ come from averaging the equations of motion, and are the same as the motion of a single damped harmonic oscillator without

random forcing:

$$\begin{aligned}\Delta \langle X \rangle &= \langle V \rangle \Delta t \\ \Delta \langle V \rangle &= -\omega^2 \langle X \rangle \Delta t - \gamma \langle V \rangle \Delta t.\end{aligned}$$

Now we work on the equation for $\langle V^2 \rangle$. Because of the presence of the term proportional to $\Delta t^{1/2}$, some of the ordinary rules of calculus do not apply. We use the fact that

$$\begin{aligned}\Delta(V^2) &= (\Delta V)^2 - 2V^2(t) + 2V(t)V(t+\Delta t) \\ &= (\Delta V)^2 - 2V^2(t) + 2V(t)\left[V(t) + \Delta V\right] \\ &= (\Delta V)^2 + 2V(t)\Delta V. \end{aligned} \tag{3.8}$$

In ordinary calculus ΔV is proportional to Δt and so $(\Delta V)^2$ is neglected in the limit of $\Delta t \to 0$, but here ΔV has a term proportional to $\Delta t^{1/2}$ and so it can not be dropped completely. From Eq. (3.7) we have

$$(\Delta V)^2 = 2\Gamma \Delta t N_t^{t+\Delta t}(0,1)^2 + O(\Delta t^{3/2}).$$

We can then do an ensemble average of Eq. (3.8) and use Eq. (3.7) to get

$$\begin{aligned}\Delta \langle V^2 \rangle &= \langle (\Delta V)^2 \rangle + 2 \langle V(t) \Delta V \rangle \\ &= 2\Gamma \Delta t - 2\omega^2 \langle XV \rangle \Delta t - 2\gamma \langle V^2 \rangle \Delta t.\end{aligned}$$

We do a similar thing for $\langle X^2 \rangle$ and for $\langle XV \rangle$, again dropping terms of order $\Delta t^{3/2}$ and higher in anticipation of taking the limit $\Delta t \to 0$.

Next we take the limit $\Delta t \to 0$ in all five equations, to get a closed system of five coupled ordinary differential equations:

$$\begin{aligned}\frac{d\langle X \rangle}{dt} &= \langle V \rangle \\ \frac{d\langle V \rangle}{dt} &= -\omega^2 \langle X \rangle - \gamma \langle V \rangle \\ \frac{d\langle X^2 \rangle}{dt} &= 2 \langle XV \rangle \\ \frac{d\langle V^2 \rangle}{dt} &= -2\omega^2 \langle XV \rangle - 2\gamma \langle V^2 \rangle + 2\Gamma \\ \frac{d\langle XV \rangle}{dt} &= -\omega^2 \langle X^2 \rangle - \gamma \langle XV \rangle + \langle V^2 \rangle .\end{aligned}$$

This system can be solved exactly as shown in [9], but since our goal is just to compare to the Monte Carlo calculation we simply numerically integrate to find a solution for a particular choice of initial conditions and parameters.

As a specific example we choose parameters $\omega = 1, \gamma = 0.3, \Gamma = 0.0015$. This implies that $kT/m = \Gamma/\gamma = 0.005$. We also choose initial conditions: $\langle X(0)\rangle = 1, \langle V(0)\rangle = 0, \langle X(0)^2\rangle = 1/R_m + 1, \langle V(0)^2\rangle = 1/R_m, \langle X(0)V(0)\rangle = 0$, where $R_m = 100$. The system can then be integrated numerically to get the exact results shown in Fig. 3.2 with solid lines representing the mean $\langle X(t)\rangle$, and mean plus or minus the standard deviation $\sigma_{X(t)} = \sqrt{\langle X(t)^2\rangle - \langle X(t)\rangle^2}$, and similarly for the velocity.

We also solve the same problem using the Monte Carlo path integration method described previously in this chapter. We specify the initial distribution by providing one measurement of velocity at $t = 0$, $y_{0,1} = 0$, and one measurement of position at $t = 0$, $y_{0,2} = 1$. Both measurements have an uncertainty of $R_m^{-1/2} = 0.1$. Equivalently, we can provide no measurement terms but include two Gaussian prior terms for the initial condition distributions with the appropriate mean and variance instead. The parameters ω and γ are included as known constants, and not integrated over.

We set the dynamical noise level in the velocity equation by setting $R_f = 1/(2\Gamma\Delta t)$. Here we choose $\Delta t = 0.2$, and so $R_{f,1} = 1667$ in the velocity equation. The position equation has no noise in it, so in principle we would like $R_{f,2} \to \infty$, but that is not possible since the assumption that the transition probabilities are stochastic is built into the method. However, we can approximate the deterministic equation by setting the noise level to be low or R_f large, so we set it to a large number: $R_{f,2} = 100R_{f,1}$. The results of the Monte Carlo calculation using 19.2 million path updates (and discarding the first 1.2 million path updates) are shown in Fig. 3.2, and match up closely with the exact results. Making $R_{f,2}$ larger seems to make the problem more difficult because the autocorrelation time of the Monte Carlo process increases, and so more paths need to be generated to get good results. Another possibility is to not include the position variables at each time step in the path, but to instead only include the initial condition. The position at each time step would then be calculated by integrating the velocity.

3.5.2 Lorenz 96

We use the Lorenz 96 model to do 'twin experiments' using the path integral Monte Carlo method of data assimilation. This means simulated data was generated from

the D-dimensional Lorenz 96 model equations given in Chapter 1, with measurement noise added. For the examples shown in this chapter $D = 5$, but the method has been successfully tested with many other D values up to $D = 100$.

More specifically, the model equations with $D = 5$ and some choice of initial conditions, and with the forcing constant set to $f_0 = 8.17$ were numerically integrated using 4th order Runge-Kutta to generate the 'true path' or actual trajectory $\mathbf{w}(t)$. The choice of $f_0 = 8.17$ puts the dynamics in the chaotic regime, with a largest Lyapanov exponent of $\lambda_{max} = 0.53$. Here we use the exact deterministic model, and do not add any noise to the dynamics, but we explore the issue of noise in the dynamics in the next chapter. From $\mathbf{w}(t)$ simulated measurements are generated by adding white Gaussian noise: $y_{n,l} = w_l(n\Delta t) + \sigma N_{n,l}(0,1)$, where $N_{n,l}(0,1)$ is a number drawn from a Gaussian distribution with zero mean and unit variance, and $\sigma = 0.35$ is the level of the measurement noise. Actually the measurements are only generated for some values of (n,l), and here we choose $n\Delta t = 0.0, 0.1, 0.2, \ldots, 4.0$, and $l = 0, 1$, so a total of 82 data points are used. In the examples the time step was set to be $\Delta t = 0.025, 0.05$, or 0.1.

We have observations of $w_0(t)$ and $w_1(t)$ available in the observation window $0 \leq t \leq 4$ with a time interval of 0.1 between each point. The other three variables $w_2(t), w_3(t), w_4(t)$ are unobserved. Also the forcing constant f_0 is treated as an unknown parameter, represented by a single component of the path. The goal is to approximate the posterior distribution of states $P(\mathbf{X}|\mathbf{Y})$ using Metropolis Monte Carlo to generate a series of paths, from which we can calculate moments of every component of the path. The path $\mathbf{X}$ includes the state vector, including observed and unobserved components, at every time point, as well as the forcing parameter. The reason for calculating the moments is to turn the the posterior distribution into something that can be used to make predictions. We use the first moment $\langle x_{n,l} \rangle$ as an estimate of what the true state was likely to be, or will likely be at future times, and use the second moment to calculate the standard deviation to use as an uncertainty estimate.

We also calculate the third and fourth moments of the components of the path. Since the model is nonlinear, there is no reason to expect the distributions to be Gaussian– even if it is Gaussian at $t = 0$ it will be distorted as it is evolved forward by the model. These moments can be used to compute skewness

$$\frac{\left\langle (x_{n,l} - \langle x_{n,l} \rangle)^3 \right\rangle}{\left\langle (x_{n,l} - \langle x_{n,l} \rangle)^2 \right\rangle^{3/2}},$$

and (excess) kurtosis

$$\frac{\left\langle (x_{n,l} - \langle x_{n,l} \rangle)^4 \right\rangle}{\left\langle (x_{n,l} - \langle x_{n,l} \rangle)^2 \right\rangle^2} - 3$$

which both vanish for Gaussian distributions. Computing these quantities allows us to test the common assumption that the marginal state distributions $P(x_{n,l}|\mathbf{Y})$ are Gaussian. The issue of non-Gaussian distributions in data assimilation is discussed in [50], and they give a way of quantifying the deviation of a distribution from Gaussian. They do this by computing the relative entropy between the distribution in question and a Gaussian distribution with the same mean and variance, which can be approximated as $s^2/12 + k^2/48$ for small absolute value of skewness, s, and kurtosis, k.

Using this simulated data set of 82 data points with observations of only two of the variables, and the Lorenz 96 model, we can calculate the action as given in Eq. (3.5). For the first example we set $R_m = 8$ for all the (n, l) points that have measurements associated with them, and as always $R_m = 0$ for all the (n, l) points that do not have measurements associated with them. We make this choice because in this case we know the level of the measurement noise was $\sigma = 0.35$, so we set $R_m = 8 \approx \sigma^{-2}$, but in a real experiment the measurement noise level would have to be estimated based on considerations of the measurement process. The method does not appear to be very sensitive to this setting, or even the form of the noise distribution. In this case the actual noise distribution and the assumed form of the noise distribution which goes into the action are both Gaussian white noise.

We also have to make a choice for the R_f term, which characterizes the amount of noise in the dynamics. In this case there is really no noise in the dynamics (aside from small numerical errors in the integration), because we generated the simulated data directly from the model. In the next chapter there are some examples of simulated noise added into the dynamics for this same system. In a real experiment the noise term can be used to represent certain types of random model error or stochastic forces. The assumption that the noise is Gaussian and uncorrelated in time and among the different components of the state are built into this form of the action. Recall that the physical quantity that represents the amount of noise is $\Gamma = 1/(2R_f\Delta t)$.

For a first example we set $\Delta t = 0.05$ and $R_f = 327.5$. Now we have everything needed to be able to evaluate the action Eq. (3.5) for a particular choice of path $\mathbf{X}$, which means we can perform the Metropolis Monte Carlo procedure to generate a series of paths, and from the paths calculate mean, standard deviation, skewness, and kurtosis

of each component of the path.

The specific settings used in the Monte Carlo process were: a block size of 1000 path updates, the first 200 blocks were discarded and the next 1920 blocks were used to calculate the statistics. The block refers to a sequence of path updates that are grouped together to reduce the storage requirements. Simulated annealing was used by starting with $\beta = \beta_0 = 10^{-4}$ which increased by multiplying by a constant factor after each path update, over the first 100 blocks generated during the initialization phase. Note that these paths are disregarded when calculating statistics.

The MC calculation was done on the interval $0 \leq t \leq 4$. To make a prediction beyond $t = 4$ the final state (and parameter) value of each path was integrated forward up to $t = 6$ using fourth-order Runge-Kutta (RK4) with a time step of 0.0125 which means in this case that the analysis time step Δt was divided into four integration steps. No noise was added to the integration in this case, so effectively $R_f \to \infty$ after $t = 4$.

The results of the MC calculation with $R_f = 327.5$ and the RK4 prediction are shown in Fig. 3.3 for one observed variable, $x_{n,0}$, and in Fig. 3.3 for on unobserved variable, $x_{n,3}$. The results of the parameter estimate were $f_0 = 8.15 \pm 0.16$, very close to the true value of 8.17, with a skewness of 0.07 and kurtosis of 0.08, consistent with the distribution being almost Gaussian. We also see that both the skewness and kurtosis are close to zero in the observation window, but that the marginal distributions $P(x_{n,l}|\mathbf{Y})$ become less Gaussian for times after the end of the observation window ($t > 4$). The figures show that the data points have a strong influence in guiding $P(x_{n,l}|\mathbf{Y})$: the marginal distributions have narrow peaks centered near the true value, and they are approximately Gaussian. Once the data points are no longer available at times past the end of the observation window, the marginal distributions spread out and become non-Gaussian. This is expected, because without any measurements for guidance, the chaotic dynamics will spread out and distort a distribution that was well-confined at $t = 4$.

Next the same example calculation was done but this time with higher weight, or more confidence, given to the model, so this time $R_f = 655$. The results are shown for one of the observed variables in Fig. 3.5, and are very similar to the previous results with $R_f = 327.5$. Increasing the R_f setting by a factor of two has caused the state uncertainty in the observation window to decrease by about a factor of about 1.2. Also the skewness and kurtosis in the observation window are slightly further from zero, now

Table 3.1: Autocorrelation numbers J_{ac} calculated from the results in Fig. 3.8 ($a = 327.5$, and $b = 0.025$), and compared to the scaling $J_{ac} \propto \Delta t^{-1}$ for constant Γ in the last two columns. As a function of Γ and Δ the scaling appears to be roughly $J_{ac} \propto \Gamma^{-1/2}\Delta t^{-1}$.

R_f	Δt	$\Gamma = (2\Delta t R_f)^{-1}$	J_{ac}	$J_{ac}/7.5$	$b\Delta t^{-1}$
$4a$	b	$(8ab)^{-1}$	7.5	1.00	1
$2a$	$2b$	$(8ab)^{-1}$	3.6	0.48	1/2
a	$4b$	$(8ab)^{-1}$	1.8	0.24	1/4
$4a$	$2b$	$(16ab)^{-1}$	6.2	0.83	
a	b	$(2ab)^{-1}$	3.6	0.48	
a	$2b$	$(4ab)^{-1}$	2.2	0.29	

that the influence of the nonlinear model relative to the influence of the data has been increased.

The histogram representation of marginal distribution $P(f_0|\mathbf{Y})$ is shown in Fig. 3.6, which was generated by binning the values of f_0 for all 1920×1000 of the MC sample paths, for the two examples just discussed. We can see that these distributions are approximately Gaussian, and that the standard deviation decreases when R_f increases.

It is also important to examine how the paths generated by the MC process change iteration to iteration. We focus on the component of the path that represents f_0. To cut down on the amount of storage required, we group the path updates into blocks of 1000, and calculate the first through fourth moments of each component of the path for each block. Figure 3.7 shows how the blocked averages of f_0 change for two different R_f settings. The initialization phase is not shown, but if it was we would see the parameter drifting from the starting guess to near the true value. Those initial paths should not be used in the statistics calculations. We also can see that the blocked averages do not appear independent: there are some short-range correlations, and this is more apparent for the larger R_f case.

To quantify the amount that the blocked averages of f_0 are correlated we calculate the autocorrelation using Eq. (3.4). The results of the autocorrelation calculation for several different values of R_f and Δt are shown in Fig. 3.8 and summarized in Table 3.1. We find that if $\Gamma = (2\Delta t R_f)^{-1}$ is a constant then the scaling for autocorrelation number is $J_{ac} \propto \Delta t^{-1}$, thus if Δt is cut in half twice as many MC path updates need to be done to get to the same level of accuracy. We also see that if R_f is increased with Δt fixed J_{ac} also increases.

3.5.3 Colpitts Oscillator

This section is about applying the PIMC method to real data recorded from the Colpitts oscillator circuit which was introduced in Chapter 1. All three state variables, $V_{CE}(t), V_E(t)$, and $I_L(t)$ were recorded from the circuit with a sampling rate of 0.01 ms when the circuit was undergoing chaotic oscillations. This is shown by the solid black line in Fig. 3.9 which is only used as a comparison with the state estimates produced by the PIMC method.

The data used in the PIMC method is represented by blue dots in Fig. 3.9. This comes from the $V_E(t)$ recording, but only every 10th point was used, so $\Delta t = 0.1$ ms. There are 101 data points and they are at times $t = 0.0, 0.1, \ldots, 10.0$ ms. From these 101 data points we estimate the three state variables (one observed and two unobserved) at times $t = 0.0, 0.1, \ldots, 10.0$ ms, as well as the six parameters C_2, L, R, V_0, V_{th}, and β_F.

We also integrate each MC path forward past the end of the observation window to make state predications. We see that the projected paths start out well-confined but then spread out. This is because the model is chaotic, so slight changes in initial conditions (at t = 10 ms) and parameters lead to large changes in the trajectory at later times.

We show the estimated parameters in Table 3.2 and compare them to the measured values. The estimates all come out pretty close to the measured values with the exception of β_F. This issue is addressed further in [54] using different methods.

The specifics of the settings used in this calculation are: $R_0^f = 2500, R_1^f = 10000, R_2^f = 16, R_0^m = 0, R_1^m = 2500, R_2^m = 0$. The $l = 0, 1, 2$ subscript refers to the three state varaibles: V_{CE}, V_E, I_L in that order. The block size was 1000 path updates. The first 300 blocks were disregarded, and the next 480 blocks were used. Also a simulated annealing procedure was used during the first 20 blocks, with $\beta_0 = 0.01$. The prediction was done using RK4 with a step size of 0.05.

This example is useful because it demonstrates that the PIMC data assimilation method can work when applied to real data. The model is only approximate: it assumes that the inductor, capacitors, and resistors behave in the ideal linear way, and more importantly that that the transistor is adequately described by the simplified version of the Ebers-Moll equations. The real transistor is not exactly described by the Ebers-Moll model, particularly when the current going into the collector is large. The problem is also made more difficult by the fact that the dynamics are chaotic. It is essential that

the measurements are used for guidance.

Table 3.2: Colpitts Circuit parameter estimates compared to the directly measured values.

Name	Mean	St. Dev.	Measured	Units
C_2	6.9	0.5	7.23	μF
L	12.0	0.5	11.74	mH
R	39	2	39.3	Ω
V_0	0.65	0.01	0.63	V
V_{th}	25	3	27	mV
β_F	77	2	180	

3.6 Summary

Path integral Monte Carlo was used to calculate conditional expectation values of the form shown in Eq. (3.1). Each path is weighted by the factor $P(\mathbf{X}|\mathbf{Y}) \propto \exp[-A_0(\mathbf{X})]$. The key idea is to approximate the path integral by sampling paths more from the regions that are more important. This is done by generating MC paths that are distributed according to $\exp[-A_0(\mathbf{X})]$ with the Metropolis algorithm.

In particular, we calculated moments of each component of the path which describes the possible state evolutions. This information was then used to make state and parameter estimates with associated uncertainties. Also predictions were made by propagating the distribution at time $t = t_m$ forward using the model dynamics.

In the next chapter we take a closer look at the function $A_0(\mathbf{X})$ and see how its shape depends on the number of measurements and how the terms in the action are weighted. We also investigate the issues of model error and dynamical noise.

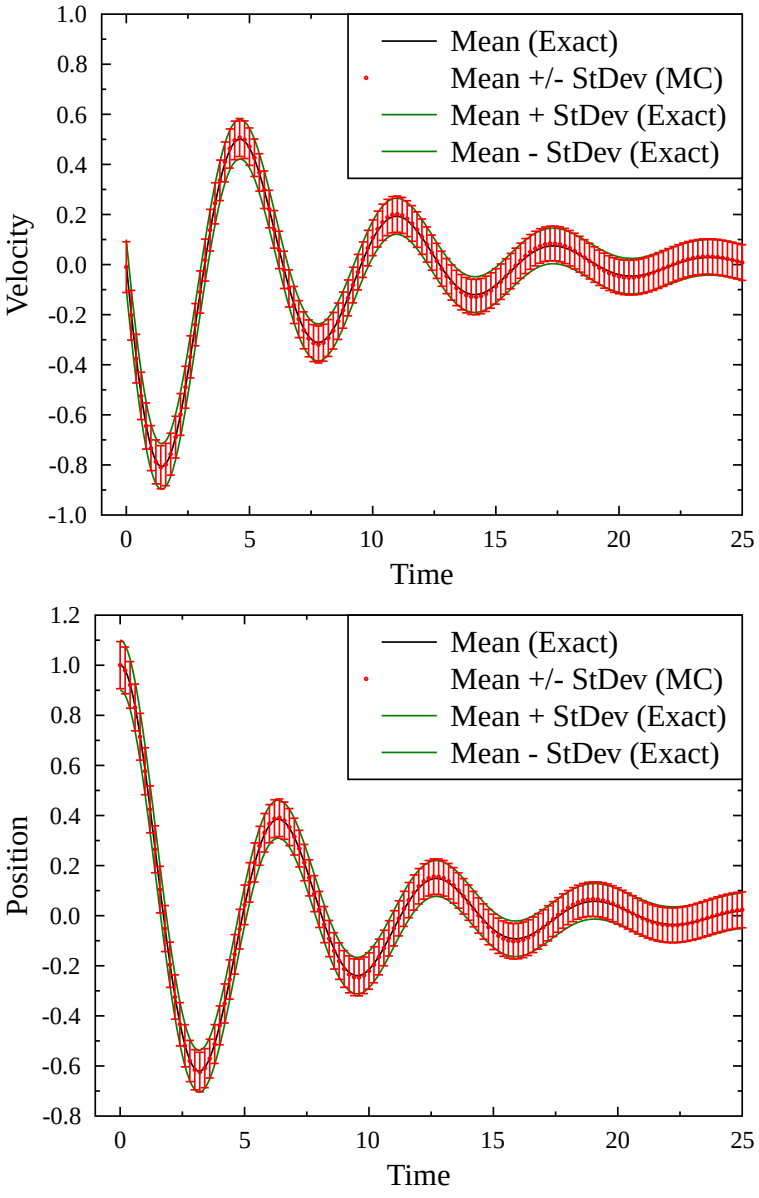

Figure 3.2: Comparision of the exact answer to the MC result for the stochastic damped harmonic oscillator.

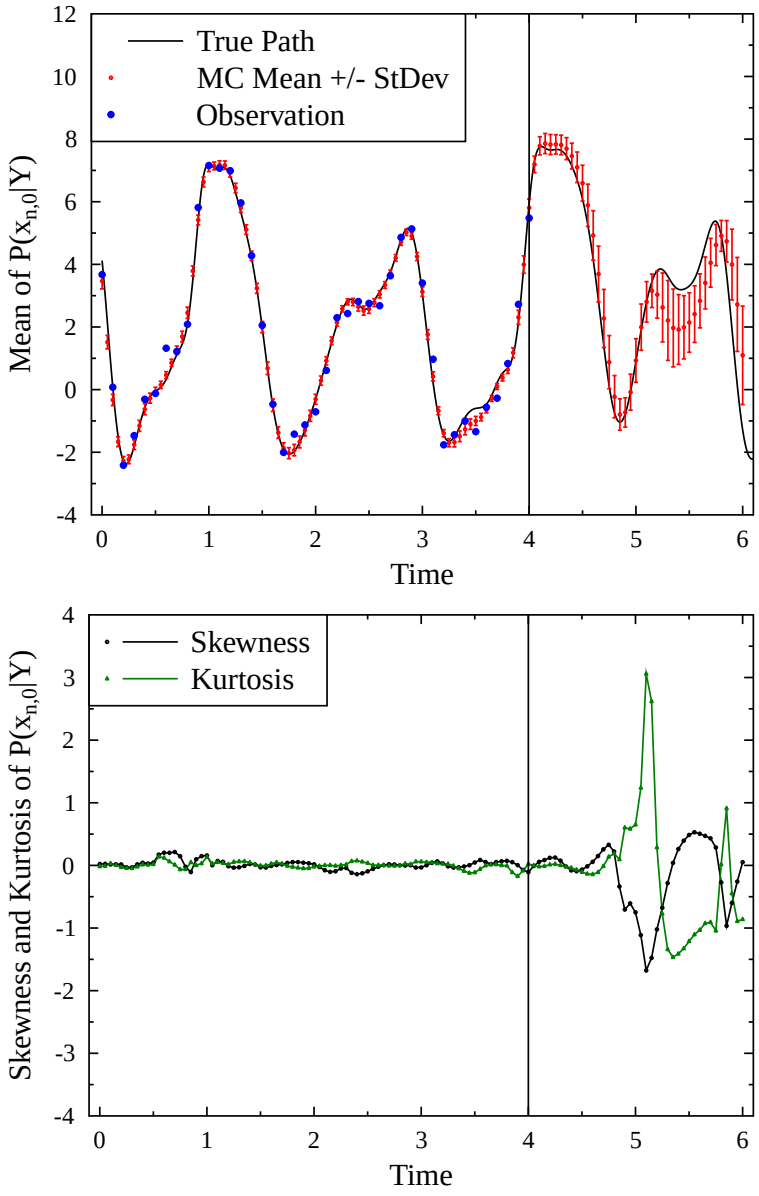

Figure 3.3: Results of the MC calculation for Lorenz 96 $D = 5$ with $R_f = 327.5$ for one observed variable, $x_{n,0}$, showing the mean plus or minus one standard deviation and compared to the true path (top), and the skewness and kurtosis of $P(x_{n,0}|\mathbf{Y})$ (bottom). Observations was only provided for $0 \leq t \leq 4$ of $x_{n,0}$ and $x_{n,1}$.

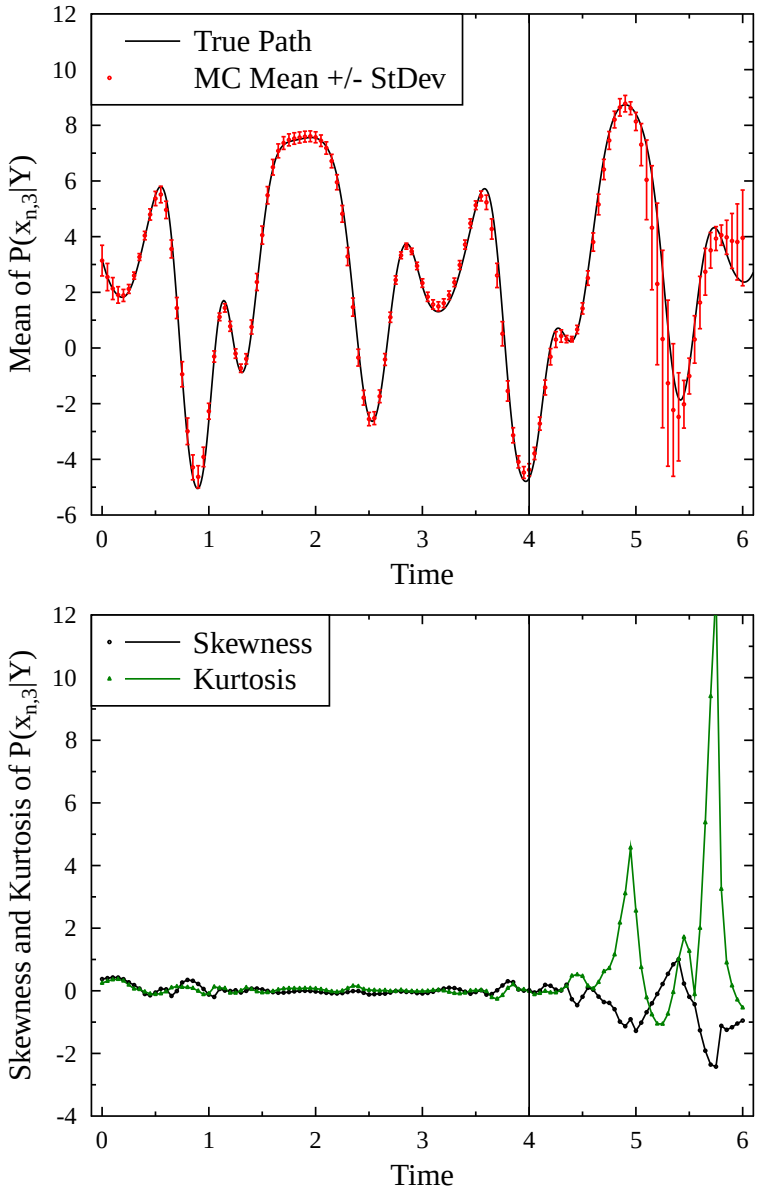

Figure 3.4: Same as the previous figure, but now showing one of the unobserved variables, $x_{n,3}$. The computed quantities come from the marginal distribution $P(x_{n,3}|\mathbf{Y})$.

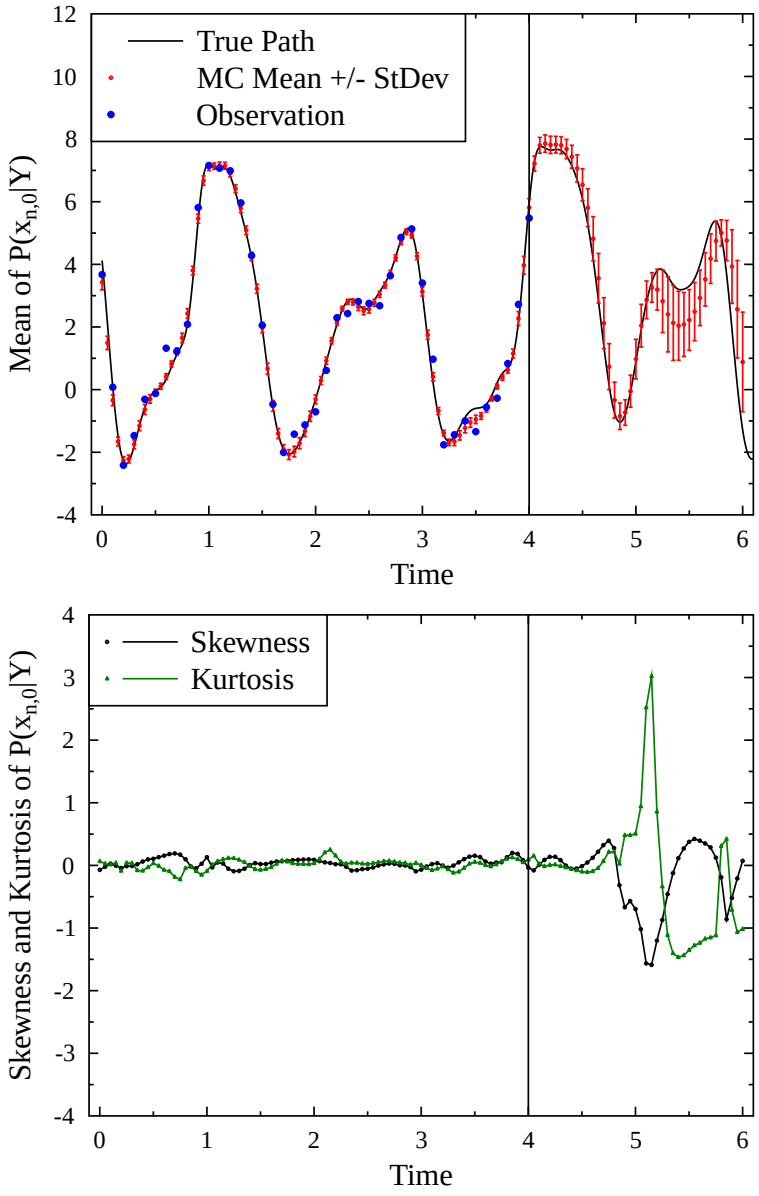

Figure 3.5: Results of the MC calculation for Lorenz 96 $D = 5$ with $R_f = 655$ for one observed variable, $x_{n,0}$, showing the mean plus or minus one standard deviation and compared to the true path (top), and the skewness and kurtosis of $P(x_{n,0}|\mathbf{Y})$ (bottom). Observations was only provided for $0 \leq t \leq 4$ of $x_{n,0}$ and $x_{n,1}$.

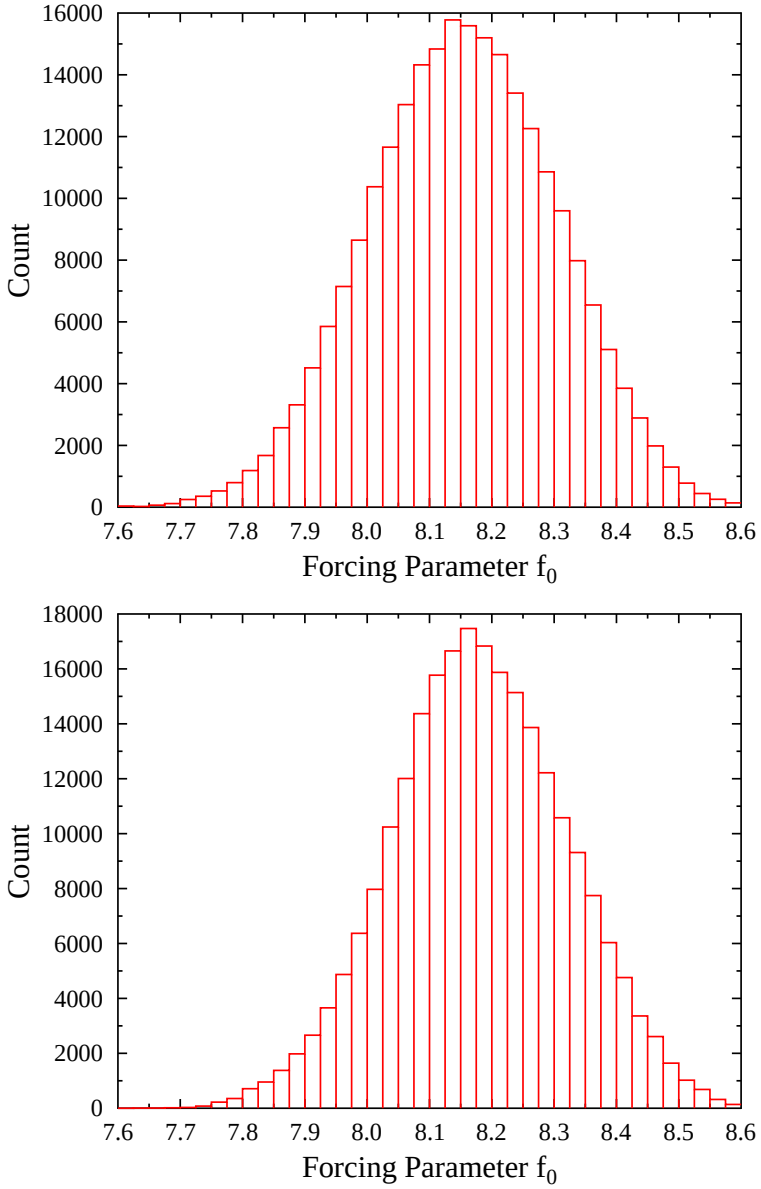

Figure 3.6: Histogram representation of the marginal distribution $P(f_0|\mathbf{Y})$ for $R_f = 327.5$ (top) and $R_f = 655$ (bottom). The top distribution has mean, standard deviation, skewness, and kurtosis of 8.15, 0.16, 0.07, and 0.08 respectively. The bottom distribution has mean, standard deviation, skewness, and kurtosis of 8.18, 0.13, -0.11, and 0.17 respectively.

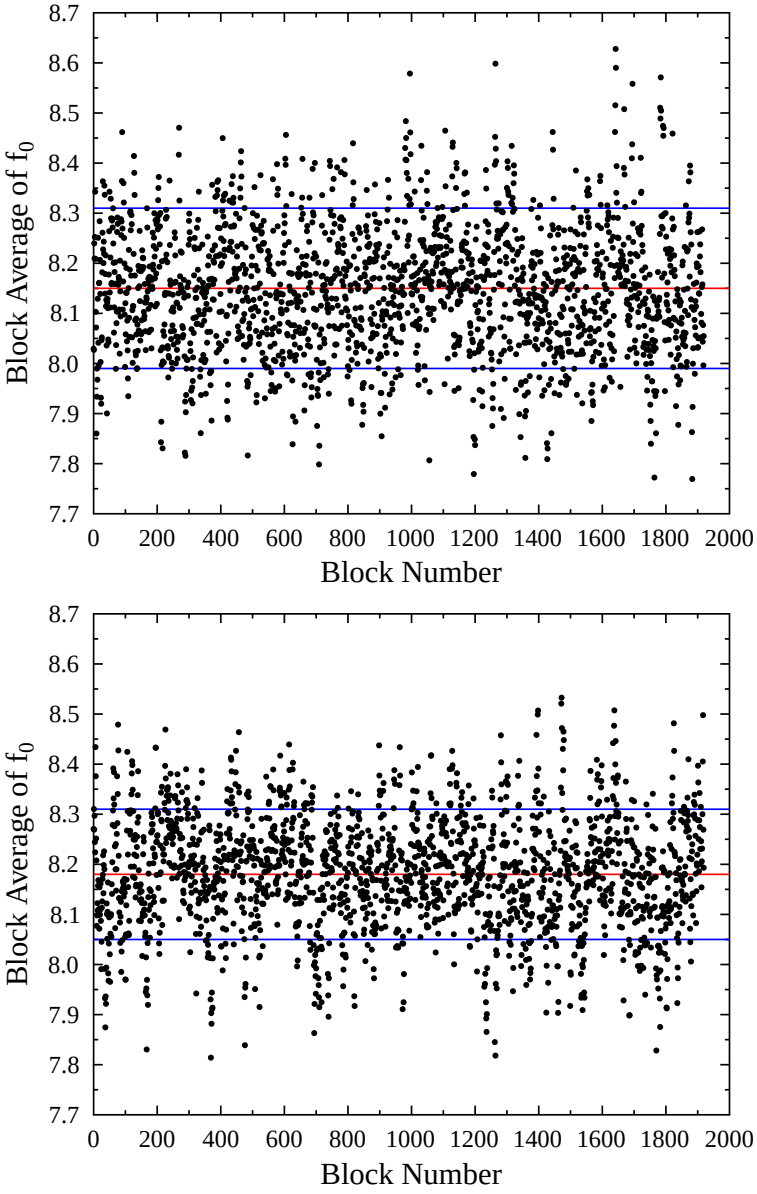

Figure 3.7: The forcing parameter f_0 averaged over blocks of size 1000 path updates is shown for 1920 blocks. Also shown is the average of f_0 over all 1000×1920 MC paths (red line) and the average plus or minus the standard deviation (blue lines). The top panel is for $R_f = 327.5$ and the bottom panel is for $R_f = 655$, both with $\Delta t = 0.05$. The true value used to generate the data was $f_0 = 8.17$. The number of blocks that need to be generated for the autocorrelation to decay to $1/e$ is roughly twice as much for the bottom as the top.

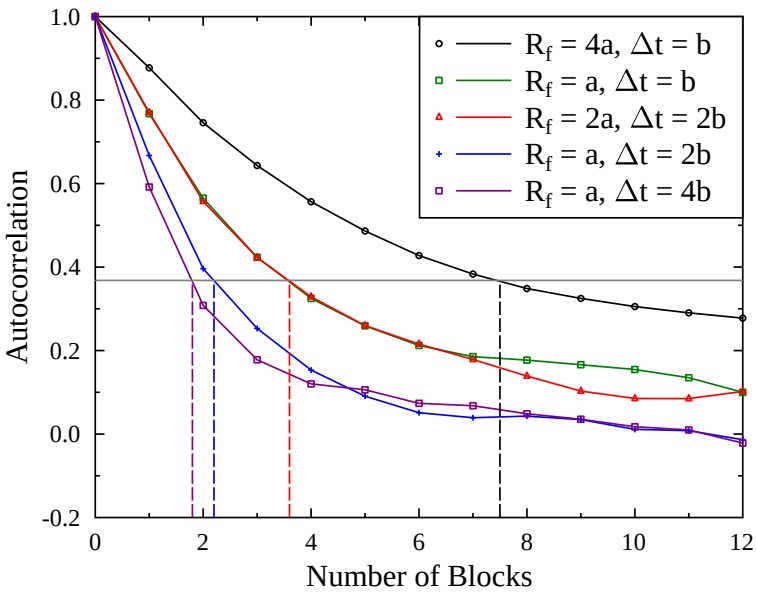

Figure 3.8: The autocorrelation calculated from the blocked averages of f_0 (two examples shown in Fig. 3.7) as a function of the number of blocks lag (1 block = 1000 path updates) shown for a variety of R_f and Δt, $a = 327.5$, and $b = 0.025$.

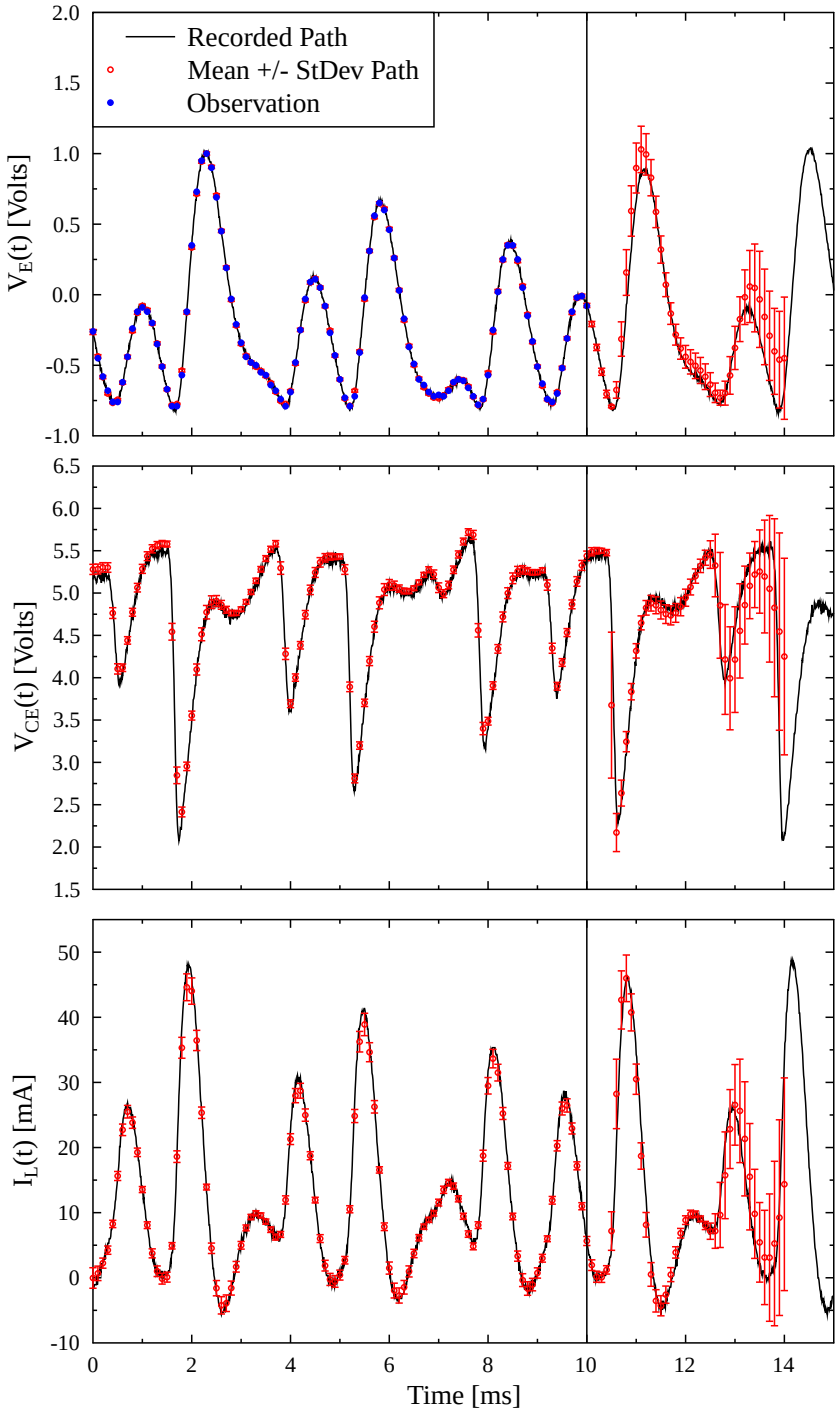

Figure 3.9: State estimate from PIMC for the Colpitts circuit. The mean path and standard deviation is found using PIMC. The estimates at $t > 10$ ms are predictions, since they are beyond the end of the observation window.

Chapter 4

Exploring the Shape of the Action

We saw in Chapter 2 how to formulate the conditional probability in path space as $P(\mathbf{X}|\mathbf{Y}) \propto \exp[-A_0(\mathbf{X}))]$, where $A_0(\mathbf{X})$ is called the action and is a function of the path $\mathbf{X}$. The action also depends on the time series of observations $\mathbf{Y}$ which we think of as known constants. Recall that the path $\mathbf{X}$ may represent any time history of the state of the model, as well as time-independent parameters. We now would like to examine the structure of the function $A_0(\mathbf{X})$, in particular we would like to know if the surface of $A_0(\mathbf{X})$ is smooth with few minima or rough with many minima. We will see that it is useful to find the local minima of $A_0(\mathbf{X})$ as well as calculate the curvature in the vicinity of these minima as a way of characterizing the function $A_0(\mathbf{X})$, which depends on the particular choice of model as well as the observations.

Recall that the action has the form

$$A_0(\mathbf{X}, \mathbf{Y}) = R_m E_m(\mathbf{X}, \mathbf{Y}) + \beta R_f E_f(\mathbf{X}), \tag{4.1}$$

where $E_m(\mathbf{X}, \mathbf{Y})$ is the measurement error term and $E_f(\mathbf{X})$ is the model error term, which both evaluate to positive numbers for any path $\mathbf{X}$. The measurement error term becomes large for paths inconsistent with the measurements, and the model error term becomes large for paths inconsistent with the model. The β is a control parameter used to adjust the relative weight of the two terms.

It is instructive to consider the simple limit of when $\beta = 0$. In this limit we have no knowledge of the system dynamics at all, so we cannot say anything about the unobserved variables, or make any predictions. We do have some information from the noisy measurements, but this only tells us about the state variables that were measured

at the times they were measured. We need to introduce a model to be able to use the measurement data to learn something about the unobserved states and parameters. As β increases the model resolution increases, or the dynamical noise level decreases, and we get some predictive ability.

It is also useful to think about the role of making additional measurements. When there are few measurements available the most probable regions of path space may be spread out over path space. Making additional measurements can greatly reduce the likelihood of a particular path that was previously improbable, if the new measurement is far away from that particular path. The additional measurement $\mathbf{y}_n$ can be thought of providing information which is used to update the distribution from $P(\mathbf{X}|\mathbf{y}_{1:n-1})$ to $P(\mathbf{X}|\mathbf{y}_{1:n})$.

We will see that if there are enough observations available then the most probable region of path space will be confined to a localized region in the vicinity of the global minimum of the action, and outside of this region of path space the action becomes much larger. If this is not the case and $A_0(\mathbf{X})$ has comparable value in many different regions of path space, then evaluating the path integrals becomes impractical or impossible. Even if the path integrals could be evaluated the answer would likely not provide much predictive power: the mean path would be an average over very many dissimilar paths spread out over path space, and so would not be very useful in estimating what the state was or predicting what the state will be in the future.

We will also see that to find the global minimum, even when sufficient measurements are available, some care must be taken in the way the minimization procedure is done. This is because for chaotic differential equations the trajectory is very sensitive to initial conditions and parameter values. The key idea to deal with this problem is to gradually turn up the influence of the model during the minimization process, or equivalent gradually reduce the model noise level. We also saw that this was necessary during the initial phase of the MC procedure for the same reason.

4.1 Motivation for Minimization

It is useful to think of a path $\mathbf{X}$ as a single point in path space. We can then consider an ensemble of points, with each one representing a path, that are initially distributed according to some density $P(\mathbf{X}, s = 0)$. We then have each member of the

ensemble evolve in a fictitious time s according to the Langévin equation

$$\frac{\partial \mathbf{X}(s)}{\partial s} = -\nabla A_0(\mathbf{X}(s)) + \sqrt{2\alpha}\boldsymbol{\eta}(s), \tag{4.2}$$

where $\boldsymbol{\eta}(s)$ is a vector of Gaussian white noise with zero mean and variance unity. There are no correlations in 'time' or among the different components: $\langle \eta_l(s)\eta_{l'}(s')\rangle = \delta_{l,l'}\delta(s-s')$. The distribution of points will evolve according to the Fokker-Planck equation

$$\frac{\partial P(\mathbf{X},s)}{\partial s} = \nabla \cdot \left(P(\mathbf{X},s)\nabla A_0(\mathbf{X}) + \alpha \nabla P(\mathbf{X},s)\right). \tag{4.3}$$

As $s \to \infty$ the distribution $P(\mathbf{X},s)$ will approach the stationary distribution $P_s(\mathbf{X}) \propto \exp[-\frac{A_0(\mathbf{X})}{\alpha}]$. When $\alpha = 1$ this distribution is proportional to the conditional probability $P(\mathbf{X}|\mathbf{Y})$ as formulated in Chapter 2, and this is the weighting term in the path integrals for conditional expectation values.

This gives us an intuitive way to think about how to sample paths from the distribution $P(\mathbf{X}|\mathbf{Y})$, at least in principle. Start with an arbitrary initial path $\mathbf{X}(s=0)$ and evolve the path according to Eq. (4.2). After enough 'time' has past the trajectory of $\mathbf{X}(s)$ will visit all of the most probable regions of the very high-dimensional path space with the 'time' spent in each volume of path space proportional to $P(\mathbf{X}|\mathbf{Y}) \propto \exp[-A_0(\mathbf{X})]$ integrated over that volume. The first term of Eq. (4.2) says to move the path down the gradient of the action toward a local minima, and the second term says to randomly perturb the path with Gaussian noise. The picture of what happens in the ideal case with only one minimum is as follows: a path starting anywhere in path space will move toward the global minimum, and the noise will cause the path to fluctuate around that minimium. The size of the fluctuations depend on α and on the curvature of $A_0(\mathbf{X})$ in the vicinity of the minimium. By tracking how this path evolves in s we get a representation of $P(\mathbf{X}|\mathbf{Y})$.

The limit where α becomes small corresponds to the case where the noise in the model as well as the noise in the measurements becomes small. This is the limit where R_f and R_m both become large, but the ratio R_m/R_f is fixed. We see that as α becomes small the fluctuations around the minimum become smaller. In the limit of $\alpha \to 0$ the only place in path space that has any contribution in the path integral is the global minimum path.

We can also think about the limit when the model becomes deterministic, that is $R_f \to \infty$, but R_m is still finite. As R_f becomes large paths that deviate from a solution to the model equations are heavily penalized. As $R_f \to \infty$ the only paths that can

contribute to the path integral are paths that are exact solutions to the model equations. In this limit the number of degrees of freedom is greatly reduced from $(M+1)D+N_p$ down to $D+N_p$, where D is the dimension of the state vector, M is the number of time steps, and N_p is the number of parameters. This is because in the deterministic limit once the D initial conditions and N_p parameters are specified, the trajectory is completely determined. The global minimum path is the path which exactly satisfies the model equations while minimizing the deviation from the measured data points. This path can be used as an estimate of the true state.

There is a balance between two pieces of the action. On one hand we have information about some of the state variables from noisy measurements. On the other hand we have information about what paths are likely based on our knowledge of the underlying dynamics as encoded in the model. The control parameter β is used to change the relative weight of these two contributions. When β is small the model is very noisy and does not have much predictive power. As β is increased the model noise is reduced, and the model will have more of an influence on $P(\mathbf{X}|\mathbf{Y})$. When $\beta \to \infty$ the model becomes deterministic, and can in principle exactly predict the true path given the initial conditions and parameters, so then the problem is to find the optimal set of initial conditions and parameters to match the data. This can be very difficult to do, because if the model is chaotic the trajectories will be very sensitive to changes in initial conditions and parameters.

In the path integral formulation of quantum mechanics, the probability amplitude is expressed as $\exp[iS/\hbar]$ integrated over all paths with fixed end points, where S is the action of classical mechanics [20]. When S is large compared to $\hbar$ the path integral is dominated by the stationary path where $\delta S = 0$. This is the only allowed path according to the principle of least action of classical mechanics. However, in quantum mechanics other paths must be considered and included in the path integral. Finding the path which minimizes $A_0(\mathbf{X})$ instead of considering the whole distribution $P(\mathbf{X}|\mathbf{Y}) \propto \exp[-A_0(\mathbf{X})]$ is analogous to the classical limit of quantum mechanics. It is also analogous to the mean field theory approximation of statistical mechanics which comes from doing a saddle point approximation of the path integral for the partition function. The path which minimizes $A_0(\mathbf{X})$ is also known as the maximum likelihood path.

4.2 Minimization of Action

We would like to examine the function $A_0(\mathbf{X})$ to see if it has a rough surface with many minima, or a smooth surface with only a few well-defined minima. This is important to know, because methods of approximating the path integral, saddle point or PIMC, will only be effective when the the most likely paths are well-confined in path space.

As a specific example, we use the Lorenz 96 model with $D = 5$ to generate a true path $\mathbf{w}(t)$ that is exactly described by the model without any noise. From $\mathbf{w}(t)$ we then generate simulated measurements by adding Gaussian noise: $y_l(n) = w_l(n\Delta t) + N_{l,n}(0, \sigma)$, where $N_{l,n}(0, \sigma^2)$ is a number drawn from a Gaussian distribution with zero mean and variance σ^2. Here we choose $\Delta t = 0.025$, $n = 0, 4, 8, \ldots, 160$, and $\sigma = 0.5$. For now we choose to only generate observations for $l = 0, 2$, so two states are observed and three are unobserved. We will revisit this later with other choices.

In the action we set $R_m = 4$ and $R_f = 0.01$, but also include a control parameter β that multiplies R_f. We choose $R_m = \sigma^{-2} = 4$ since here we know the strength of the measurement noise exactly, but this is not critical. We choose our data assimilation window to be $0 \leq t \leq 4$, and use $M = 160$ time steps. The number of components in our path is $(M + 1)D + N_p = 161 \times 5 + 1 = 806$, and so we are doing a minimization in a 806-dimensional space.

We use the Fleecther-Reeves conjugate gradient method [51, 21] to do an unconstrained minimization of $A_0(\mathbf{X})$. To use this method we need to numerically evaluate the function $A_0(\mathbf{X})$ and the gradient $\frac{\partial A_0(\mathbf{X})}{\partial \mathbf{X}}$ for any value of $\mathbf{X}$. We also need to provide an initial guess path $\mathbf{X}^{(0)}$. Here the initial guess is made by choosing each of the 5 state variables to be a constant in time with a value drawn from a uniform distribution $-1 < x_l(0) < 1$, and the forcing parameter is drawn from a uniform distribution $6 < f < 10$. Clearly this guess path will not be consistent with the model and will not be close to the measurements, so the action will start out large.

To get an idea of the the number of minima of $A_0(\mathbf{X})$ we do the minimization procedure 100 times for different initial guesses of the path, and plot the action of the paths found after each of each minimization procedure. This is shown in Fig. 4.1 for $\beta = 1$. We find four distinct minimum paths which do not differ very much in action. In Fig. 4.2 one observed state and one unobserved state is plotted for two of the minimum paths that were found. The top two graphs shows the global minimum path ($A_0 \approx 0.036$)

and the bottom two graphs show the next highest local minimum ($A_0 \approx 0.062$). We see that in both cases the measurements are over-fit, meaning that the path follows the measurements more closely than it should considering that the measurements are noisy, and they are about the same. However, the unobserved components of the four paths are different.

Next we follow each of these minima up to larger β. This is done by starting with one of the four minima found with $\beta = 1$, increasing β by a factor of two and doing the minimization again. Keep increasing β and minimize again starting at the optimal point found in the previous step. In this way we can gradually get to the limit where β is large and the dynamics is essentially deterministic. This is shown in Fig. 4.3 for the sequence of $\beta = 2^0, 2^1, \ldots, 2^{29}$. It is also shown in Fig. 4.4 for just the global minimum, with the measurement part of the action and the full action displayed separately. In this way we are following these four minima up to larger β, but not necessarily finding new minima that may appear. We see that the minima separate in action as β increases. This means for large β only the global minimum is relevant because the next largest minimum found has action about 10 times larger. Also since we know in this case that the model is perfect and we know the measurement noise level, we can find the expected action for the true path which is $A_0 = N_{data} R_m / 2\sigma^2 = 82 \times 4/2 = 41$. The solid gray line in the figure is plotted at this value. We also compute the RMS error by comparing the minimum path to the true path. We see that as β gets large only one solution approaches the true path, and the RMS error of the other solutions increases. The two lowest action paths found at $\beta = 1$ when followed up to $\beta = 2^{24}$ are displayed in Fig. 4.5. We see that the global minimum path approaches the true path, but the second local minimum goes further away from the true path as β increases. This is consistent with the fact that the action of the second local minimum grows much more than the action for the global minimum as β is increased.

4.2.1 Getting to the Low Noise Limit

We now will see why it is necessary to start with β small and gradually increase. In Fig. 4.6 we show what happens if we start with $\beta = 2^{24}$. We do not find the global minimum at $A_0 \approx 41$ (marked with a gray line) that we did find using the process of increasing β gradually. This minimum is still there, it is just very difficult to find, because of the chaotic nature of the model which now has a large influence since β is

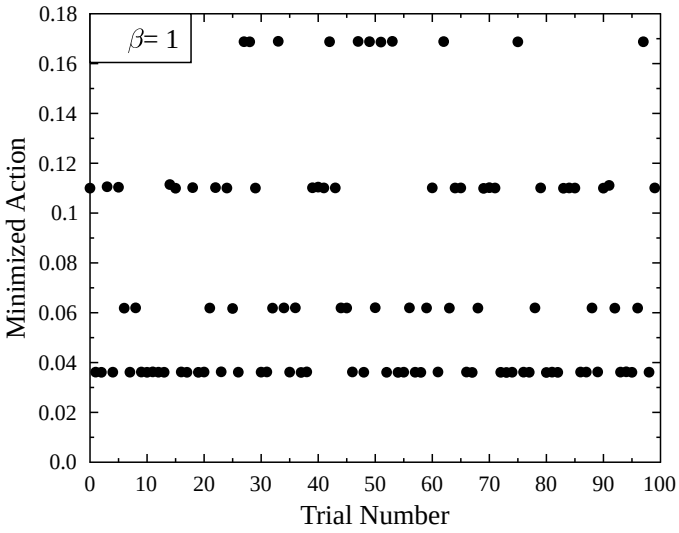

Figure 4.1: Minimization $A_0(\mathbf{X})$ starting from 100 different initial paths, with $\beta = 1$. We find four distinct minima that are not very well-seperated.

large. The global minimum is hard to find because the volume of path space which 'drain' to the global minimum is a small fraction volume of the entire path space that we are considering.

By starting with small β the model will initially have very little impact, and so any model will be able to fit the data. As β is increased we are forcing the path to follow the model more precisely, and at a certain point that means the fit to the data points will get worse if the data is inconsistent with the model. Starting with small β is also a form of 'regularization', in the sense that unstabilizing chaotic effects in the model will be suppressed initially. In other words, the dynamics term in the action starts out as a very 'loose' constraint on the paths, and so the path will not be forced to follow the model exactly, it will instead stay close to the data. If the constraint was 'rigid' the path would have to be exactly consistent with the model, and the trajectory would be very sensitive to initial conditions and parameters, which means it would be very hard to find the correct path. So we start out with the constraints loose and gradually tighten them by increasing β. In this case we have a perfect model, so there is no limit in principle to how large β should be made (of course there are still errors because errors that come from the discretization used in the action), but for data from any real system the model will not be perfect, and so β does not need to go to infinity even in principle.

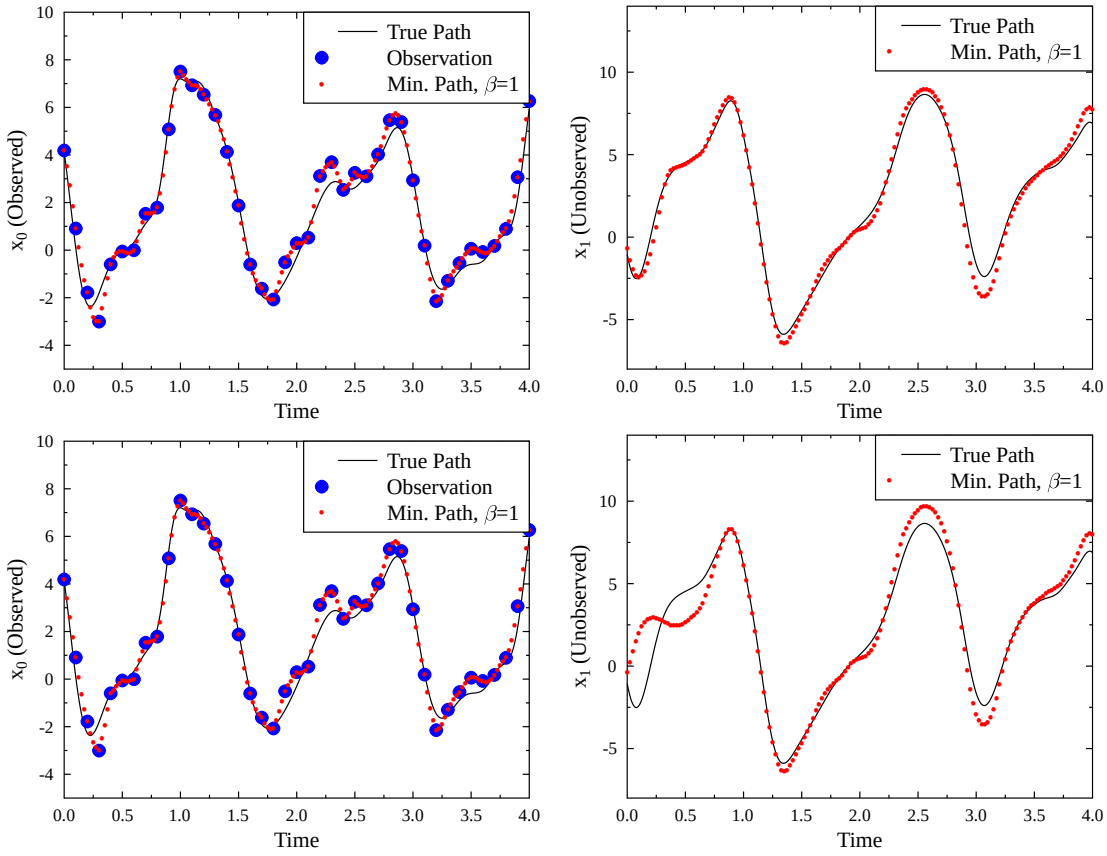

Figure 4.2: One observed and one unobserved component of the global minimum path found with $\beta = 1$. The top two graphs represent the global minimum path, and the bottom two graphs represent the second smallest minimum path. The parameter values are: $f = 8.54$ (top) and $f = 9.86$ (bottom).

4.2.2 Effect of Observations

We now continue with the same example of Lorenz 96 $D = 5$ to examine the effect of observations. We choose to observe either $L = 1, 2$, or 3 of the 5 state variables. Also we now double the length of our time series to $T = 8$ and so now $M = 340$ and the dimension of path space is $(M + 1)D + N_p = 341 \times 5 + 1 = 1706$. We increased the length of the time series so that the chaotic effects would be more apparent. We do the same kind of test as before by doing the minimization for 100 different initial paths. The results are shown in Fig. 4.7 for the case with $\beta = 2^{12}$.

For $L = 1$ observation we measure y_0, but expect to find similar results for any choice of one variable because of the symmetry of the model. We see that 14 out of the 100 initial paths end up at the global minimum. We also see that there are many other

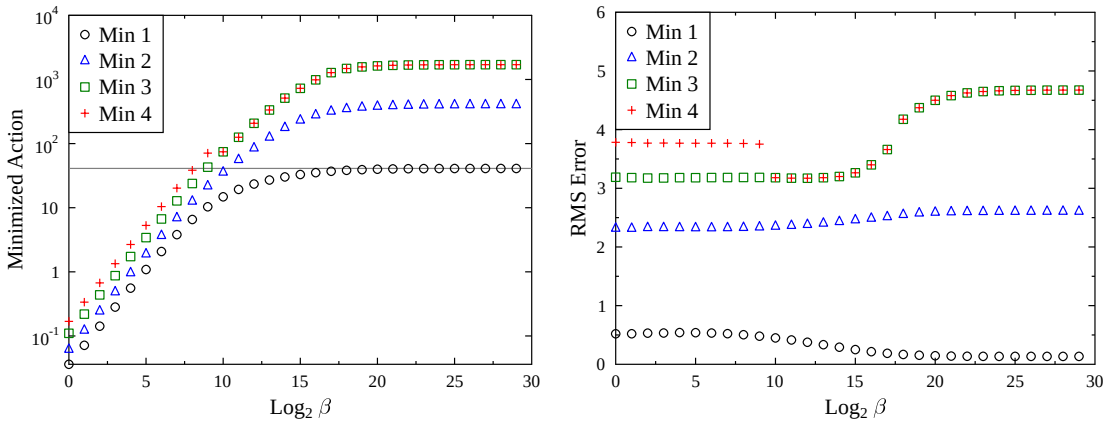

Figure 4.3: Minimization of the action following the four minima found at $\beta = 1$ up to $\beta = 2^{29}$. The four minima start out with action not much different, but the separation grows dramatically as β increases. The gray line indicates the expected level of the action assuming a perfect model and a measurement noise level of $\sigma = 0.5$.

minima that are close in action to the global minimum and cannot be neglected. The lowest minimum is at $A_0 \approx 22.0$ and the second lowest minimum is at $A_0 \approx 29.9$, which means global minimum path is about $\exp(29.9/22.0) \approx 3.9$ times as likely as the second minimum. This means that to evaluate the path integral we would need to sample paths from many different regions of path space, instead of just in the vicinity of the global minimum. This would be hard or impossible to do, and the result for the mean path would probably not be very predictive since the likely paths are not localized in path space. Note that it is still possible to do a state and parameter estimation if we are only interested in the maximum likelihood path (global minimum path), which is very close to the true path in this case, and it gets even closer (the RMS error decreases to around 0.2) as β is increased.

We then add observations of an additional variable. For Lorenz 96 $D = 5$ there are only two distinct possibilities: either measure two variables with adjacent indices (y_l and y_{l+1}), or measure two variables with a skip of one (y_l and y_{l+2}). Recall that the state variables are arranged on a ring, so $l = 4$ is adjacent to $l = 0$. We see these two possibilities in Fig. 4.7 for observing y_0, y_1 or observing y_0, y_2 (the results of the other possible ways of choosing two variables are similar). In both cases the ratio between the the 2nd lowest minima and the lowest minima is about 2, so now the global minimum is roughly 10 times more likely than the next minimum. This is an increase from the $L = 1$ case. If β was increased this ratio would become larger and we could ignore all

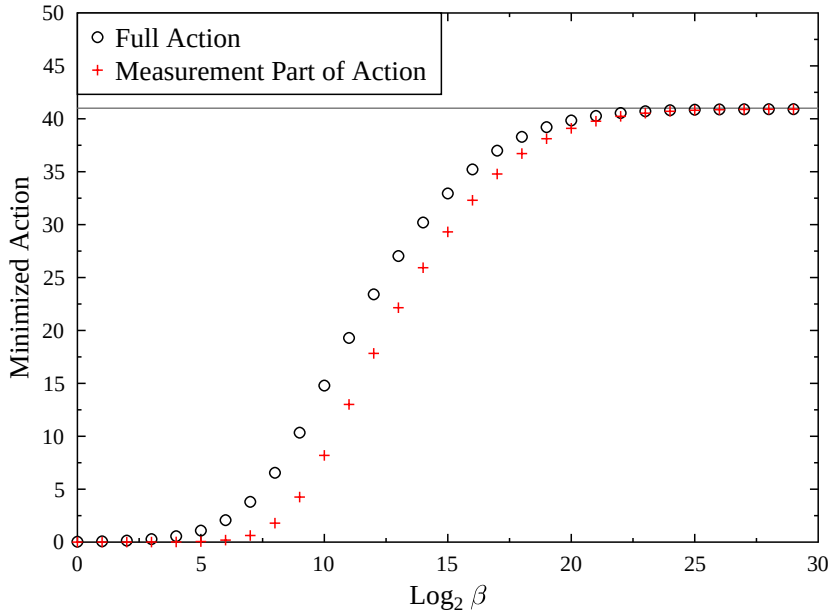

Figure 4.4: Minimized full action and measurement part of the action as a function of β. The dynamics part of the action goes to zero when β is large because in this case the model is perfect. The measurement part of the action goes to the level expected given that the measurement noise level is $\sigma = 0.5$ (gray line).

paths except those in the vicinity of the global minimum. We also see that measurement adjacent variables seems to work better, meaning it is easier to find the global minimum: 86 of the 100 paths end up at the global minimum versus 31 out of 100 for the other $L = 2$ configuration.

If we add observations of one more variable, so now $L = 3$, the shape of the action becomes very simple. Now all 100 initial paths go to the same global minimum. This means we only need to consider paths in the vicinity of the global minimum. There are two ways of choosing three variables to observe, either y_0, y_1, y_2 (shown) or y_0, y_2, y_3 (not shown) and both give similar results.

We can also compare all the paths we found by minimization to the true path by calculating the RMS error averaged over all observed and unobserved variables. The true path does not include the measurement noise. Of course in a real experimental application the true path is not known, but here it is useful to look at as a diagnostic.

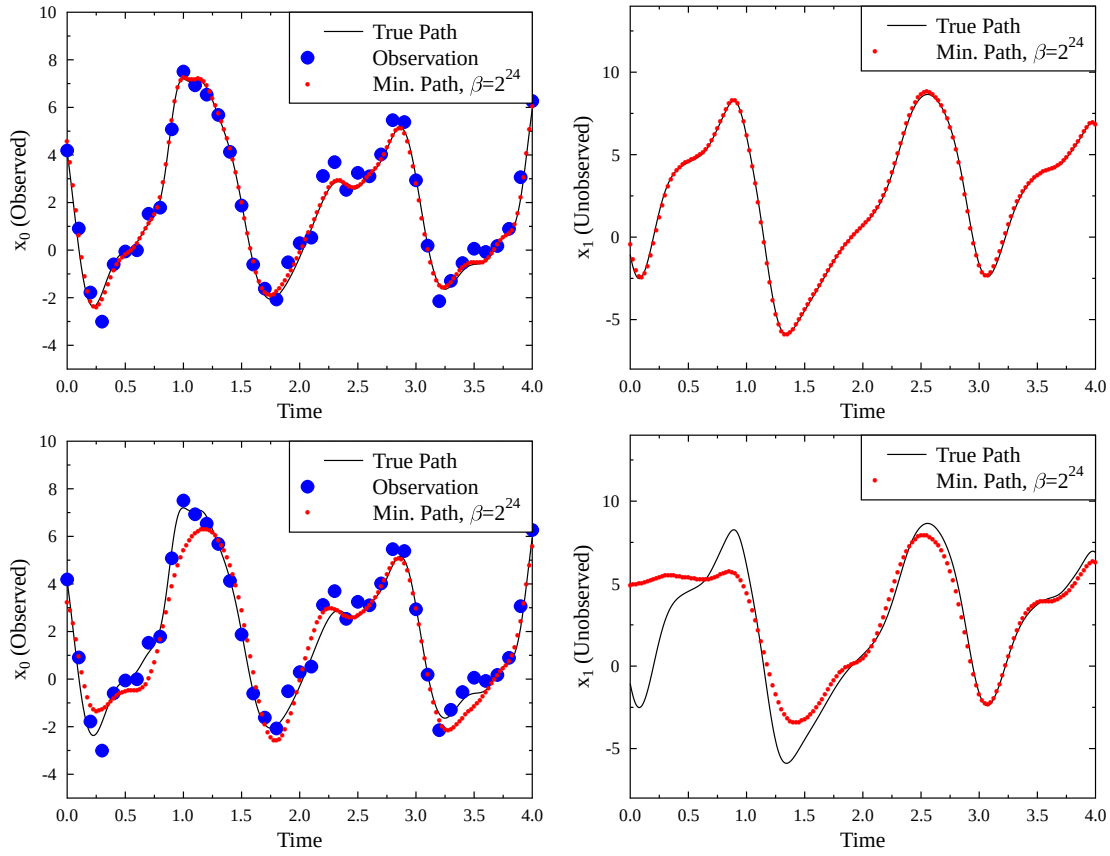

Figure 4.5: The same situation as in Fig. 4.2 but with $\beta = 2^{24}$. This shows how the global minimum path approaches the true path when β becomes large, but the non-global minimum path goes further away from the true path. The parameter values are: $f = 8.20$ (top) and $f = 7.37$ (bottom).

We should expect the global minimum paths in each case to be close to the true path when β is large, since we know the model is exact in this case. The bottom graph of Fig. 4.7 shows that all $14 + 31 + 86 + 100 = 231$ of the paths that ended up at the global minimum, for any of the choice of observations, all do about equally as well in matching the true path with an RMS error of about 0.3. This means if we were only interested in the maximum likelihood path any of the measurement choices would work equally well. The difference is that the global minimum is easier to find for $L = 3$ than for $L = 2$, and easier for $L = 2$ than for $L = 1$. In this case the most likely path is essentially the the true path, and that becomes even more true at larger β.

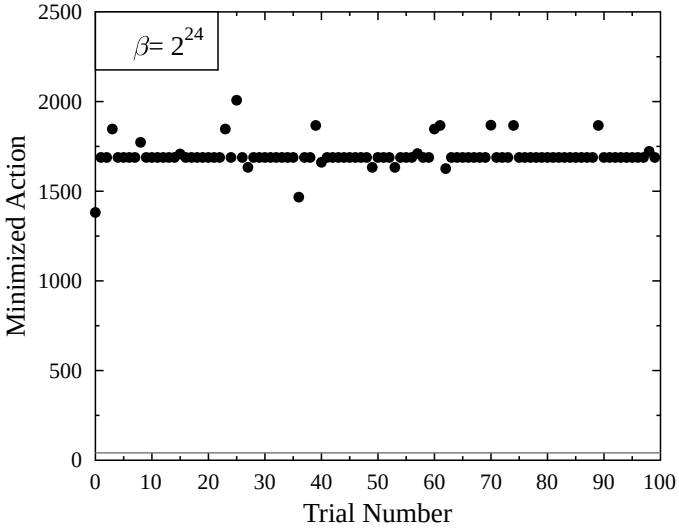

Figure 4.6: Minimization of $A_0(\mathbf{X})$ starting from 100 different initial paths, with $\beta = 2^{24}$. The global minimum at $A_0 \approx 41$ (gray line) previously found by gradually increasing β is not located. This shows that it is important to the minimization in several steps.

4.2.3 Effect of Model Error

We examine the effect of error in the model equations. We only consider a particular type of error that can be represented as a Gaussian white noise forcing term added onto the model equations. We do this by allowing the forcing parameter to fluctuate in time. We choose a new set of values for the forcing parameters at each integration time step labeled by n and of size Δt_{int} according to

$$f_l(n) = f_0 + \sqrt{\frac{2\Gamma}{\Delta t_{int}}} N_{ln}(0,1),$$

where $l = 0, 1, \ldots, D-1$, and $f_0 = 8.17$ is the mean forcing parameter. We examine three different noise levels $\Gamma = 0.0625, 0.125, 0.25$, and compare to the deterministic case where $\Gamma = 0$. The relation between R_f in the action and the noise level Γ in the model differential equations is

$$R_f \Delta t = \frac{1}{2\Gamma},$$

where Δt is the discretization time step used in the action. We generate simulated noisy measurements of $y_0(n)$ and $y_2(n)$ with noise level $\sigma_m = 0.5$ in the same way as before.

We then do the same type of calculation as before where the action is minimized for $\beta = 2^0, 2^1, \ldots, 2^{29}$ in order. We only look at the maximum likelihood solution found

at $\beta = 1$ and follow it up to larger β. The action found in this procedure is shown in Fig. 4.8, and the RMS error is shown in Fig. 4.9. We see that when noise is added to the model the optimal RMS error occurs approximately where $\beta R_f = \frac{1}{2\Gamma}$. As β is increased beyond this point the fit gets worse because the paths are forced to satisfy the model equation too precisely, and this does not work since the true path was not generated with the exact model.

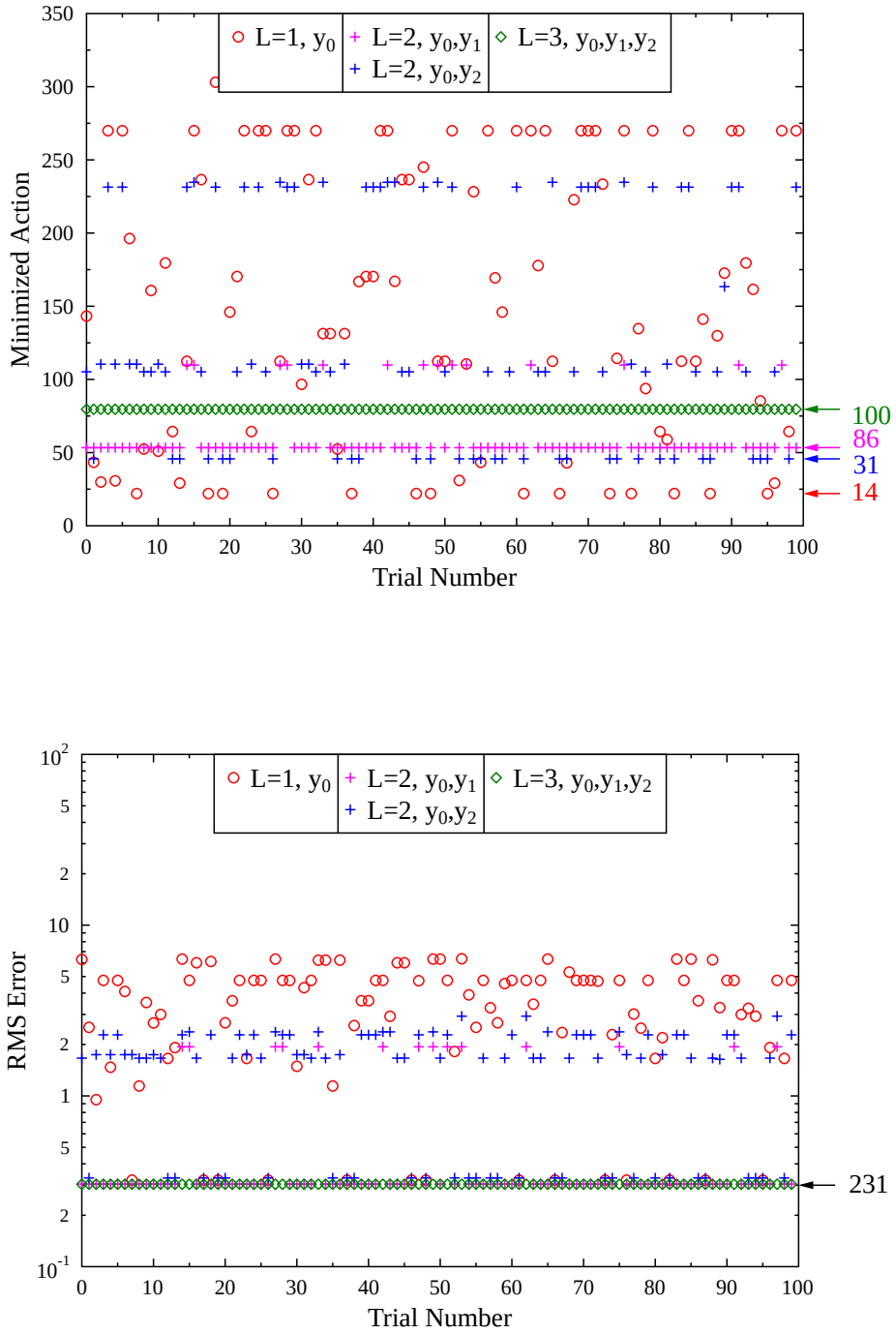

Figure 4.7: Minimization of $A_0(\mathbf{X})$ for different choices of observations, $\beta = 2^{12}$, $T = 8$, showing the action (top) and the RMS error (bottom). The arrows on the side indicate the global minima and the numbers are the number of initial paths that arrived at the global minimum. All the global minimum paths are closer to the true path than any of the other solutions, and they are all equally as close to the true path irrespective of the number of observations.

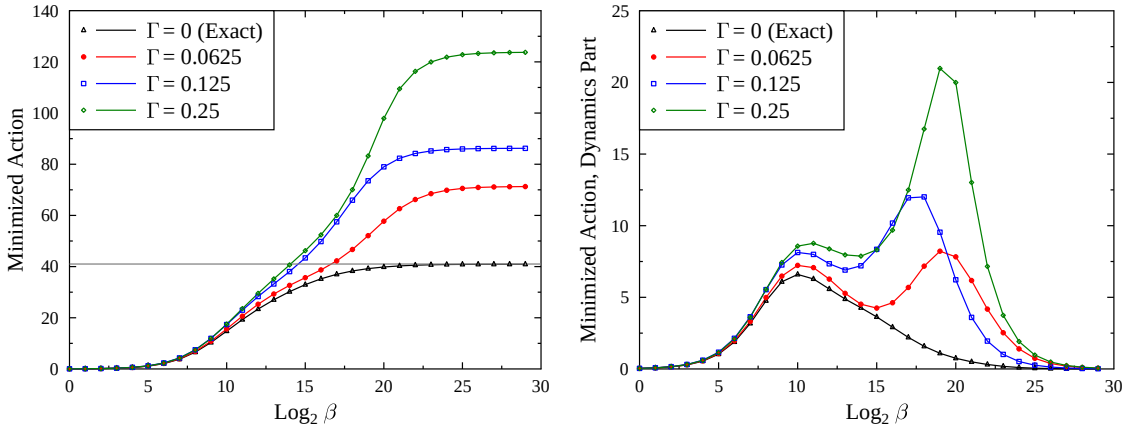

Figure 4.8: The action is minimized for different values of β for data generated with various noise levels. The full action is shown on the left and just the dynamics part of the action is on the right. The gray line indicates the expected action for data generated with an exact model with measurement noise of $\sigma_m = 0.5$.

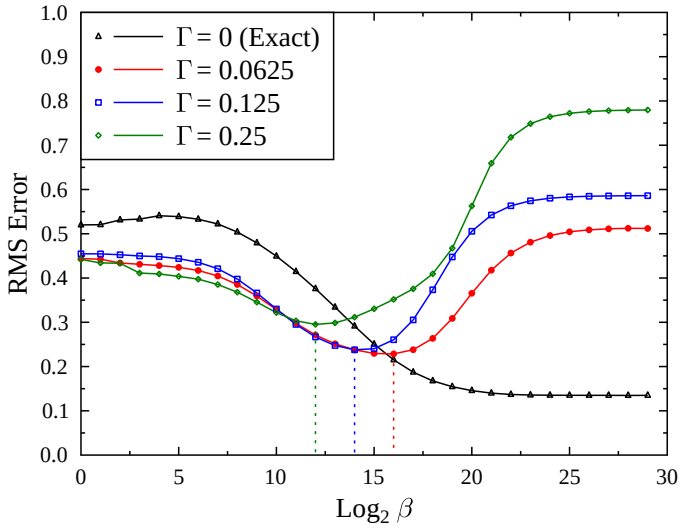

Figure 4.9: The action is minimized for different values of β for data generated with various noise levels. The RMS error is calculated by comparing the maximum likelihood path to the true path. The dotted lines indicate the values of β where the RMS error is optimal.

4.3 Quadratic Approximation Near Minimum

We are interested in the shape of the action surface $A_0(\mathbf{X})$. More specifically, we would like to find the global minimum and ask what shape the action has in its vicinity. The shape tells us the uncertainty of state estimates along different directions in path space: if the curvature is very flat in a particular direction then there is large uncertainty associated with that direction. The curvature in every direction is non-negative, since by assumption we are considering a minimum. For convenience we refer to the components of the path $\mathbf{X}$ as x_i for $i = 0, 1, \ldots, N-1$. The components of the path represent the states at every time step and the parameters.

Approximate the action near a minimum located at $\mathbf{X}_0$ as

$$A_0(\mathbf{X}) \approx A_0(\mathbf{X}_0) + \frac{1}{2}\Delta\mathbf{X}^T \cdot \mathbf{H} \cdot \Delta\mathbf{X},$$

where the Hessian $\mathbf{H}$ is $N \times N$ real symmetric matrix with components $[\mathbf{H}]_{ij} = \frac{\partial^2 A_0(\mathbf{X}_0)}{\partial x_i \partial x_j}$, and $\Delta\mathbf{X} = \mathbf{X} - \mathbf{X}_0$ is the deviation from the minimum. We then diagonalize $\mathbf{H}$ using $\mathbf{H} = \mathbf{P}^T \cdot \mathbf{D} \cdot \mathbf{P}$. The matrix $\mathbf{P}$ is made up of the set of orthogonal eigenvectors $\mathbf{\Psi}_i$, and $\mathbf{D}$ is a diagonal matrix with eigenvalues λ_i as entries. The eigenvalues represent the curvature along the set of special orthogonal directions (principle axes) in path space defined by the eigenvectors. We can check that all $\lambda_i > 0$ to see that we are in fact dealing with a minimum. If there is some continuous transformation in path space which leaves the action invariant then there will be a zero eigenvalue, and then the minimum is not well-defined.

The quadratic term in the action becomes

$$\begin{aligned}\Delta\mathbf{X}^T \cdot \mathbf{H} \cdot \Delta\mathbf{X} &= (\mathbf{P} \cdot \Delta\mathbf{X})^T \cdot \mathbf{D} \cdot (\mathbf{P} \cdot \Delta\mathbf{X}) \\ &= \sum_{i=0}^{N-1} \lambda_i (\Delta\mathbf{X} \cdot \mathbf{\Psi}_i)^2.\end{aligned}$$

The first moment of the projection of $\Delta\mathbf{X}$ onto the the nth eigenvector is

$$\langle \Delta\mathbf{X} \cdot \mathbf{\Psi}_n \rangle = \frac{1}{Z} \int d\mathbf{X} (\Delta\mathbf{X} \cdot \mathbf{\Psi}_n) \prod_{i=0}^{N-1} \exp[-\frac{\lambda_i}{2} (\Delta\mathbf{X} \cdot \mathbf{\Psi}_i)^2] = 0,$$

where Z is for normalization, and the second moments are

$$\langle (\Delta\mathbf{X} \cdot \mathbf{\Psi}_n)(\Delta\mathbf{X} \cdot \mathbf{\Psi}_m) \rangle = \frac{1}{Z} \int d\mathbf{X} (\Delta\mathbf{X} \cdot \mathbf{\Psi}_n)(\Delta\mathbf{X} \cdot \mathbf{\Psi}_m) \prod_{i=0}^{N-1} \exp[-\frac{\lambda_i}{2} (\Delta\mathbf{X} \cdot \mathbf{\Psi}_i)^2] = \frac{\delta_{nm}}{\lambda_n}.$$

Now we would like to find the the moments of the projection of $\Delta\mathbf{X}$ along the original coordinate system with basis vectors $\{\hat{\mathbf{x}}_n\}$, because that coordinate system has its axes aligned with the model space. For an arbitrary vector $\mathbf{v}$

$$\mathbf{v} \cdot \hat{\mathbf{x}}_n = \sum_i (\mathbf{v} \cdot \mathbf{\Psi}_i)(\mathbf{\Psi}_i \cdot \hat{\mathbf{x}}_n).$$

We can use this fact to get

$$\langle \Delta\mathbf{X} \cdot \hat{\mathbf{x}}_n \rangle = \sum_i \langle \Delta\mathbf{X} \cdot \mathbf{\Psi}_i \rangle (\mathbf{\Psi}_i \cdot \hat{\mathbf{x}}_n) = 0, \tag{4.4}$$

$$\begin{aligned} \left\langle (\Delta\mathbf{X} \cdot \hat{\mathbf{x}}_n)^2 \right\rangle &= \left\langle \sum_i (\Delta\mathbf{X} \cdot \mathbf{\Psi}_i)(\mathbf{\Psi}_i \cdot \hat{\mathbf{x}}_n) \sum_j (\Delta\mathbf{X} \cdot \mathbf{\Psi}_j)(\mathbf{\Psi}_j \cdot \hat{\mathbf{x}}_n) \right\rangle \\ &= \sum_i \frac{(\mathbf{\Psi}_i \cdot \hat{\mathbf{x}}_n)^2}{\lambda_i}. \end{aligned} \tag{4.5}$$

We can use these results to calculate the RMS variation

$$\sigma_n = \sqrt{\sum_{i=0}^{N-1} \frac{(\mathbf{\Psi}_i \cdot \hat{\mathbf{x}}_n)^2}{\lambda_i}} \tag{4.6}$$

This means we can compute an estimate of the RMS variation for each component of the path by numerically calculating the matrix $\mathbf{H}$ at a particular minimum, and then finding its eigenvalues and eigenvectors. We also may find out about symmetries in the model by computing the eigenvalue spectrum and looking for a zero eigenvalue. Small eigenvalues may also indicate that the minimum is not very well-defined.

4.3.1 Eigenvalues and Eigenvectors of the Hessian

Next we examine the curvature of the action in the vicinity of the global minimum $\mathbf{X}_0(\beta)$ for the same example of Lorenz 96 $D = 5$ that we have already found in the previous section. To do this we calculate the Hessian matrix $\mathbf{H}(\mathbf{X}_0(\beta))$ by approximating the second derivatives using a centered finite difference of the analytic first derivatives.

The eigenvalues and eigenvectors of $\mathbf{H}(\mathbf{X}_0(\beta))$ are then found using the GNU Scientific Library (GSL) routines for finding eigenvalues and eigenvectors of real symmetric matrices. The routines use symmetric bidiagonalization and QR reduction [24]. The eigenvalues are sorted so that $\lambda_0 \leq \lambda_1 \leq \ldots \lambda_{N-1}$.

In Fig. 4.10 we see the eigenvalue spectra for several values of β. We see that for small β there is a group of 82 eigenvalues with value of approximately 4. This is

because we have provided 82 measurements and set $R_m = 4$. In the limit of $\beta = 0$ the only terms in the action have the form $\frac{R_m}{2}(y-z)^2$, so this makes sense. The associated eigenvectors are aligned with the measurement directions. We see that as the model influence is turned up the spectrum becomes continuous, and the the 82 components of the path which have measurements are no longer isolated. The measurement directions now get mixed in with all the other eigenvectors.

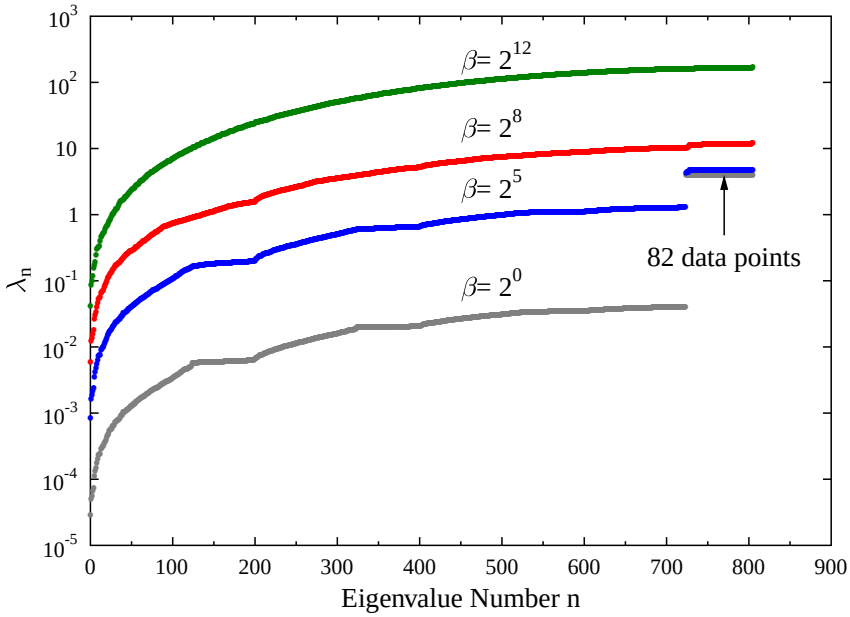

Figure 4.10: Sorted eigenvalue spectra for various values of β. For $\beta < 2^8$ there is a distinct group of 82 eigenvalues with value $\lambda = R_m = 4$ that correspond to the 82 data points provided. Once $\beta > 2^8$, they get mixed in with the other eigenvalues.

In Fig. 4.11 we track the smallest and largest eigenvalues, λ_0 and λ_{N-1}. The most significant thing seen here is that as β gets large λ_0 becomes constant. This means that at least one direction in path space space will continue to have uncertainty associated with it even as we get to the $\beta \to \infty$ limit of no noise in the model.

An even more interesting effect is shown in Fig. 4.12. Here we focus on the 11 lowest eigenvalues. The associated eigenvectors represent the most uncertain directions in path space. We see that as β becomes large six special eigenvectors $\mathbf{\Psi}_0, \ldots, \mathbf{\Psi}_5$ are

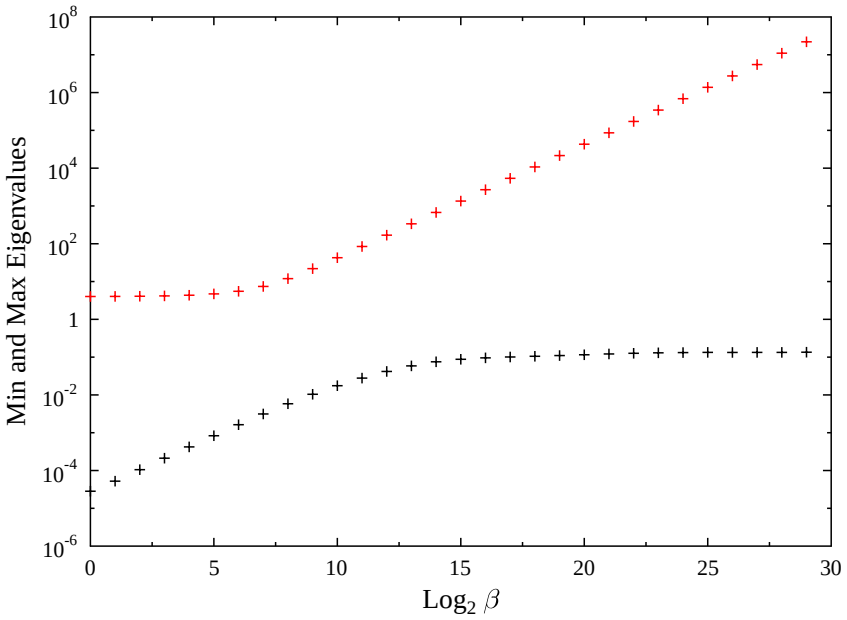

Figure 4.11: Minimum eigenvalue λ_0 and maximum eigenvalue λ_{N-1} as a function of β. The minimum levels off at around 0.1, indicating that there are at least some uncertain directions even as the dynamical noise becomes low. The maximum continues to grow proportional to β at large β.

singled out. As β increases further the gap between λ_5 and λ_6 continues to grow, while $\lambda_0, \ldots, \lambda_5$ converge to values between $\lambda_0 \approx 0.13$ and $\lambda_5 \approx 0.75$. This means that in the limit of no error in the model, only these six uncertain directions remain (some of them shown in Fig. 4.13). In this case since $D = 5$ and $N_p = 1$ six quantities are all the information that is needed to completely specify the trajectory in the deterministic limit.

4.4 Comparison of Mean and Maximum Likelihood Paths

We have discussed two ways of estimating the true path: the maximum likelihood path and the mean path. We try them both out on the same problem of Lorenz 96 $D = 5$ with $R_f = 0.01 \times 2^{12} = 40.96$ and $R_m = 4$ with x_0 and x_2 measured. We find the maximum likelihood path by minimizing $A_0(\mathbf{X})$. We also find the mean path

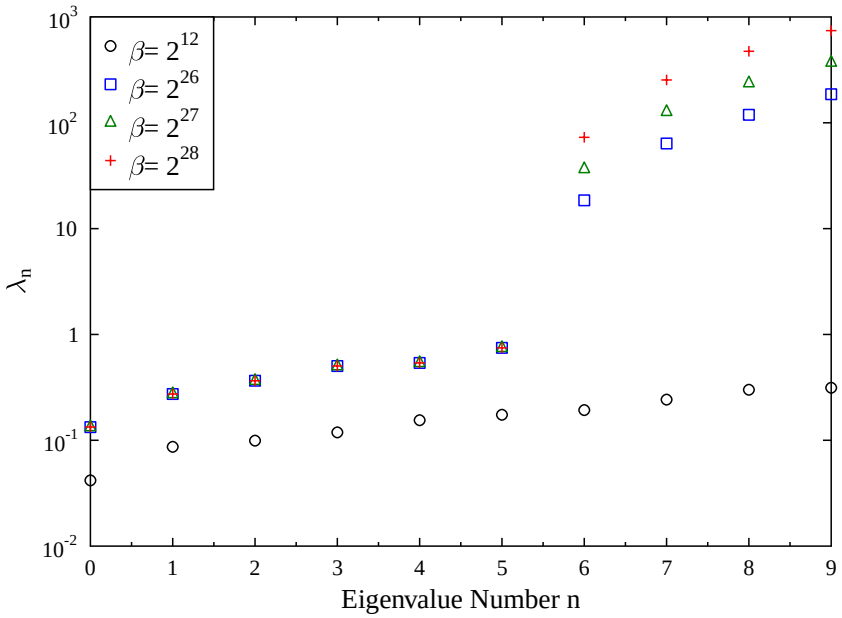

Figure 4.12: The eleven smallest eigenvalues of the Hessian of $A_0(\mathbf{X})$. These are important because they indicate the most uncertain directions. For large β six eigenvalues separate from the rest by a significant amount and converge to the values shown. The corresponding eigenvectors represent six special directions, $\mathbf{\Psi}_0, \ldots, \mathbf{\Psi}_5$, which remain uncertain as β becomes large. This is because in the deterministic limit six numbers (five initial conditions and one parameter) completely specific the trajectory.

for the same problem using the PIMC method, as described in the previous chapter, to approximate $\langle \mathbf{X} \rangle$. The two resulting paths are displayed in Fig. 4.14 and are compared to the true path. We see that the maximum likelihood path and the mean path are almost the same, consistent with $P(x_{n,l}|\mathbf{Y})$ being Gaussian, as was suggested in Chapter 3.

We saw in the previous chapter how to use PIMC to calculate the RMS variation around the mean path to use as an estimate of uncertainty. Now we calculate the RMS variation by calculating the Hessian of the action evaluated at the minimum path and finding its eigenvalues and eigenvectors as described in the previous section. Then the RMS variation is found using Eq. (4.6). The results of using these two different methods are shown in Fig. 4.15 for two different values of β. We see that the two methods agree very closely for $\beta = 2^{12}$ and less so for $\beta = 2^{20}$. This indicates that the quadratic

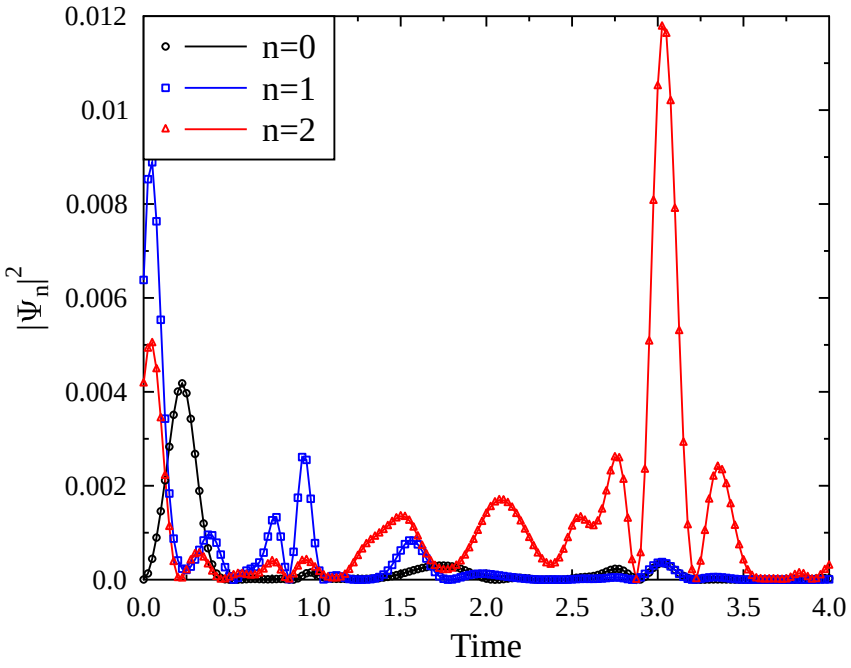

Figure 4.13: Some examples of the special eigenvectors $\mathbf{\Psi}_0, \mathbf{\Psi}_1, \mathbf{\Psi}_2$ for $\beta = 2^{28}$ representing the three most uncertain directions in path space. Only showing the $l = 0$ components.

approximation of $A_0(\mathbf{X})$ gets worse as β gets larger, meaning the contribution from higher order terms in $\mathbf{\Delta X}$ become greater. This is expected, because as β increases the influence of the nonlinear terms in the model compared to the quadratic measurement terms increases.

We also compare the mean path and the maximum likelihood path (minimum path) using the data from the real Colpitts oscillator circuit. The mean path was found in the previous chapter using PIMC. The maximum likelihood path was found by minimizing the same action function which was used in the PIMC method. The results for one of the unobserved state variables, $V_{CE}(t)$, are shown in Fig. 4.16, and once again both paths match closely. Also shown is the actual recorded time series, which was not actually used as data, but only for comparison with the state estimates.

It is also useful to consider the dynamics term in the action separately from the full action, which includes the measurement error term also. The dynamics term is a sum of model violation g_n at each time step. Recall that the model violation is defined

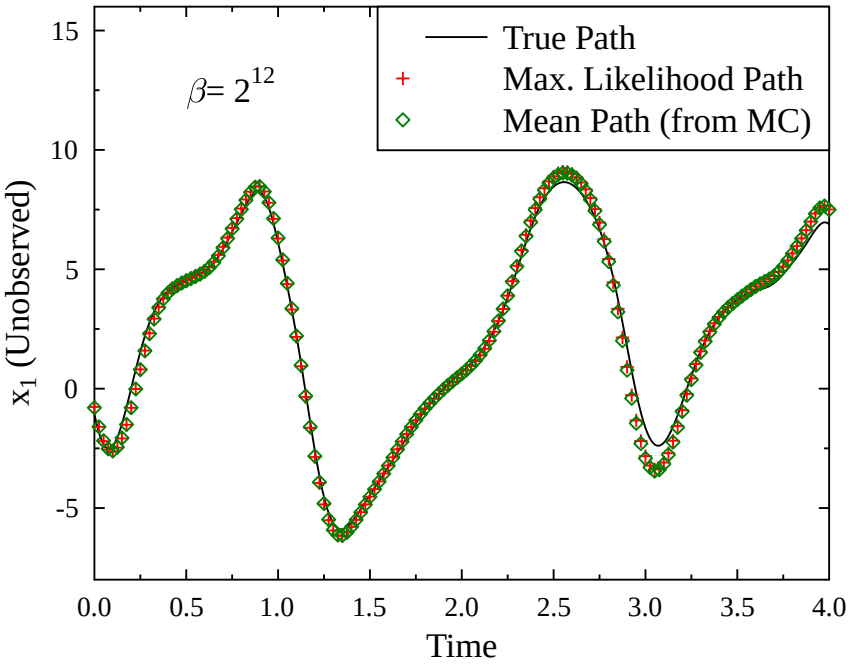

Figure 4.14: Comparison of the maximum likelihood path found by minimizing $A_0(\mathbf{X})$ and the mean path found by PIMC approximation of $\langle \mathbf{X} \rangle$, for one of the unobserved variables of the Lorenz 96 $D = 5$ problem.

as

$$g_{n,l} = x_{n,l} - x_{n+1,l} - \frac{\Delta t}{2} \left[F_l(\mathbf{x}_n) + F_l(\mathbf{x}_{n+1}) \right],$$

and it says how much the model is violated at each time step $n = 0, 1, \ldots, M-1$ by a particular path. In Fig. 4.17 we plot $g^2_{n,E}$ evaluated at the minimum path as found by minimizing $A_0(\mathbf{X})$ for the Colpitts oscillator problem. The subscript E refers to the V_E state variable. The top graph is from the real data which is not exactly described by the model, and the bottom graph was from data simulated using the exact model. If we only looked at the top graph we may have come to the conclusion that the real circuit behavior deviates from the model the most when $V_E(t)$ is at its extreme minimum values. This is when the currents going into the base and the collector of the transistor are at a maximum, and it seems reasonable that the simplified Ebers-Moll transistor model would be inaccurate here. However, the bottom graph shows very similar model violation also with spikes at the minima of $V_E(t)$. The model violation in the simulated case comes only from numerical errors: the simulated data was generated using RK4 with a time step of 0.01 so it is a very accurate solution to the differential equations,

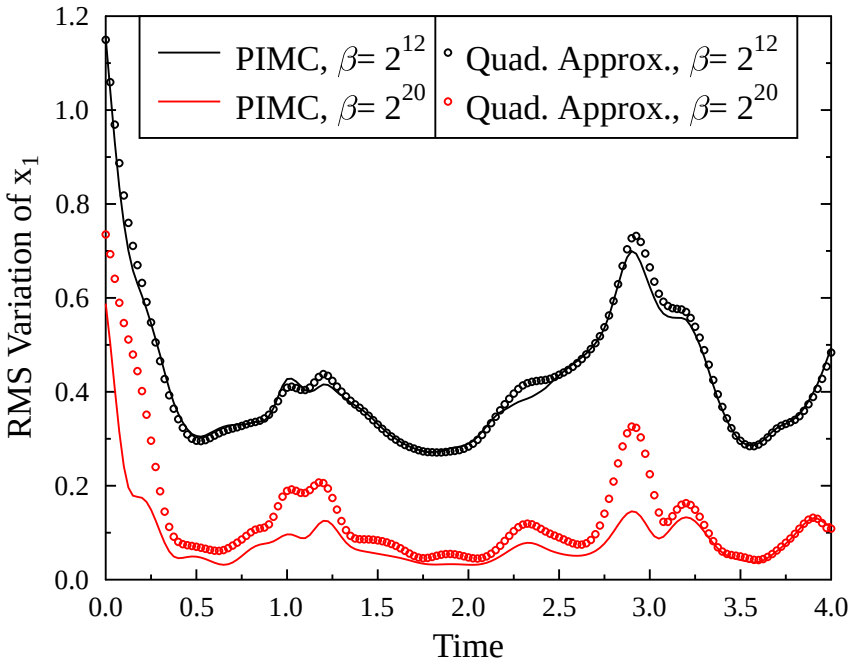

Figure 4.15: Comparison of the RMS variation around the mean found by PIMC with the RMS variation around the maximum likelihood path as found by using a quadratic approximation of $A_0(\mathbf{X})$ around the minimum for $\beta = 2^{12}$ and $\beta = 2^{20}$.

but the model violation term in the action uses a less accurate integration method (2nd order Adams-Moulton) and has a larger time step of $\Delta t = 0.1$. So we should expect model violation to be largest when the RHS of the differential equations is largest, and this happens when $V_E(t)$ is small.

The model violation term can also be thought of as a model correction term: the correction may be needed because of inaccuracy in the implicit integration rule, inaccuracies in the model, or to provide small stabilizing corrections when the model is chaotic. In the case where the inaccuracy is in the model, the model violation may tell us where the model is inadequate, and it may be useful in comparing several candidate models.

4.5 Summary

We saw that it is useful to locate minima of $A_0(\mathbf{X})$, especially the global minimum, and to examine the curvature of $A_0(\mathbf{X})$ at the minima. The shape of the surface

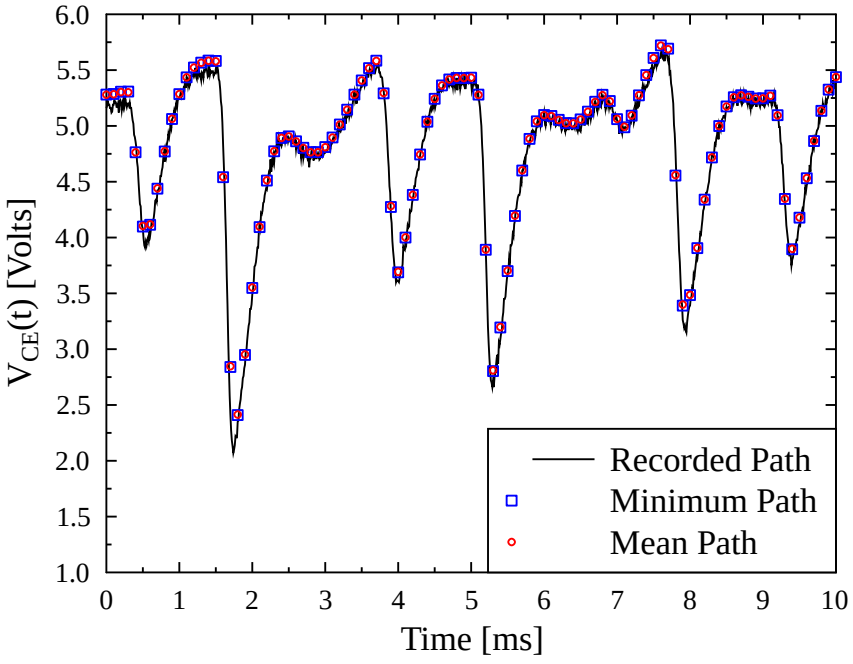

Figure 4.16: Comparison of the mean path found by PIMC and the maximum likelihood path found by minimizing $A_0(\mathbf{X})$ for the Colpitts circuit $V_{CE}(t)$.

of $A_0(\mathbf{X})$ tells us when the state estimation problem can be done and give useful results. If the action has many local minima, or very flat valleys, then it will be impossible to find a region of likely paths that is localized in path space, which is the goal of state estimation. Making additional observations can make the shape of the action nicer, with a single well-defined minimum. When that happens then the region of likely paths is confined to the vicinity of the global minimum.

We also saw that the global minimum may be difficult to find in chaotic models. Even if it is in a deep valley of the action, which means it is it the only relevant region in path space, it may be very narrow. This problem can be solved by gradually turning up the influence of the dynamics term in the action. This way the action is initially dominated by the smooth measurement terms. The hope is that a search in path space will get to the correct region near the global minimum, and then be trapped there as the valley becomes deeper and more narrow as β increases. This applies to doing minimizations of $A_0(\mathbf{X})$ as discussed in this chapter and to doing PIMC calculations as discussed in the previous chapter.

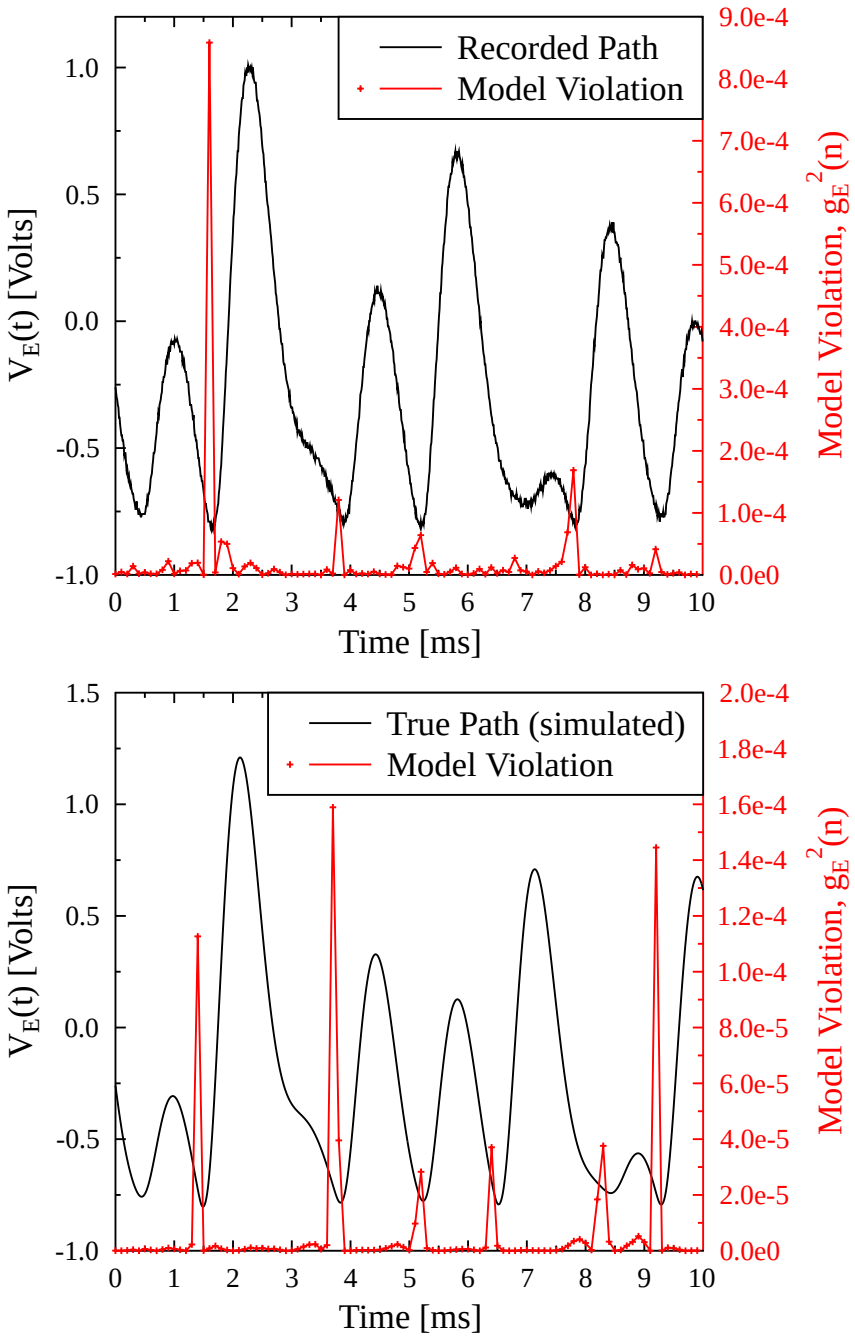

Figure 4.17: The model violation term squared, $g^2_{n,E}$, for the Colpitts circuit (top) and for simulated data using the Colpitts model equations (bottom). Also shown is the real and simulated $V_E(t)$.

Chapter 5

Alternative Methods

In this chapter we review some other methods of data assimilation and compare them to the PIMC method.

5.1 Particle Filters

The starting point for particle filters [32, 7, 13], which are sequential Monte Carlo methods, is the same as the first few steps shown in Chapter 2. The steps are repeated here as a review and to make all the assumptions clear. We use the Chapman-Kolomogrov equation and the assumption that the model is Markov to write

$$P(\mathbf{x}_n|\mathbf{y}_{1:n-1}) = \int d\mathbf{x}_{n-1} P(\mathbf{x}_n|\mathbf{x}_{n-1}) P(\mathbf{x}_{n-1}|\mathbf{y}_{1:n-1}). \tag{5.1}$$

This says how to use the model to evolve the state forward one time step. We also use Bayes rule in the form of Eq. (2.1) to write

$$P(\mathbf{x}_n|\mathbf{y}_{1:n}) \propto P(\mathbf{y}_n|\mathbf{x}_n, \mathbf{y}_{1:n-1}) P(\mathbf{x}_n|\mathbf{y}_{1:n-1}), \tag{5.2}$$

which tells us how to incorporate the new piece of measurement information $\mathbf{y}_n$.

The idea of particle filtering is to approximate the filtered distribution at each time step, $P(\mathbf{x}_n|\mathbf{y}_{1:n})$, using an ensemble of N discrete 'particles' with positions $\mathbf{x}_n^i$ and associated weights w_n^i, where $i = 1, 2, \ldots, N$ is the particle label. If we have a set of particle positions and associated weights to approximate the state distribution at time $n-1$, then we can write

$$P(\mathbf{x}_{n-1}|\mathbf{y}_{1:n-1}) = \sum_{i=1}^{N} w_{n-1}^i \delta(\mathbf{x}_{n-1} - \mathbf{x}_{n-1}^i).$$

Now the goal is to evolve the particle's positions and weights using the model via Eq. (5.1) and to incorporate the new measurement $\mathbf{y}_n$ via Eq. (5.2) to get an approximation of $P(\mathbf{x}_n|\mathbf{y}_{1:n})$. To do this, first we apply Eq. (5.1) to the particle representation of $P(\mathbf{x}_{n-1}|\mathbf{y}_{1:n-1})$ to get

$$P(\mathbf{x}_n|\mathbf{y}_{1:n-1}) = \sum_{i=1}^{N} w_{n-1}^i P(\mathbf{x}_n|\mathbf{x}_{n-1}^i).$$

We then apply Eq. (5.2) to get

$$P(\mathbf{x}_n|\mathbf{y}_{1:n}) \propto P(\mathbf{y}_n|\mathbf{x}_n, \mathbf{y}_{1:n-1}) \sum_{i=1}^{N} w_{n-1}^i P(\mathbf{x}_n|\mathbf{x}_{n-1}^i). \tag{5.3}$$

We would like to represent $P(\mathbf{x}_n|\mathbf{y}_{1:n})$ with particles $\mathbf{x}_n^i$ and associated weights w_n^i in the same as we we did at time step $n-1$, so we need a procedure for updating the particle positions and weights. To do this we use the idea of importance sampling.

Importance sampling [38] says that if we want to approximate a distribution $P(x)$ we can do this by drawing N samples $\{x^i\}$ from another distribution $Q(x)$, called the importance density, which is easier to sample from. Then we can approximate $P(x)$ as

$$P(x) \approx \sum_{i=1}^{N} w^i \delta(x - x^i),$$

where the weight is

$$w^i \propto \frac{P(x^i)}{Q(x^i)}. \tag{5.4}$$

The weights must then be normalized so that $\sum_i w^i = 1$. Note that this means we do not need to sample from $P(x)$, we just need to evaluate $P(x^i)$ and $Q(x^i)$ for each sample x^i drawn from $Q(x)$.

Going back to the formulation of particle filtering, the distribution we want to approximate is $P(\mathbf{x}_n|\mathbf{y}_{1:n})$. To do this, we need to choose an importance density to draw particles $\mathbf{x}_n^i$ from. A simple and intuitive choice, but certainly not the only choice, is to draw $\mathbf{x}_n^i$ from $P(\mathbf{x}_n|\mathbf{x}_{n-1}^i)$. Then by plugging Eq.(5.3) into the numerator of Eq.(5.4) and this choice of importance density into the denominator of Eq.(5.4) we get

$$w_n^i \propto \frac{P(\mathbf{y}_n|\mathbf{x}_n^i, \mathbf{y}_{1:n-1}) \sum_{j=1}^{N} w_{n-1}^j P(\mathbf{x}_n^i|\mathbf{x}_{n-1}^j)}{P(\mathbf{x}_n^i|\mathbf{x}_{n-1}^i)}.$$

Since particles do not change labels $P(\mathbf{x}_n^i|\mathbf{x}_{n-1}^j) \propto \delta_{ij}$, and so the sum over particles goes away. Also because of the good choice of importance density the denominator cancels

one of terms in the numerator, and we are left with

$$w_n^i \propto w_{n-1}^i P(\mathbf{y}_n|\mathbf{x}_n^i, \mathbf{y}_{1:n-1}). \tag{5.5}$$

So now we have a simple and intuitive procedure for evolving the particles and weights: update each particle's position by drawing $\mathbf{x}_n^i$ from $P(\mathbf{x}_n|\mathbf{x}_{n-1}^i)$ and then update each particle's weight with Eq. (5.5). One additional step is needed to normalize the weights.

In practice drawing $\mathbf{x}_n^i$ from $P(\mathbf{x}_n|\mathbf{x}_{n-1}^i)$ typically means that the particle positions are evolved using the dynamical model, and then the positions are perturbed by noise drawn from a random number generator. Also it is typically assumed that the measurements only depend on the current state and so $P(\mathbf{y}_n|\mathbf{x}_n, \mathbf{y}_{1:n-1}) = P(\mathbf{y}_n|\mathbf{x}_n)$, which is simply the measurement noise distribution. A common assumption is that the noise is Gaussian, and so

$$P(\mathbf{y}_n|\mathbf{x}_n) \propto \exp\left[-\frac{1}{2}(\mathbf{y}_n - \mathbf{x}_n)^T \cdot \mathbf{R_m} \cdot (\mathbf{y}_n - \mathbf{x}_n)\right],$$

where $\mathbf{R_m}$ is the inverse of the measurement noise correlation matrix.

A common problem with particle filtering is that very few of the particles end up with most of the weight while most of the particles have negligible weight. This issue can be at least partially solved by using resampling algorithms [32, 7]. There many ways to do this, but the basic idea is to occasionally throw away particles with small weight, duplicate particles with large weight, and then recalculate all the weights.

One remaining issue is how to start the whole process at time step $n = 0$. A prior distribution $P(\mathbf{x}_0)$ must be chosen to draw the initial particle positions $\mathbf{x}_0^i$ from. A typical choice is that the $P(\mathbf{x}_0)$ is uniform over some range of state values and zero outside of this range. This means that the filtered distributions $P(\mathbf{x}_n|\mathbf{y}_{1:n})$ will typically start out spread out and become more localized at later times due to the guidance of the measurements. The large initial spread could be problematic when there are many unobserved states and parameters with very uncertain values. Then the state space that needs to be sampled at early times, $n = 0$ in particular, may be very large and require very many particles to adequately cover. This problem may be partially solved by starting at some later time $n > 0$ and working backward to $n = 0$. Then use the resulting particle ensemble at $n = 0$ as the starting point to move forward to the end of the observation window.

In contrast, the path integral Monte Carlo (PIMC) approach naturally includes all of the measurements over the whole observation window. So the states already 'know

where they are going' in addition to knowing where they have been. This is particularly useful at early times. Also the assumption of a uniform $P(\mathbf{x}_0)$ is handled very easily: it simply means the $\log P(\mathbf{x}_0)$ term can be dropped from the equation for $A_0(\mathbf{X})$ because it is a constant. In PIMC the paths $\mathbf{X}^{(i)}$ distributed according to $P(\mathbf{X}|\mathbf{Y}) \propto \exp[-A_0(\mathbf{X})]$ which are generated with Metropolis MC (or other methods) play a very similar role as the particles in the particle filtering approach. In PIMC the MC paths $\mathbf{X}^{(i)}$ are used to approximate the distribution $P(\mathbf{X}|\mathbf{Y})$ as

$$P(\mathbf{X}|\mathbf{Y}) \approx \frac{1}{N}\sum_{i=1}^{N} \delta(\mathbf{X} - \mathbf{X}^{(i)}).$$

These MC sample paths can then be used to calculate things of interest, such as the mean path and RMS variation around the mean. This is very similar to particle filtering, except the central objects are the MC sample paths $\mathbf{X}^{(i)}$ which represent the the entire time evolution of the state distribution, instead of the particles $\mathbf{x}_n^i$ which represent the state distribution at a given time step.

5.2 Various Kalman Filters

The Ensemble Kalman Filter (EnKF) is another data assimilation technique similar to the particle filtering method just discussed [16]. It is also a sequential Monte Carlo method, and it can be formulated within the same Bayesian probability framework. The difference is that in EnKF it is assumed that the filtered distribution $P(\mathbf{x}_n|\mathbf{y}_{1:n})$ is Gaussian, and this allows the computations to be made more efficient since only the mean and covariance need to be tracked. If the filtered distribution is not actually Gaussian, it will be effectively treated as Gaussian with the same mean and variance as the actual distribution. This can lead to bad estimates in the cases where $P(\mathbf{x}_n|\mathbf{y}_{1:n})$ is far from Gaussian, as shown in [18] with the example of a bimodal distribution.

When the dynamics is nonlinear it should be expected that $P(\mathbf{x}_n|\mathbf{y}_{1:n})$ will be non-Gaussian, even if $P(\mathbf{x}_0)$ is Gaussian. However, we saw in the Lorenz 96 example from Chapter 3 that the measurements tend to keep the filtered state distributions close to Gaussian. The strength of this effect will depend on many things such as the time between measurements, the degree of nonlinearity in the model, and the measurement noise amplitude and distribution. A similar regularizing effect of the measurements was noted in [18]. This suggests that EnKF may be successful in many cases even with

highly nonlinear models, if there are enough measurements available to confine and guide the filtered distributions. As we saw in Chapter 3, the assumption that $P(\mathbf{x}_n|\mathbf{y}_{1:n})$ is Gaussian can be checked using PIMC methods to calculate moments beyond the second moment.

The original Kalman filter [28] was derived with the assumptions that the dynamical model is linear, and that the measurement and dynamical noise is Gaussian and uncorrelated. Also the prior distribution $P(\mathbf{x}_0)$ must be Gaussian and the measurement function must be linear. If these serious assumptions are satisfied then the Kalman filter provides the optimal state estimate very efficiently, but this is rarely the case in real problems.

The Extended Kalman filter (EKF) is the application of the original Kalman filter to the case where the model is nonlinear. This is done by using a linearized version of the model equations in the same framework as the original Kalman filter. This can be problematic because the linearization may cause instabilities which are not physical [17].

5.3 Methods for Deterministic Dynamics

There are several state and parameter estimation methods for the case where the model is deterministic. This can be thought of as the limit where $R_f \to \infty$ in the action. This means that the only paths that are considered are solutions to the deterministic model equations. Then the goal is to find one particular solution to the model equations which minimizes the error between the model output and the data. This solution serves as the maximum likelihood estimate of the state, and the parameters used to generate the solution serve as the maximum likelihood estimate of the parameters.

5.3.1 Optimization over Initial Conditions and Parameters

One of the most obvious ways to attempt to do this is to formulate the problem as a minimization problem over the space of initial conditions and parameters. In this approach the thing to be minimized is the squared error between the data and the model output:

$$C(\mathbf{x}_0, \mathbf{p}) = \frac{1}{N} \sum_{n=1}^{N} \left[x_1(n\Delta t; \mathbf{x}_0, \mathbf{p}) - y_1(n\Delta t) \right]^2 ,$$

where $x_1(t; \mathbf{x}_0, \mathbf{p})$ is one component of the result of integrating the model with initial conditions $\mathbf{x}_0$ and parameters $\mathbf{p}$, and $y_1(t)$ is a scalar measurement time series. We assume for simplicity that the measurement is simply a projection of the state vector onto the $\hat{\mathbf{x}}_1$ direction, but this could easily be generalized to a L-dimensional measurement vector that is some function of the D-dimensional state vector. The goal is to minimize the cost $C(\mathbf{x}_0, \mathbf{p})$ by finding an optimal $\mathbf{x}_0$ and $\mathbf{p}$ which produces a model output that is as close to the data as possible. If the model is exact then the minimum of the cost should be roughly σ_m^2, the variance of the measurement noise.

There are several derivative-free optimization algorithms could be used for this purpose, such as Nelder-Mead or Brent's method [51, 8], all which require many evaluations of the cost for many different values of $\{\mathbf{x}_0, \mathbf{p}\}$, but no evaluation of derivatives. Each evaluation of the cost requires an integration to produce the model output, so it may be computationally intensive, but there is a more serious problem with this approach if the model is chaotic.

The output of a chaotic model is very sensitive to the initial conditions and parameters. This means that if even just one of the parameters or initial conditions is slightly wrong the data and model trajectories will diverge and the cost function will be large. Thus the cost will only be small in a very small region of the $\{\mathbf{x}_0, \mathbf{p}\}$ space around the true value and it will be very hard for any minimization algorithm to locate. The problem gets even worse as $T\lambda_{max}$ becomes larger, where $T = N\Delta t$ is the length of the time series and λ_{max} is the largest Lyapunov exponent. Another related reason why the global minimum may be hard to locate is because the surface of the cost function may form a very rough surface with many local minima. An example of this is shown in Fig. 5.1 which was calculated using the the Colpitts oscillator model and $V_E(t)$ data from the actual circuit. This plot shows the cost only as a function of one the parameters, R, with all the other parameters and initial conditions set at optimal values. We see that when the time series is 10 ms or longer local minima appear, making the search difficult.

One obvious solution to this problem is to use a time series that is short compared to $1/\lambda_{max}$, however this may not always be adequate. A better solution is to use the idea of synchronization of chaos, which was first reported by Pecora and Carrol [46]. The idea of synchronization, which was first discovered in pendulum clocks by Huygens in the 17th century, is that if some form of coupling is introduced between two similar systems then the systems may become synchronized [49]. In modern language this means

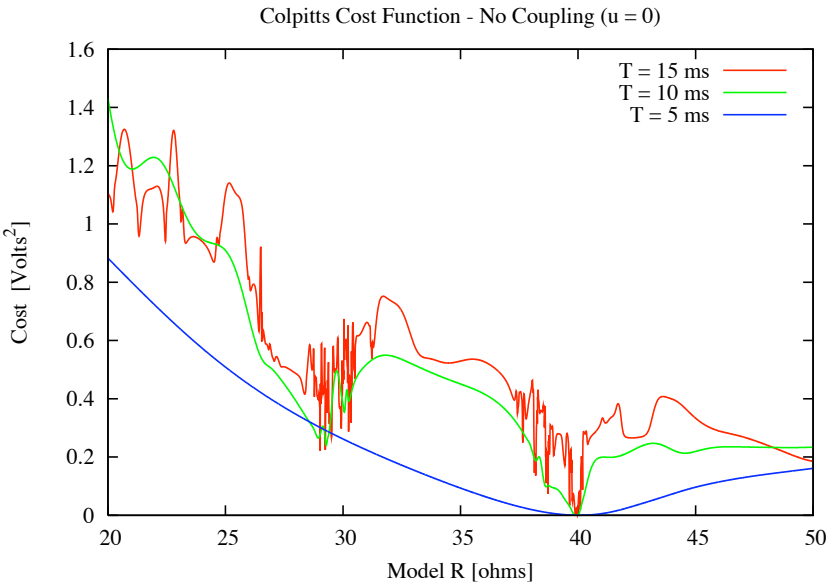

Figure 5.1: The cost $C(\mathbf{x}_0, \mathbf{p})$ for the Colpitts oscillator as a function of just one parameter R, shown for different lengths of time series. As the time series gets longer the minimum gets harder to find.

that the phase space trajectories of the two system converge. To apply this idea to our problem we think of the data as a driver system which is unidirectionally coupled to a response system– the dynamical model. In practice this can be done by changing the model differential equations to

$$\frac{dx_1}{dt} = F_1(\mathbf{x}(t), \mathbf{p}) + u(y_1(t) - x_1(t))$$
$$\frac{d\mathbf{x}_\perp}{dt} = \mathbf{F}_\perp(\mathbf{x}(t), \mathbf{p})$$

where $\mathbf{x}_\perp$ refers to all of the components of the state vector except x_1, u is a positive coupling constant, $y_1(t)$ is the data time series, and $x_1(t)$ is the one component of the model output which is measured. When $u = 0$ this reduces to the original model which represents the true dynamics. When $u > 0$ the coupling term pushes the model toward the data. If the coupling constant is large enough and the model and data (actual) parameters are close enough, then the two systems will synchronize. This means that $\mathbf{x}(t) \approx \mathbf{y}(t)$, and in particular $x_1(t) \approx y_1(t)$, so the cost will be small. The effect of adding this coupling term is shown in Fig. 5.2, again for the Colpitts problem. Increasing the coupling constant eliminates the local minima, making the global minimum easier to find. However, it also flattens the cost function near the global minimum, making the

global minimum less well-defined. This is because if the coupling is large enough then $x_1(t)$ will be forced to follow $y_1(t)$ regardless of what $\mathbf{p}$ and $\mathbf{x}_0$ are.

This suggests that the minimization be done in several steps. Start with u large enough to eliminate local minima and do the minimization. Then to refine the parameter estimate, reduce u and do the minimization again starting from the same $\{\mathbf{x}_0, \mathbf{p}\}$ point, which should be close to the true optimal point.

Another approach is include the coupling constant in the optimization process and to modify the cost function by adding a penalty for u being large, such as $\eta^2 u^2$. The control parameter η sets the relative weight of the penalty for not matching the data and the penalty for having $u > 0$. There is a penalty for having $u > 0$ because unless $u = 0$ the differential equations do not represent the true dynamics: the coupling to the data was artificially introduced. Note that if the data and the model is exact then having a $u > 0$ does not change the model output once the optimal $\{\mathbf{x}_0, \mathbf{p}\}$ is found, but it does have a stabilizing effect.

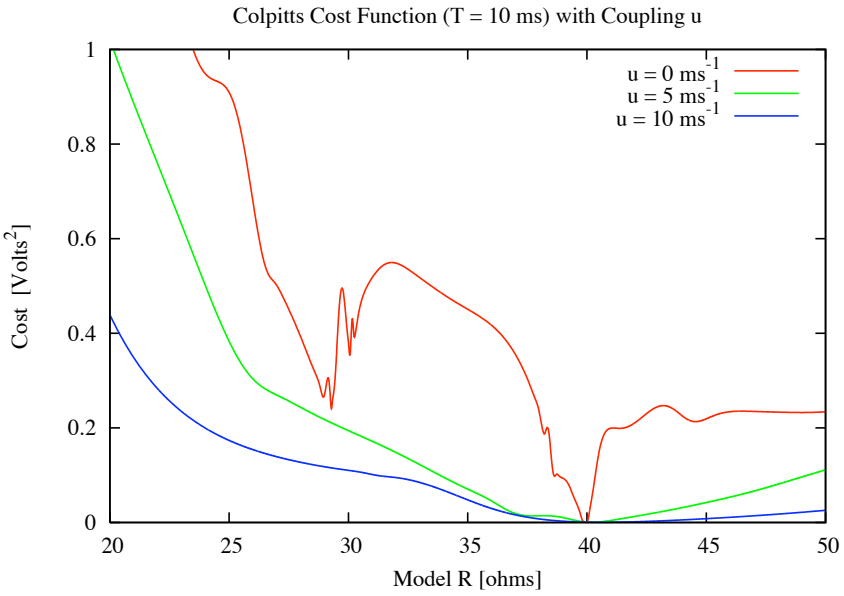

Figure 5.2: The cost $C(\mathbf{x}_0, \mathbf{p})$ for the Colpitts oscillator as a function of just one parameter R, shown for different coupling constants $u = 0, 5$, or 10 ms^{-1}.

A way to quantify the stabilizing effect of the coupling term is to calculate the conditional Lyapunov exponents (CLE) of the model system when driven by the data system [47, 2]. The largest CLE in particular is useful to know, because it gives the

approximate average rate at which the model and data trajectories converge ($\lambda_{max} < 0$) or diverge ($\lambda_{max} > 0$). The distance between the data and model trajectories is roughly

$$|\mathbf{x}(t) - \mathbf{y}(t)| \approx |\mathbf{x}(0) - \mathbf{y}(0)| e^{\lambda_{max} t}.$$

When $u = 0$ there is no coupling between the model and data systems and so λ_{max} is just the Lyapunov exponent of the model, which is positive for chaotic models. As u increases λ_{max} decreases, and once u becomes large enough λ_{max} becomes negative which means the data and model trajectories will converge. This is shown for the Colpitts example in Fig. 5.3.

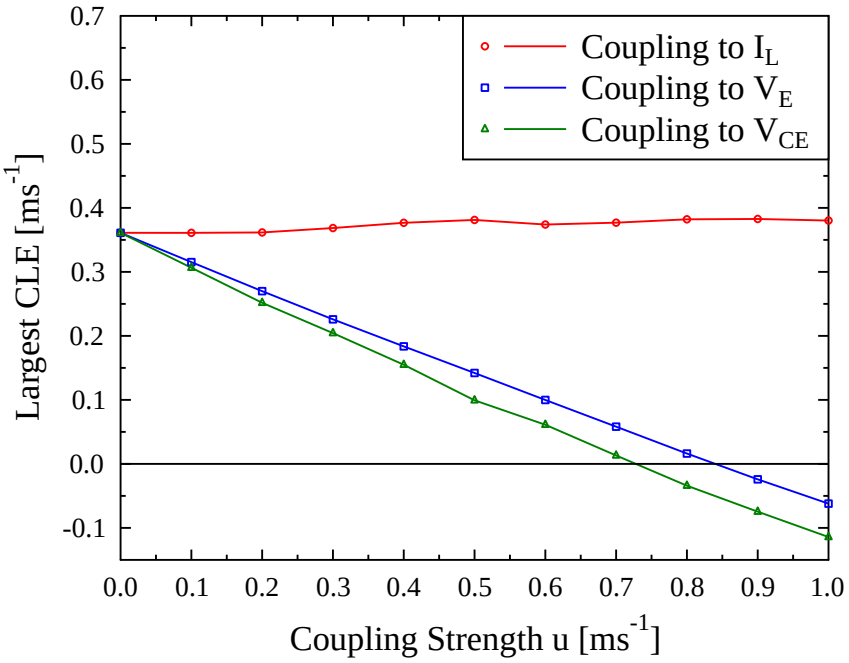

Figure 5.3: Largest CLE as a function of coupling strength for the Colpitts system. The coupling was made to each of the three state variables, one at a time.

5.3.2 Constrained Optimization

Another approach used in [12, 1, 54] is to move into the higher dimensional space $\{\mathbf{x}(0), \mathbf{x}(\Delta t), \ldots, \mathbf{x}(N\Delta t), \mathbf{p}\}$. So now the space being searched over is the space of state variables at every time step and the parameters. The model is introduced in the form of constraints, which come from discretizing the model differential equations, relating time-neighboring state vectors. The constraints effectively reduce the dimension of the search

down to the number of parameters plus number of initial conditions. The constrained optimization can be done using the SNOPT software package [23] which is based on a sparse sequential quadratic programming algorithm. Another variation used in this approach is to allow the coupling strength to vary in time, so now it is treated as an optimal control parameter $u(t)$ that is free to be adjusted by the optimizer. These extra degrees of freedom effectively loosen the constraints and increase the dimension of the search space. Once again the the coupling strength should go to zero at the end of the process.

One benefit of this approach is that the coupling strength $u(t)$ does not need to really go to exactly zero at all times at the end of the process. This allows for some deviation from the model, and this flexibility may be very useful for dealing with real data for which the model is not exactly known. In fact, something about model deficiencies, or about where integration errors are largest, may be learned by seeing where $u(t)$ ends up largest. The $u(t)$ terms plays a very similar role to the model violation term $g(t)$ discussed in Chapter 4. One difference is that typically $u(t)$ is forced to be positive because of its interpretation as a stabilizing term, but it may be useful to allow it to be negative as well: when the data is noisy the optimal correction to the state may not always be toward the data.

5.4 Summary

We reviewed several data assimilation methods besides PIMC. Of the alternative methods discussed here, particle filtering is the least restrictive. It is based on the same formulation as the PIMC method. The main difference is that particle filtering uses particles with associated weights to represent $P(\mathbf{x}_n|\mathbf{y}_{1:n})$, while PIMC uses paths, all with equal weight, to represent $P(\mathbf{x}_{0:M}|\mathbf{y}_{1:M})$. The advantage of the PIMC approach is that all the data $\mathbf{y}_{1:M}$ is used at each time point, and not just data from the past. Another advantage is the natural way which PIMC handles uninformative priors, which may be problematic for particle filters in problems with many unobserved states and parameters.

Chapter 6

Conclusion

The problem of data assimilation was formulated using path integrals, and then the path integrals were numerically approximated using a Markov chain Monte Carlo method to draw sample paths from $P(\mathbf{X}|\mathbf{Y}) \propto \exp[-A_0(\mathbf{X})]$. Each of the many sample paths can be thought of as a particular realization of the model evolution in a virtual experiment, where each virtual experiment produces the same observed time series $\mathbf{Y}$.

Path integral Monte Carlo is very similar to particle filtering, but it has the advantage of considering the entire data time series at once, instead of just the measurement at the current time point. Also broad initial condition distributions are very easily handled, in contrast to in particle filters, which must use a particle distribution at $t = 0$ to approximate $P(\mathbf{x}_0)$. If the initial distribution is broad, then very many particles may be necessary to sufficiently cover the important regions of $\mathbf{x}_0$ space.

The information theory interpretation of the measurement term in the path integral shows explicitly how the measurements provide information, which is used to guide the model states. It is essential that measurement data is used for guidance, otherwise chaos, noisy dynamics, and errors in the model, will quickly cause the model output to diverge from the data.

There is a benefit working in path space, as opposed to the space of initial conditions and parameters. The dimension of the space is much higher, but the effect of perturbing one component of the path is usually small, so the path can be gradually changed without changing the action very much. Working in path space comes about naturally when noise is introduced into the dynamics, increasing the degrees of freedom. Even if the actual dynamical noise level is zero, meaning the model is deterministic and

known exactly, noise can still be introduced as a way form of 'regularization' by saying the dynamics is more noisy than it actually is. The assumed noise level can then be gradually reduced during the minimization process or the Monte Carlo process, using a form of simulated annealing.

The shape of the action as a function of $\mathbf{X}$ tells us a lot about a data assimilation problem using a particular model, with a particular choice of measurements. One important question is: can the neighborhood of path space containing the true path be located? This can be tested by doing many minimizations of the action starting from many different locations in path space. If after minimizing, all the paths end up at the same location in path space, it is a good indication that the true path is near this location, and that the PIMC data assimilation method will be successful.

If instead the minimized paths end up scattered in different regions of path space, but with similar action, then more measurements may be required to help localize the likely regions of path space. If the minimized paths end up scattered in path space, but with one minimum that has a much lower action than all the others, then probably only the region around the lowest minimum needs to be considered, and PIMC will still be successful as long as the region of small action is visited.

Even if we can find the region of path space surrounding the true path, it is important to know the shape of the action in its vicinity. If there are some directions that are very flat, then the minimum is not very well-defined. To make it more well-defined we may have to make more measurements, or simplify the model. This information can be learned by looking at the RMS variation output of the PIMC process, when a path in the vicinity of the the minimum is used as a initial path.

A useful test to do using simulated data is to start out the PIMC process with an initial path equal to the true path. Since the data was simulated, the true path is already known, unlike in a real experiment. Then by examining the RMS variation around the minimum path, the uncertainty levels can be estimated. This will depend on the model, on which variables are measured, how frequently, and for how long, as well as the noise levels. On problems with an external drive signal, the results may also depend on the choice of drive signal. By doing this kind of test one can compare different measurement choices and find out which one will be most useful in reducing uncertainty.

Appendix A

User's Guide to *Carl*

A.1 Background

Carl is a program that does Monte Carlo data assimilation. Its purpose is to estimate unobserved states and parameters of a system, together with the associated uncertainties. The inputs into this method are a time series of measurements of state variables with uncertainties, and a model of the system in the form of a system ordinary differential equations. Model error and stochastic effects are represented by a Gaussian white noise term of a particular strength added onto each differential equation. These inputs are combined to form the action $A_0(\mathbf{Y})$ which is a function of the path $\mathbf{Y}$ [3, 53]. The path represents the discrete time history of all the state variables as well as any time-independent parameters $\mathbf{p}$. The probability density in path space conditioned on the time series of measurements $\mathbf{Z}$ is $P(\mathbf{Y}|\mathbf{Z}) \propto \exp[-A_0(\mathbf{Y})]$. This is used to compute moments of the components of the path

$$< y_l(n)^p >= \frac{\int d\mathbf{Y}([\mathbf{Y}]_{l,n})^p \exp[-A_0(\mathbf{Y})]}{\int d\mathbf{Y} \exp[-A_0(\mathbf{Y})]} \tag{A.1}$$

where the time index is $n = 0, 1, \ldots, M$, and the state variable index is $l = 0, 1, \ldots, K-1$, and $d\mathbf{Y}$ means integrate over all paths. The notation $([\mathbf{Y}]_{l,n})^a$ means select the component $y_l(n)$ from the path $\mathbf{Y}$ and raise it to the power a. This integral is approximated by generating a sequence of paths $\{\mathbf{Y}^{(1)}, \mathbf{Y}^{(2)}, \ldots, \mathbf{Y}^{(N_{path})}\}$ that are distributed according to $\propto \exp[-A_0(\mathbf{Y})]$. This takes care of the exponential weighting factor in the integral

and turns it into a sum

$$< y_l(n)^p > \approx \frac{1}{N_{path}} \sum_{i=1}^{N_{path}} ([\mathbf{Y}^{(i)}]_{l,n})^p. \tag{A.2}$$

There exist several algorithms for generating the sequence of paths with the correct distribution. *Carl* uses the Metropolis Monte Carlo method to do this [41, 42].

With assumptions of uncorrelated Gaussian noise in measurements and in the dynamics the action takes the form

$$\begin{aligned} A_0(\mathbf{Y}) = & \frac{1}{2} \sum_{n=0}^{M} \sum_{l=0}^{K-1} \mathrm{R^m}_l(n) \left(y_l(n) - \mathrm{z}_l(n)\right)^2 \\ & + \frac{\beta}{2} \sum_{n=0}^{M-1} \sum_{l=0}^{K-1} \mathrm{R^f}_l \left[y_l(n+1) - y_l(n) - \frac{\Delta t}{2} \left(F_l(\mathbf{y}(n+1), \mathbf{p}) + F_l(\mathbf{y}(n), \mathbf{p})\right) \right]^2 \\ & + \frac{1}{2} \sum_{l=0}^{K-1} \mathrm{R^{y0}}_l \left(y_l(0) - \mathrm{y^c}_l\right)^2 \\ & + \frac{1}{2} \sum_{l=0}^{N_p-1} \mathrm{R^p}_l \left(p_l - \mathrm{p^c}_l\right)^2 . \end{aligned} \tag{A.3}$$

We also make the assumption that the prior distributions $P(\mathbf{y}(0))$ and $P(\mathbf{p})$ are Gaussian, which is where the last two terms come from. The deterministic part of the model differential equation is $\frac{d\mathbf{y}}{dt} = \mathbf{F}(\mathbf{y}, \mathbf{p})$ which is entered into the `model.h` file. The constants $\mathrm{R^f}, \mathrm{R^m}, \mathrm{R^{y0}}, \mathrm{R^p}, \mathrm{y^c}, \mathrm{p^c}$ may all be specified in the file `specs.txt`, and the measurements $\mathrm{z}_l(n)$ are read in from separate files named `z.`l`.dat`, where $l \in \{0, 1, \ldots, K-1\}$. Since measurements are not typically available at every point in the (l, n) grid, the $\mathrm{R^m}_l(n)$'s associated with missing measurements are set to zero. The parameters are p_l where $l = 0, 1, \ldots, N_p - 1$, and they are considered part of the path $\mathbf{Y}$.

A.2 Overview of the process

Carl is complied by running `make` in the directory where the source code and the problem-specfic `model.h` files are located. A subdirectory should be created for each problem which contains all the data input files `z.`l`.dat`, any external drive files, and the settings file `specs.txt`. All of *Carl*'s output will be saved into this same problem subdirectory. See the sections below for details on the input and output files.

To run the program type, for example, '`./carl ./Lor96K5`'. The command line argument is the path of the problem subdirectory, in this case it is the path where the

Lorenz 96 example files are located. The program will then read the file `specs.txt` from the problem subdirectory which tells it which other data files to read, and all the settings for the calculation.

The path is stored in an array `y[N]` with size $N = (M+1)K + N_p$. The array is indexed as follows: the state $y_l(n)$, where $n \in \{0, 1, \ldots, M\}, l \in \{0, 1, \ldots, K-1\}$, has index $i = nK + l$, and the parameter p_l, where $l \in \{0, 1, \ldots, N_p - 1\}$, has index $i = (N - N_p) + l$.

The path is initialized to the guess values given for each state and parameter in the `specs.txt` file. Each state will be initially set to be constant in time. Alternatively, the entire initial path may be loaded from a file. This is useful for starting a new run at same location in the N-dimensional path-space that a previous run finished at, or for starting at the true path as a test, in twin experiments.

Then the Metropolis Monte Carlo process begins, which will generate a sequence of paths that are distributed according to $\propto \exp[-A_0(\mathbf{Y})]$. The pseudo-code in Fig. 3.1 shows the Metropolis Monte Carlo algorithm, in this example it is only used to calculate the mean of each component of the path, but *Carl* actually computes the first four moments of each component of the path. Each iteration through the outer loop is called a "sweep" and corresponds to one path in the sum of Eq. (A.2). There are `NIt` total sweeps, but the first `Ninit` of them are discarded during the initialization phase. It is necessary to discard the initial sweeps, because the path may start out far from equilibrium.

In the actual program, the sweeps are grouped into "blocks" of `Bsize` number of sweeps, mainly to decrease the amount of data that needs to be stored. The total number of blocks generated is `Nblock`, and the total number of sweeps is `NIt = Nblock*Bsize + Ninit`. `Nblock` should be a multiple of 24 for the re-blocking error calculation to work properly (see below).

Statistics are averaged over each block, and are stored in a 2-dimensional array `block[4*N][Nblock]`. The first index identifies the particular quantity that is being averaged, and the second index identifies the block. If i is the index of the particular component of the path in question, then $i + (p-1) * N$ is the index of the pth moment of that component. This is the indexing system that is used in the `#define PRINT_BLOCK` statement to select particular moments of particular components to write to the `blocks.dat` file. It is important to view the blocked averaged to see if the path

has equilibrated, and to get an idea of the number of sweeps needed to eliminate correlations. The ability to calculate errors in the MC evaluation by re-blocking the blocks is built into *Carl*. The idea is that if the blocks are large enough, adjacent blocks should not be correlated.

The random walk step size `delta[i]` may be different for each component of the path. *Carl* automatically updates each `delta[i]` during the initialization phase after each block is completed, to aim for a target acceptance rate of 50% for each component of the path. The rule used is

$$\Delta_i \leftarrow \Delta_i \left(1 + \alpha(f_{accept} - 0.5)\right), \tag{A.4}$$

where f_{accept} is the acceptance rate over the block that has just finished, and $\alpha = 0.3$. The idea is to decrease Δ_i if the acceptance rate is $< 50\%$ and increase Δ_i is the acceptance rate is $> 50\%$. The purpose of this is to make the step sizes large enough to sufficiently explore the the important regions of path-space (in a finite number of sweeps), but not so large that most of the proposed steps are rejected. It is important to have different Δ_i for each component of the path, because the size of the likely regions in path-space may be very different along different directions in path-space. For the MC procedure to be valid, Δ_i should stop changing before the initialization phase is complete.

After `NIt` number of paths are generated the MC process is complete. Then *Carl* averages the blocked averages to get one value the first four moments of each component of the path. From these $4N$ values, the mean, standard deviation, skewness, and kurtosis are calculated for each component of the path, and these are written to the output files `mc.dat`, `mc3.dat`, `mc4.dat` (see the section on output files for details). Diagnostics may be calculated by comparing to the true path in the file `x.dat` if available.

At this point, error calculations may be done by combining the blocks into larger blocks [42]. Consider a particular quantity of interest $x = y_l(n)^a$, or $x = p_l^a$. This quantity takes on a series of P values, $\{x_0, x_1, \ldots, x_{P-1}\}$, one for each path generated. The paths are grouped into B blocks of size $S = P/B$. Here we assume S, P, B are all integers. The blocked averages of x are

$$\bar{x}_j = \frac{1}{S} \sum_{n=jS}^{jS+S-1} x_n, \tag{A.5}$$

where $j = 0, 1, \ldots, B-1$ identifies the block. The average over all blocks is

$$\bar{x} = \frac{1}{B} \sum_{j=0}^{B-1} \bar{x}_j. \tag{A.6}$$

The variance of the blocked quantities is

$$var[\bar{x}] = \frac{1}{B(B-1)} \sum_{j=0}^{B-1} (\bar{x}_j - \bar{x})^2. \tag{A.7}$$

This is useful to calculate, because $\sqrt{var[\bar{x}]}$ is the RMS error in the MC calculation of the quantity x, *but only if the block averages are uncorrelated.* Since the Metropolis Monte Carlo method is based on a random walk, it may take many sweeps to eliminate correlations in x_i. The idea is that once the block size S becomes large enough, the correlations are averaged out, and so the correlation among the blocked averages $\bar{x}_j$ is eliminated. This suggests plotting $\sqrt{var[\bar{x}]}$ as a function of S. For small S the error will be underestimated, but when S becomes large enough $\sqrt{var[\bar{x}]}$ should level off at the true error level. *Carl* calculates $\sqrt{var[\bar{x}]}$ for $S = r$`*Bsize`, where $r = 1, 2, 3, 4$, for the specific x's that are selected using the `#define PRINT_BLOCK` statement. This is why `Nblock` should be a multiple of 24.

A.3 Details of the Inputs

A.3.1 Model file

A file named `model.h` has to be located in the main *Carl* directory. It contains the model equations in the function `double g(int l, int n, double *y , double *u, double *z, double *p, double inj)`. This function should return the value of $F_l(\mathbf{y}(n), \mathbf{p})$. The current values of the states at the current time step can accessed with `y[0], y[1],...,y[K-1]`. The current values of the parameters can be accessed with `p[0]`, `p[1]`, ..., `p[NP-1]`. See Fig. A.1 for an example.

For normal usage the `z[]` and `u[]` are not needed, but they are available to the user anyway inside the `g(...)` function. The measurements at the current time step can be accessed using `z[0],z[1],...,z[K-1]`, but beware that the missing measurements should not be used, since they will artificially have a value of zero. The value of `RmMask[n*K+l]` will be 1 if the point (l, n) has a measurement associated with it, and 0 otherwise. If controls are activated, then the current value of controls at the current time step can be accessed with `u[0],u[1],...,u[K-1]`. The controls that do not have measurements associated with them will be set to zero, and so the term `+ fabs(u[l])*(z[l] - y[l])` will have no effect unless a measurement is present.

If an external drive file has been loaded (see the next section), then the value of the drive signal at the correct time step will be passed in the `inj` argument of `g(...)`. If no drive file is loaded `inj = 0`.

The header file also contains the number of equations `K`, the number of time steps `M`, and the number of parameters `NP`. If this file is changed the code needs to be recompiled by using `make`. Figure A.1 shows an example set up for the Lorenz 96 model.

Also there are some optional define statements that activate certain features when present. The line `#define COUP_GEN` activates the automatic calculation of the coupling matrix when included. This is useful for speeding up problems with many state variables that are sparsely coupled (such as Lorenz 96 with large `K` since each variable is only coupled to three neighbors). This causes a speed up because it reduces the number of terms in the action that need to be computed to calculate changes in the action. If it is not included the coupling will be assumed to be all-to-all (but still local in the time direction).

The statement `#define PRINT_BLOCK` can be included followed a list of any number of indices (see Fig. A.1). This will select particular moments of particular components of the path to be output to the file `blocks.dat`. This will also cause the re-blocking error calculation for these same quantities to be output to the file `error.dat`.

Diagnostics can be turned on by including the statement `#define TSCORE`. This will read the true path from the file `x.dat`, and compute the t-scores, $T_l(n) = (\langle y_l(n) \rangle - x_l(n))/\sigma_l(n)$, of each component of the path. The t-scores are put into 30 bins of ranging from $T = -3$ to $T = 3$, and written to the file `t-hist.dat`.

A.3.2 Problem specification file

The file named `specs.txt` must be located in the the problem subdirectory. Each line is a command which is comprised of a one or two lowercase letter key (which needs to start in the first column of the line) followed by a number of entries which are integers, floats, or strings, all tab or space delimited. The order of the commands does not matter, except `dt` should come first, and they do not need to all be present. Table A.3.2 shows all the possible commands and all the possible options for each. Commands numbers 6 through 13 all activate certain functionality which is turned off by default. The entries shown in parenthesis can be left out and will be set to default values, for instance if ${\mathrm{R}^{\mathrm{y0}}}_l$ or ${\mathrm{R}^{\mathrm{p}}}_l$ is left off they will be set to zero, which means the the prior terms

```
// Example model.h file
// Number of variables
#define K 5
// Number of time steps
#define M 100
// Number of parameters
#define NP 1
// Total number of dimensions
#define N   ((M+1)*K + NP)
//Selects the 1st and 2nd moment of the parameter for output
#define PRINT_BLOCKS {N-1, N-1+N}
#define COUP_GEN
int RmMask[N];
double coup;
//////////////////////////////////////////////////////////////////////
// Put the Differential Equations (dy_l/dt =  ) here
// Variables are y[0], ..., y[K-1],  Parameters are p[0],...,p[NP-1]
// z[l] and u[l] are the measurement and control terms at the current
// time step n.  u's without associated z's are set to zero
// inj is the drive signal at the current time step
//////////////////////////////////////////////////////////////////////

double g(int l, int n, double *y, double *u
         , double *z, double *p, double inj){
    // Lorenz 96
   return  y[(l+K-1)%K] * (y[(l+1)%K] - y[(l+K-2)%K]) - y[l] + p[0];
}
```

Figure A.1: Example `model.h` file for Lorenz 96

in Eq. (A.3) will have no effect (or in other words, the distribution is uniform from $-\infty$ to ∞).

Table A.1: List of options that can be entered in `specs.txt`.

Description	Key					
1. Time step	`dt`	Δt				
2. MC Sweeps	`mc`	`Ninit`	`Bsize`	`Nblock`		
3. Measurement	`z`	l	$\mathrm{R^m}_l$	`(Ndata)`	(t_0)	
4. State variable	`y`	l	$\mathrm{R^f}_l$	guess	$(\mathrm{y^c}_l)$	$(\mathrm{R^{y_0}}_l)$
5. Parameter	`p`	l	-	guess	$(\mathrm{p^c}_l)$	$(\mathrm{R^p}_l)$
(6) State Bound	`yb`	l	$y_{l,min}$	$y_{l,max}$		
(7) Param. Bound	`pb`	l	$p_{l,min}$	$p_{l,max}$		
(8) RK4 Prediction	`rk`	`Nrk`	`(rkSub)`			
(9) Sim. Annealing	`sa`	`Ncool`	β_0			
(10) External Drive	`in`	`Fname`	(t_0)			
(11) Constant coup.	`co`	`coup`				
(12) Controls	`u`	`NuIt`	`Ru`			
(13) Load Guess	`lg`	`Fname`				
(14) Display Param.	`np`	l				

1. Time step: Δt . This should come first in the `specs.txt` file.

2. MC Sweeps: `Ninit` is the number of initialization sweeps which are discarded. `Bsize` is the number of sweeps per block, and `Nblock` is the number of blocks generated and should be a multiple of 24 for the error calculation to work correctly. Statistics are recorded for each block, and then combined at the end of the process. Total number of sweeps is `NInit + Bsize * Nblock`.

3. Measurement: Each time this is called the file `z.`l`.dat` is loaded from the problem subdirectory, where $l \in \{0, 1, \ldots, K-1\}$. Only the points which are aligned with the time grid are used. In other words, only the data points where $n = (t - t_0)/\Delta t$ is (approximately) an integer and $0 \le n \le M$. It will stop reading when $n > M$, the file runs out, or `Ndata` points are read, which ever comes first. Missing data points are allowed, as well as extra data points in between time steps (they will just be ignored). Also a file called `z.`l`.in` is created which has only the data points which are actually used, for plotting purposes.

4. State variable: This may be called one time for each state variable with $l \in \{0, 1, \ldots, K-1\}$, or can be called once with $l = -1$ to set all the variables with

the same settings. ${\rm R^f}_l, {\rm y^c}_l, {\rm R^{y_0}}_l$ are defined in Eq. (A.3). Guess is used to set the initial constant value of the state variable. When setting ${\rm R^f}_l$ keep in mind that ${\rm R^f}_l \Delta t$ is actually the meaningful parameter.

5. Parameter: This may be called one time for each parameter with $l \in \{0, 1, \ldots, N_p - 1\}$, or by using $l = -1$ can be called once to set all the parameters with the same settings. ${\rm p^c}_l, {\rm R^p}_l$ are defined in Eq. (A.3), and can be used to provided some information about the parameters. Guess is used to set the initial value of the parameter.

6. State Bound: Sets a sharp bound on state variables $y_{l,min} < y_l(n) < y_{l,max}$. This may be called for individual state variables with $l \in \{0, 1, \ldots, K-1\}$, or can be called with $l = -1$ to set all the state variables with the same settings. The bounds must enclose the initial guess value.

7. Parameter Bound: Sets a sharp bound on parameters $p_{l,min} < p_l < p_{l,max}$. This may be called for individual parameters with $l \in \{0, 1, \ldots, N_p - 1\}$, or can be called with $l = -1$ to set all the parameters with the same settings. The bounds must enclose the initial guess value. It is sometimes useful to set $p_{l,min} = p_{l,max}$ to fix a parameter to a known value.

8. Runge-Kutta Prediction: Including this key activates RK4 prediction using a number of time steps `Nrk` which means the MC window size is decreased by `Nrk` number of steps. Setting `rkSub` allows the time steps in RK4 integration to be divided for better accuracy, for example `rkSub` $= 2$ means do 2 time steps of size $\Delta t/2$ in the RK4 integration.

9. Simulated Annealing: Including this key activates simulated annealing for the first `Ncool` number of sweeps. β in Eq. (A.3) is initially set to $\beta_0 < 1$ and is multiplied by the factor $\beta_0^{-1/{\rm Ncool}}$ after each sweep, until $\beta = 1$. This should only happen in the initialization phase, so `Ncool` should be less than `Ninit`.

10. External drive: Including this key loads a file with the name given by `Fname` located in the project subdirectory, starting at time t_0. This works in the same way as loading data files, by ignoring points not coincident with the time steps. The drive signal at the current time step is passed to the function `g(...)` as the last argument called `inj`. This will be a linear interpolation of the drive signal when

`g(...)` is called at subdivided time steps which happens in the RK4 prediction window only.

11. Constant coupling: This sets `coup` to the value specified during the initialization phase, and can be accessed in `model.h`. After the initialization phase is complete `coup` is set to zero, so this will not interfere with the path distribution. The intended usage is to add a coupling term to the model with `+ coup*RmMask[n*K+l] *(z[l] - y[l])`. The term `RmMask[n*K+l]` is 1 at (l, n) points that have measurements and 0 at (l, n) points that do not have measurements.

12. Controls: Set `NuIt` to the number of sweeps where controls terms are allowed to change. Set `NuIt` $<$ `Ninit` to avoid interfering with the path distribution. When this is set an additional term $\frac{\mathrm{Ru}}{2}\sum_{n,l} u_l(n)^2$ is added to the action. There is a control term associated with every point on the (l, n) grid, but the controls are fixed at zero for the points which do not have measurements associated with them. The intended usage is to add `+ fabs(u[l])*(z[l]-y[l])` onto the equations in the `g(...)` function where desired. When sweep number `NuIt` is reached the current value of the controls will be output to `u.dat` and the R-test to `R.dat`, and then the controls will be set to zero, and the MC process will continue.

13. Load Guess: This loads an initial guess for the path from the file with name `Fname`. The file should have `N` lines with a single value for `y[i]` on each line, in the usual order. This has to come after all the `y` and `p` commands to override the guesses.

14. Display Parameter: This will cause p_l to be displayed to the screen. Default is $l = 0$.

An example `specs.txt` is shown in Fig. A.2 for the Lorenz 96 example. Not all entries are present, just the ones that are being used. Refer to the table for the meanings. It is important that the first column be a lowercase letter, and that each entry separated by tabs or spaces, but it insensitive to the number of tabs or spaces and the order of the lines.

A.3.3 Measurement and External Drive files

The measurement data files are name `z.`l`.dat` where $l \in \{0, 1, \ldots, K-1\}$, and are located in the problem subdirectory. The format is the time values in the first column

```
# specs.txt Lorenz 96 example
# Set time step
dt  0.05
# Set all state variables to have Rf=100, guess = 1.0
y  -1        100.0       1.0
# Set for p[0]:  guess = 10.0,  pc = 8.0, Rp = 0.25
p  0        10.0        8.0       0.25
# Set bounds on p[0] between 0.0 and 30.0
pb  0        0.0         30.0
# Load z.0.dat and set Rm[0] = 8.0
# Load z.2.dat and set Rm[2] = 8.0
z  0      8.0
z  2      8.0
#   Ninit   Bsize   Nblock
mc  20000   1000    120
#   Ncool   Beta0
sa  10000   0.1
#   Nrk     rkSub
rk  10      2
```

Figure A.2: Example `specs.txt` file for Lorenz 96 example

and measurement values in the second column. Also external drive signals can be loaded from any file in the problem subdirectory with the same two-column format. See items 3 and 10 in the above list for more details.

A.4 Details of the Outputs

All the output files are written the problem subdirectory which is specified as a command line argument when *Carl* is run. In addition, there is output to the screen as the program is running.

Output to the screen: The first part of the output is to verify that all the settings are being used as expected. It may quickly scroll of the screen, but it should be checked when setting up a new problem, especially for warnings or errors. Then during the initialization phase the columns in order are: `NIt`, β, `p[l]`, number of accepted changes for `p[l]` over the previous block, and then the action for the current path. The parameter to be displayed may be selected with the `np` command. After the initialization phase is complete, the output is: 1st column is block number, 2nd column is `p[l]`, 3rd column is number of accepted changes for `p[l]` over the previous block, and 4th column is the action for the current path.

Mean and Standard Deviation in `mc.dat`: The means $\langle y_l(n)\rangle$ and the standard deviations $\sigma_l(n) = \sqrt{\langle (y_l(n) - \langle y_l(n)\rangle)^2\rangle}$ are written to the file `mc.dat`. The format is: the 1st column is time, 2nd column is $\langle y_0(n)\rangle$, 3rd column is $\sigma_0(n)$, 4th column is $\langle y_1(n)\rangle$, 5th column is $\sigma_1(n)$, and so on for all the state variables in order.

Skewness in `mc3.dat` and Kurtosis in `mc4.dat`: The skewness is written to the file `mc3.dat` and the kurtosis to the file `mc4.dat`. The 1st column in both files is time, and the nth column is the skewness or kurtosis associated with the state variable with the index $l = n - 2$.

Parameters in `param.dat`: The mean, standard deviation, skewness, and kurtosis for each parameter is written to a file called `param.dat`.

Blocked averages in `blocks.dat`: The blocked averages are written to the file `blocks.dat`. The first column is the block index, and there is one additional column for each entry in `#define PRINT_BLOCKS`.

Error Calculation in `error.dat`: The file `error.dat` will have four lines. The first column is the block size which will be `Bsize, 2*Bsize, 3*Bsize, 4*Bsize`. The second column is the number of blocks, which will be `Nblock, Nblock/2, Nblock/3,`

`Nblock/4`. Then there will be one additional column, with entries $\sqrt{var[\bar{x}]}$, for each quantity selected with the `#define PRINT_BLOCKS` statement. Each entry in this statement refers to a particular moment of a particular component of the path.

T-score Diagnostic in `t-hist.dat`: If the statement `#define TSCORE` is included in `model.h`, and if the true path is available in the file `x.dat`, then a t-score histogram is written to the file `t-hist.dat`. The first column is the t-score value at the center of the bin and the second column is the number of t-scores that fall into the associated bin.

Other outputs: A problem-specific `gnuplot` script file is created called `mc.plot`. This can be used to automatically generate plots of every state variable, showing the measurements that were used from `z.`l`.in`, the true path from `x.dat` (if available), and the MC results for mean and standard deviation of all the state variables at every time step.

The last path generated is output to a file called `y.dat`. The format is one value of `y[i]` per line, in order.

The file `model.h` is copied into the problem subdirectory and called `model.bac` for purposes of record keeping. That way all the problem-specfic files will be together the proper problem subdirectory for archiving.

Appendix B

Parallel Implementation of PIMC Data Assimilation

This appendix is about a parallel version of the path integral Monte Carlo data assimilation method which was implemented using CUDA, the Compute Unified Device Architecture. Doing this allows hundreds of threads of execution to be run in parallel on a graphics processing unit (GPU). The PIMC approach is well-suited to parallel execution, because the computational tasks can be divided up in a way such that the order of execution does not matter, and little coordination is required among threads.

The basic flow of the program is described here. First the measurement data, $\mathbf{Y} = \{\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_M\}$, is loaded from files and the current path is set to an initial guess path $\mathbf{X} = \{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_M\}$. Then $M+1$ threads are launched, one for each time point, $n = 0, 1, \ldots, M$. Each thread is assigned to a particular time n and it calculates the RHS of the model differential equations for the current path, $\mathbf{F}(\mathbf{x}_n)$, and stores the results in global device memory on the GPU. These numbers will later be used to calculate changes in action, and they will be updated every time part of the path changes. The idea behind this is to avoid unnecessary re-evaluations of $\mathbf{F}$, which may be expensive.

Then the main path update loop starts. Each time through the loop the following events happen. First $(M+1)/2$ threads (assume M is odd here for convenience) are launched, and each thread is assigned to an even n.

Each of these thread does the following tasks. First it adds a K-dimensional random perturbation $\Delta\mathbf{x}_n$ to the current state vector $\mathbf{x}_n$ by generating K random numbers, $\mathbf{x}'_n = \mathbf{x}_n + \Delta\mathbf{x}_n$. Then it calculates the RHS of the model differential equations

at time n using the perturbed state, $\mathbf{F}(\mathbf{x}'_n)$, and stores the result. The change in action which would occur if $\mathbf{x}_n$ was changed to $\mathbf{x}'_n$ is then computed. This is done using the pre-computed $\mathbf{F}(\mathbf{x}_n)$ and $\mathbf{F}(\mathbf{x}'_n)$ values, and also depends on $\mathbf{y}_n$, $\mathbf{x}_n$, $\mathbf{x}_{n-1}$, $\mathbf{x}_{n+1}$, and $\mathbf{x}'_n$. Then the change is either accepted or rejected using the usual Metropolis rule. Note that this is a slightly different approach than discussed in Chapter 3 and Appendix A, because there each component was accepted or rejected separately. This approach requires fewer function evaluations, but also may require a smaller random walk step size to get a good acceptance rate, so it is a trade off. If the change is accepted then $\mathbf{x}'_n \to \mathbf{x}_n$ and $\mathbf{F}(\mathbf{x}'_n) \to \mathbf{F}(\mathbf{x}_n)$.

Once all the threads operating on even n complete, then the same thing happens with all the odd n: another $(M+1)/2$ threads are launched and each one is assigned to an odd n. The reason for doing it in two steps is to uncouple the state vectors: to calculate the change in action due to perturbing $\mathbf{x}_n$, we need to know $\mathbf{x}_{n-1}$ and $\mathbf{x}_{n+1}$, but none of the other state vectors. This way each even n, and then each odd n can be updated independently, in any order.

Next each of the parameters p_l is updated one at a time in sequence, $l = 0, 1, \ldots, N_p - 1$. To each parameter a random perturbation is added. Changing a parameter value changes every model violation term, $\mathbf{g}_n$, and so these all need to be recalculated. The measurement terms in the action do not change though, since the states are not perturbed during this part of the process. Recall the model violation term is

$$\mathbf{g}_n = \mathbf{x}_{n+1} - \mathbf{x}_n - \frac{\Delta t}{2}(\mathbf{F}(\mathbf{x}_{n+1}, \mathbf{p}) + \mathbf{F}(\mathbf{x}_n, \mathbf{p})),$$

and the dynamics term of the action is

$$A_{0,f} = \frac{R_f}{2} \sum_{n=0}^{M-1} \mathbf{g}_n^2.$$

To calculate the new $\mathbf{g}_n^2$'s, $M+1$ threads are launched, and each one assigned to one n. Once again the pre-computed $\mathbf{F}(\mathbf{x}_n)$'s are used, and new $\mathbf{F}(\mathbf{x}_n)$'s are computed. Once this is completed, one thread per block sums up all the $\mathbf{g}_n^2$'s which are stored in shared memory. A block is a collection of threads which can interact by sharing memory and synchronizing with each other. The number of threads per block is limited by the hardware, and was 512 for this particular GPU. If the number of time steps used is larger than this number, the computations must be split over multiple non-interacting blocks which is what was done here.

Once the partial per block sums of ΔA_0 are completed, then a single thread is launched which sums the partial sums to find a total ΔA_0. This is then used in the Metropolis rule to make a decision about weather or not to accept the parameter change. If the change is accepted, then the path is updated, and the $\mathbf{F}(\mathbf{x}_n)$'s are updated. Actually there are always two copies of the $\mathbf{F}$'s, and a variable to select which is the current $\mathbf{F}$ and which is the $\mathbf{F}'$. This way excessive memory transfers are avoided (which are slow on the GPU). If the parameter change is accepted then the $\mathbf{F}$ selector just has to be flipped.

This whole process is repeated for each parameter, one at a time.

Then $N = (M+1)K + N_p$ threads are launched (usually split among multiple blocks, since typically $N > 512$, the maximum number of threads per block) to add to the running sums for the moments of each of the N components of the path. Each thread does the sums for one of the N components. Actually this step does not need to be done after each path update, only occasionally, since the path does not change very much after a single path update. Also it is not done during the initialization phase when the statistics are not recorded.

The random number generation is done in parallel using the "Hybrid Tausworthe" algorithm described in [43]. Each thread is given its own initial seed, and so each thread will generate a different pseudo-random sequence. The random seeds are read from the linux random device `/dev/urandom` when the program starts.

Some sample calculations were done as performance benchmarks. The code was run in parallel on an NVIDIA GeForce GT 320 GPU and also in series on an AMD Athlon II CPU for comparison. The same code was executed in both cases, but to run on the CPU using a single core instead of launching many threads, everything was run in sequence as a single thread.

The first test was the Lorenz 96 model with $K = 5$ state variables and $N_p = 1$ parameters, with $M = 640$ time steps. To do 32,000 path updates, it took 11 seconds to run in parallel on the GPU and 88 seconds to run in series on the CPU, so there was a factor of 8 speed increase. However the Lorenz 96 model is probably too computationally simple to be a good test: all that is involved in computing $\mathbf{F}$ is a few multiplications and additions.

The second test was with a much more complicated model. The Hodgkin-Huxley [25] spiking neuron model was used with $K = 4$ state variables and $N_p = 22$ parameters

was used. The number of time steps was $M = 999$. This model function is significantly more complicated, because it involves the evaluation of six exponential functions per time step. This is more likely to keep each GPU core busy doing calculations, instead of being idle. Also since M was set larger, we should except a greater increase in speed up when running in parallel. To do 20,000 path updates, it took 50 minutes to run in series on the CPU, and 51 seconds to run in parallel on the GPU. The speed increase was about a factor of 60.

Bibliography

[1] H. D. I. Abarbanel, D. R. Creveling, R. Farsian, and M. Kostuk. Dynamical state and parameter estimation. *SIAM Journal of Applied Dynamical Systems*, 8:1341–1381, 2009.

[2] Henry D. I. Abarbanel, Daniel R. Creveling, and James M. Jeanne. Estimation of parameters in nonlinear systems using balanced synchronization. *Phys. Rev. E*, 77(1):016208, 2008.

[3] Henry D.I. Abarbanel. Effective actions for statistical data assimilation. *Physics Letters A*, 373(44):4044 – 4048, 2009.

[4] Francis J. Alexander, Gregory L. Eyink, and Juan M. Restrepo. Accelerated Monte Carlo for optimal estimation of time series. *Journal of Statistical Physics*, 119(5), 2005.

[5] Jeffrey L. Anderson and Stephen L. Anderson. A Monte Carlo implementation of the nonlinear filtering problem to produce ensemble assimilations and forecasts. *Monthly Weather Review*, 127(12):2741–2758, 1999.

[6] A. Apte, M. Hairer, A.M. Stuart, and J. Voss. Sampling the posterior: An approach to non-Gaussian data assimilation. *Physica D: Nonlinear Phenomena*, 230(1-2):50 – 64, 2007. Data Assimilation.

[7] Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A tutorial on particle filters for on-line non-linear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing*, 50:174–188, 2001.

[8] R.P. Brent. *Algorithms for Minimization without Derivatives*. Prentice-Hall, 1973.

[9] S. Chandrasekhar. Stochastic problems in physics and astronomy. *Rev. Mod. Phys.*, 1943.

[10] Thomas Cover and Joy Thomas. *Elements of Information Theory*. John Wiley and Sons Inc., 1991.

[11] Henry Cox. On the estimation of state variables and parameters for noisy dynamic systems. *IEEE transactions on automatic control*, pages 5–12, 1964.

[12] Daniel R. Creveling, Philip E. Gill, and Henry D.I. Abarbanel. State and parameter estimation in nonlinear systems as an optimal tracking problem. *Physics Letters A*, 372(15):2640 – 2644, 2008.

[13] Arnaud Doucet, Simon Godsill, and Christophe Andrie. On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and Computing*, 10:197–208, 2000.

[14] Daniel Dubin. *Numerical and Analytical Methods for Scientists and Engineers Using Mathematica*. John Wiley and Sons, 2003.

[15] J.J Ebers and J.L. Moll. Large-signal behavior of junction transistors. *Proceedings of the IRE*, 42(12):1761–1772, Dec. 1954.

[16] Geir Evensen. *Data Assimilation: The Ensemble Kalman Filter*. Springer-Verlag Berlin Heidelberg, 2009.

[17] Geir Evensen. The ensemble Kalman filter for combined state and parameter estimation. *IEEE Control Systems Magazine*, 2009.

[18] Geir Evensen and Peter Jan Van Leeuwen. An ensemble Kalman smoother for nonlinear dynamics. *Monthly Weather Review*, 128:1852 –1867, 2000.

[19] R. M. Fano. *Transmission of Information: A Statistical Theory of Communication*. Wiley, New York, 1961.

[20] R.P. Feynman and A.R. Hibbs. *Quantum Mechanics and Path Integrals*. McGraw-Hill Book Company, 1965.

[21] R. Fletcher. *Practical Methods of Optimization*. John Wiley and Sons, 1987.

[22] Bernard Friedland and Irwin Bernstein. Estimation of the state of a non-linear process in the presence of nongaussian noise and disturbances. *Journal of the Franklin Institute*, 281(6), 1966.

[23] Philip E. Gill, Walter Murray, and Michael A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Rev.*, 47(1):99–131, 2005.

[24] Gene Golub and Charles Van Loan. *Matrix Computations*. Johns Hopkins University Press, 1996. Section 8.3.

[25] A.L. Hodgkin and A.F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *J. Physiology*, pages 500–544, 1952.

[26] Andrew H. Jazwinski. *Stochastic processes and filtering theory*. Academic Press, 1970.

[27] B. Jouvet and R. Phythian. Quantum aspects of classical and statistical fields. *Phys. Rev. A*, 19(3):1350–1355, Mar 1979.

[28] R.E. Kalman. A new approach to linear filter and prediction problems. *J. Basic Eng.*, 82:35–45, 1960.

[29] N.G. Van Kampen. *Stochastic Processes in Physics and Chemistry*. Elsevier, 1981.

[30] M.P. Kennedy. Chaos in the Colpitts oscillator. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, 41(11):771–774, Nov 1994.

[31] Rahul Konnur. Synchronization-based approach for estimating all model parameters of chaotic systems. *Phys. Rev. E*, 67(2):027204, Feb 2003.

[32] Peter Jan Van Leeuwen. Particle filtering in geophysical systems. *Monthly Weather Review*, 137:4089–4114, 2009.

[33] Don S. Lemons. *An Introduction to Stochastic Processes in Physics*. John Hopkins University Press, 2002.

[34] A.C. Lorenc. Analysis methods for numerical weather prediction. *Q.J.R. Met. Soc.*, 112:1177–1194, 1986.

[35] Andrew C. Lorenc and Tim Payne. 4D-var and the butterfly effect: Statistical four-dimensional data assimilation for a wide range of scales. *Q. J. R. Meteorol. Soc.*, 133(624):607–614, 2007.

[36] E. N. Lorenz. Predictability–a problem partly solved. In *European Centre for Medium Range Weather Forecasting*, volume 1, pages 1–18, 1996.

[37] Edward N. Lorenz and Kerry A. Emanuel. Optimal sites for supplementary weather observations: Simulation with a small model. *J. Atmos. Sci.*, 55(3):399–414, 1997.

[38] David J.C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

[39] G.M. Maggio, O. De Feo, and M.P. Kennedy. Nonlinear analysis of the Colpitts oscillator and applications to design. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, 46(9):1118–1130, Sep 1999.

[40] Anil Maybhate and R. E. Amritkar. Use of synchronization and adaptive control in parameter estimation from a time series. *Phys. Rev. E*, 59(1):284–293, Jan 1999.

[41] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. Equations of state calculations by fast computing machines. *J. Chem. Phys*, 21(6), 1953.

[42] Radford M. Neal. Probabilistic inference using Markov chain Monte Carlo methods. *Technical Report CRG-TR-93-1*, 1993.

[43] Hubert Nguyen. *GPU Gems 3*. NVIDIA Corporation, 2008.

[44] Athanasios Papoulis. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill Book Company, 1984.

[45] U. Parlitz. Estimating model parameters from time series by autosynchronization. *Phys. Rev. Lett.*, 76(8):1232–1235, Feb 1996.

[46] Louis M. Pecora and Thomas L. Carroll. Synchronization in chaotic systems. *Phys. Rev. Lett.*, 64(8):821–824, Feb 1990.

[47] Louis M. Pecora, Thomas L. Carroll, Gregg A. Johnson, Douglas J. Mar, and James F. Heagy. Fundamentals of synchronization in chaotic systems, concepts, and applications. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 7(4):520–543, 1997.

[48] Dinh Tuan Pham. Stochastic methods for sequential data assimilation in strongly nonlinear systems. *Mon. Weath. Rev.*, 129:1194–1207, 2001.

[49] A. Pikovsky, M. Rosenblum, and J. Kurths. *Synchronization: A universal concept in nonlinear sciences*. Cambridge University Press, 2001.

[50] Carlos A. Pires, Olivier Talagrand, and Marc Bocquet. Diagnosis and impacts of non-gaussianity of innovations in data assimilation. *Physica D: Nonlinear Phenomena*, 239(17):1701 – 1717, 2010.

[51] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, 2007.

[52] R. Pythian. The functional formalism of classical statistical dynamics. *J. Phys. A: Math. Gen.*, 10:777–789, 1977.

[53] John C. Quinn and Henry D.I. Abarbanel. State and parameter estimation using Monte Carlo evaluation of path integrals. *Quarterly Journal of the Royal Meterological Society*, 2010. In Press.

[54] John C. Quinn, Paul H. Bryant, Daniel R. Creveling, Sallee R. Klein, and Henry D. I. Abarbanel. Parameter and state estimation of experimental chaotic systems using synchronization. *Phys. Rev. E*, 80(1):016201, Jul 2009.

[55] Juan M. Restrepo. A path integral method for data assimilation. *Physica D: Nonlinear Phenomena*, 237(1):14 – 27, 2008.

[56] Mutsuo Saito and Makoto Matsumoto. SIMD-oriented fast Mersenne twister: a 128-bit pseudorandom number generator. In *Monte Carlo and Quasi-Monte Carlo Methods*, pages 607–622. Springer Berlin Heidelberg, 2008.

[57] Paul So, Edward Ott, and W. P. Dayawansa. Observing chaos: Deducing and tracking the state of a chaotic system from limited observation. *Phys. Rev. E*, 49(4):2650–2660, Apr 1994.

[58] P.J. van Leeuwen and G. Evensen. Data assimilation and inverse methods in terms of a probabilistic formulation. *Monthly Weather Review*, 124:2892–2913, 1996.

[59] H.U. Voss, J. Timmer, and J. Kurths. Nonlinear dynamical system identification from uncertain and indirect measurements. *Int. J. Bifurcation Chaos Appl. Sci. Eng.*, 14(6), 2004.