1 Introduction

This document is a userguide for a biased random walk Markov Chain Monte Carlo method, called Strategic Monte Carlo. This method was developed during my thesis work at UCSD and the details of the method may be found in [2]. The purpose of this code is to determine the expected model parameter and state values and the probability density structure given some mathematical model. This mathematical model is specifically in the form of a set of differential equations

$$\frac{d}{dt}\mathbf{x} = \mathbf{F}(\mathbf{x}, \mathbf{p}) \tag{1.1}$$

Where $\mathbf{x}$ is the vector of state variables of the system, $\mathbf{p}$ is the vector of model parameters, and $\mathbf{F}$ is the mathematical model for the dynamics.

The method relies on searching a probability space for $P(\mathbf{x}, \mathbf{p}\,|\mathbf{y}\,)$ where $\mathbf{y}$ is the experimentally collected data. The derivation of this probability, through the definition of a negative log-likelihood or action, can be found in [1], [3], and [2].

This monte carlo code requires as input the location of probability maxima, the model equations, and measured data. The probability space is searched biased around the local probability maxima determined previously. This allows for searches in high dimensional space without wasting time on regions already known to be of low probability. Due to the required locational input of the probability maxima (or negative log-likelihood/action minima) this method must be done following an optimization routine. The code has been written to interface with and accept the output of the Variational Annealing optimizer, minAone, which you can download from here: https://github.com/yejingxin/minAone.

2 How to

2.1 Work flow

As mentioned in the introduction, Strategic Monte Carlo depend on having previously performed an optimization to find the location of probability maxima to explore the structure of the probability around those points. We define the location of these probability maxima as $(\mathbf{x}, \mathbf{p})_0$. The precise order of executions of scripts is the following

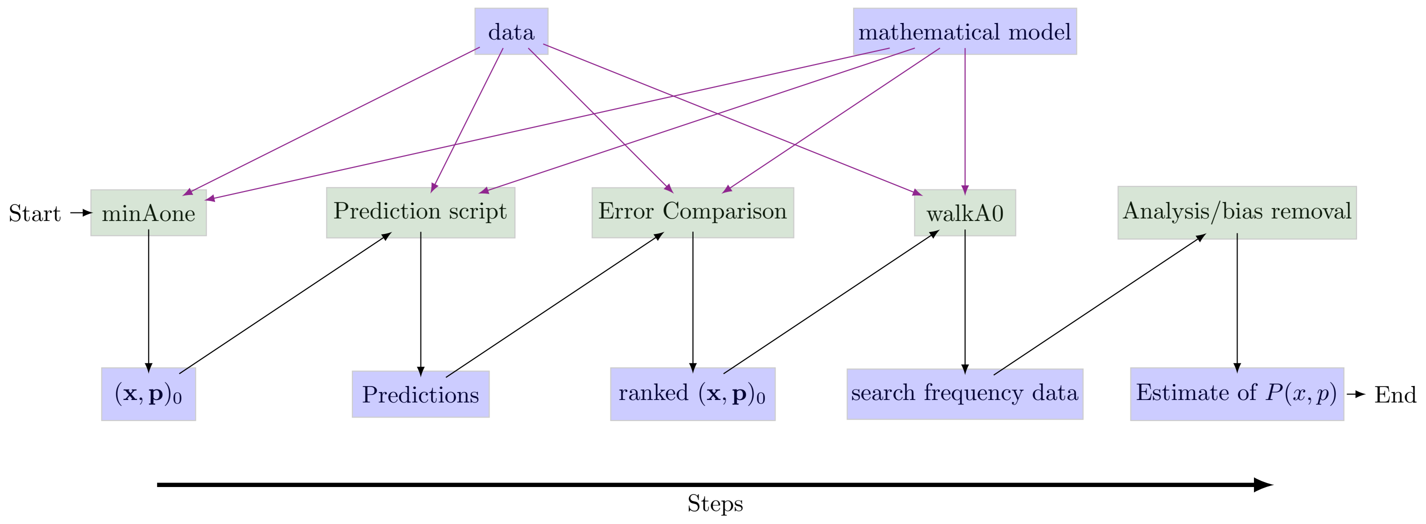

Figure 1: Workflow for this method. Highlighted in light blue are inputs and outputs. These files are static. Highlighted in light green are different scripts. These are code that must be run with the appropriate input. The plum lines represent inputs that feed into several scripts and are determined independently from this process (such as data and a physical model)

2.2 Necessary Files

All time series data should be saved

2.2.1 minAone

More detail on this particular code can be found under the information for minAone. But a brief overview is included here as well.

MinAone requires two text files in addition to the code and data to run. These two files are an "equations.txt" file and a "specs.txt" file. In addition, since the Strategic Monte Carlo requires the location of as many local probability maxima as possible to run efficiently, a "sub.sh" file is useful to be able to submit an ensemble of optimization routines starting at several different random initial guesses. An example of this "sub.sh" script can be found at the link above. Otherwise a bash script may be written to run all X ensemble entrees at once. Each ensemble entree is defined as a "path" with some numerical label. This label will be conserved throughout all scripts for this particular ensemble entree.

2.2.2 Prediction - predict_anneal_steps.py

This script was modified from one originally written by Daniel Breen. The prediction script is a python code that takes all of the outputs of minAone (at each annealing value - see minAone Userguide for details) and runs a prediction forward in time using the model. To run this code, you need the same "equations.txt" and "specs.txt" as for minAone. Additionally, all the output files from minAone are required. A similar submission script "predict.sh" may be written for this code.

Without a submission file, to run this script, run it in python with 2 arguments. The first argument is the number of time points forward you want to predict. And the second is which "path" you are predicting on.

2.2.3 Prediction Comparison - PredictionComparison.ipynb

This script was written to analyze the predictions from the previous prediction script and give the path number and annealing step for the paths that give the best and worst prediction error. These two locations in parameter space determine the center and width of the bias for the monte carlo search. This script requires the data (also required for minAone). The data must be named "data$x$.txt". Where x is the index in $\mathbf{x}$ corresponds to the state being measured. Since both this and walkA0.py are python based, the first entree of $\mathbf{x}$ has index 0.

2.2.4 Strategic Monte Carlo - walkA0.py

This is the actual Monte Carlo code. Most of the computation is spent on this. This code requires all the output from minAone as well as data files and 2 additional files. The data files must be named "data$x$.txt" where x is the index of the measured variable. These files must be column data of just the state variable at each time point. The time points are assumed to be equally spaced. Time is assumed to start at 0 and go by even increments which are externally defined in one of the supplemental text files for this section.

The first additional file is a one of the numerical information necessary to run the correct MC in the space you are interested in. This file is called "NumInfo.txt". The second is a list of differential equations, "ParallelEqns.py". Though this second file has the same function as "equations.txt" it has a different form.

The format for "NumInfo.txt" is the following:

```
NumberStates, nS, This is the number of states in the system
NumberParameters, nP, this is the number of parameters in the system
NumberMeasurements, nM, this is the number of measurements recorded
TimeSteps, nT, This is the total number of time steps that you have used
deltaTime, dT, This is the time difference between each of the time steps
NumberWalkers, nW, This is the number of walkers you want to have running
NumberBurnInSteps, nB, This is the number of steps you want for "burn in"
alpha, A, This is the value of alpha you used during minAone
beta, Bgood, This is the value of beta where the best predictions occur
beta bad, Bbad, This is the value of beta where the worst predictions occur
RfVariable1, Rf1, This is the scaling Rf factor from minAone
RfVariable2, Rr2,
RfVariable3, Rf3,
Measurement1, m1, Index of measured variable
Measurement1, m2,
BestPathIndex, pathgood, best path index
WorstPathIndex, pathbad, worst path index
AnalysisAttemptNumber, nA,
```

walkA0.py reads this file in a comma separated way and only reads the second item on each line. This means that the numerical value of each term listed here must be the *second* item on each line. In the example above we only put 3 Rf terms and 2 measurement terms. In a real example, the number of Rf lines should match the total number of variables and the number of measurement lines should match nM. The Rf terms should be taken directly from the minAone "specs.txt" file. These terms effectively give the allowed relative model error of each variable in comparison to the other variables. Since Rf is model precision (1/model variance), if there is no particular variable that should be constrained more strongly to the model than others, it is typical to set the Rf value such that it is inversely proportional to the variable's dynamical range. In other words, a variable that may span a larger region should have a larger Rf value than one which spans a smaller region. This allows for approximately the same percentage model precision for each variable.

The format for "ParallelEqns.py" is the following:

```
import sys
import numpy as np
import scipy as sp
import sympy as sym
import matplotlib.pyplot as plt
import os

def ParDeriv(x,p):
    d1 = (x[1] - x[9])*x[10] - x[0] + p
    d2 = (x[2] - x[10])*x[0] - x[1] + p
    d3 = (x[3] - x[0])*x[1] - x[2] + p
    d4 = (x[4] - x[1])*x[2] - x[3] + p
    d5 = (x[5] - x[2])*x[3] - x[4] + p
    d6 = (x[6] - x[3])*x[4] - x[5] + p
    d7 = (x[7] - x[4])*x[5] - x[6] + p
    d8 = (x[8] - x[5])*x[6] - x[7] + p
    d9 = (x[9] - x[6])*x[7] - x[8] + p
    d10 = (x[10] - x[7])*x[8] - x[9] + p
    d11 = (x[0] - x[8])*x[9] - x[10] + p
return np.array([d1,d2,d3,d4,d5,d6,d7,d8,d9,d10,d11])
```

This example is specifically for an 11 dimensional Lorenz 96 model. For other models in which there are more parameters, the "p" term should be treated as a vector similarly to the way the "x" term is treated here. It is important that the function in this file has the name "ParDeriv" and takes the state variable as the first argument and the parameters you wish to estimate as the second argument.

If you wish to include a time dependent forcing term (like a driving current), the function should be changed by adding a third argument, the forcing term. The file with this forcing term as a function of time must be named "Input.txt". Currently walkA0.py is only set up for at most 1 time varying driving force.

2.2.5 Bias Removal - MeasurementAnalysis.ipynb

This code takes as an input the raw output of walkA0.py and some user defined inputs that are prompted as the code is run. This file generates a raw histogram of the parameter search space (currently set up for only a one dimensional parameter vector) and an estimate of the probability surface with the weight due to the bias removed. Additionally this script plots the mean and standard deviation for parameter states with the bias from the search removed.

2.3 File Structure

Here I have included the file structure for properly running all the code. Blue boxed items are directories, green boxed items are code that is run, and others are files.

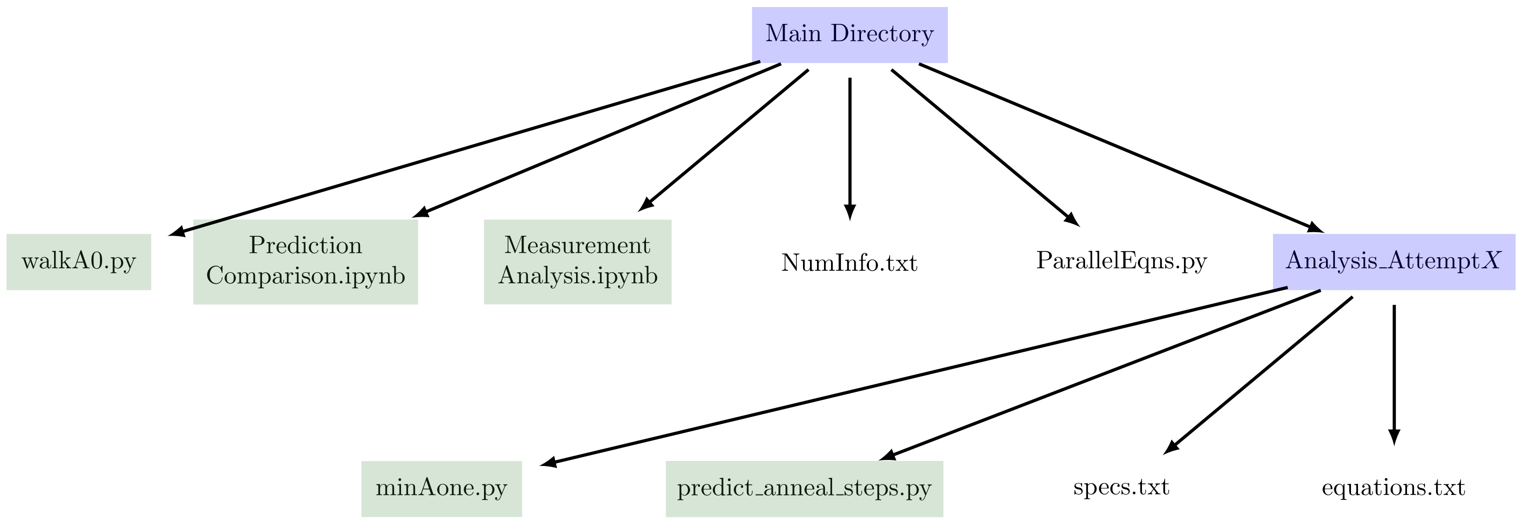

Figure 2: File structure to properly use walkA0.py and all preceding scripts

References

[1] H. Abarbanel, *Predicting the future: completing models of observed complex systems*, Springer, 2013.

[2] A. Shirman, *Strategic Monte Carlo and Variational Methods in Statistical Data Assimilation for Nonlinear Dynamical Systems*, PhD thesis, UC San Diego, 2018.

[3] J. Ye, N. Kadakia, P. Rozdeba, H. Abarbanel, and J. Quinn, *Improved variational methods in statistical data assimilation*, Nonlinear Processes in Geophysics, 22 (2015), pp. 205–213.