Data Assimilation with Markov Chain Monte Carlo

Stephen Ingraham

University of Minnesota

Abstract

The methods of data assimilation are currently being applied in various fields of research. Data assimilation allows us to combine a mathematical model with empirical data in order to improve the accuracy of the model. In this paper, one of the techniques of data assimilation (smoothing) is applied using Markov Chain Monte Carlo (MCMC) algorithms. Four discrete dynamical systems along with artificially generated ‘data’ are used to illustrate the principles of using MCMC in smoothing problems. Both deterministic and stochastic systems will be analyzed.

Data Assimilation with Markov Chain Monte Carlo

During the last several decades, advances in computational power have enabled researchers in a variety of fields to simulate complex processes with sophisticated mathematical models. With modern technology, we also have the power to gather, store, and analyze very large data sets. New areas of research, such as bioinformatics, have developed rapidly in the wake of modern data processing capabilities.

The techniques of data assimilation combine the power of mathematical modeling and data analysis. By incorporating empirical data into the model, researchers can generate more accurate results. For example, it is possible to better approximate measurements which are missing from a data set. Alternatively, one can make more accurate predictions about future data points. These techniques have been particularly useful in fields such as ecology and climate science (Zobitz et. al., 2011).

In this paper, an implementation of the Markov Chain Monte Carlo (MCMC) method will be applied to several abstract data assimilation problems. In each case, the goal will be to compute the most likely initial condition of a discrete dynamical system based on some 'data' that we have on the orbit of the system. This data will be randomly generated, rather than taken from a real-world system. Then we can compare the MCMC-approximated initial condition with the actual initial condition that was chosen.

Several dynamical systems will be analyzed. The first is the deterministic logistic map. The orbits of deterministic dynamical systems are completely determined by the initial condition. The second system is the stochastic sine map. In stochastic dynamical systems, each iteration of the map will have some noise added, so that the orbit of the system has an element of randomness. The third system is the deterministic Gauss map. The last system to be analyzed is

a two-dimensional deterministic map known as the Tinkerbell map. For each of these systems, MATLAB programs which implement MCMC algorithms will be used to generate plots which display the probability distribution of the initial condition (deterministic case) or the probability distribution of the entire orbit vector (stochastic case).

Discrete-Time Data Assimilation

Let $\psi \in C(\mathbb{R}^n, \mathbb{R}^n)$ and let v be a Markov chain $v = \{v_j\}_{j\in\mathbb{Z}^+}$, where v is defined by the deterministic map: $v_{j+1} = \psi(v_j), j \in \mathbb{Z}^+$, with the initial condition $v_0 \sim N(m_0, C_0)$. We call the Markov chain v the signal. In addition, we have data $y = \{y_j\}_{j\in\mathbb{N}}$ which is defined by $y_{j+1} = h(v_{j+1}) + \eta_{j+1}, j \in \mathbb{Z}^+$. The observation operator $h \in C(\mathbb{R}^n, \mathbb{R}^m)$ is a function of the signal, and the noise term $\eta = \{\eta_j\}_{j\in\mathbb{N}}$ is an i.i.d. sequence.

The objective of data assimilation is to condition the random variable v on the observed data y. For instance, in the deterministic case, we evaluate the probability distribution of the initial condition v_0, given the data. This initial condition determines the entire orbit of the dynamical system.

In the stochastic case, the situation is slightly different. We define the stochastic map as $v_{j+1} = \psi(v_j) + \xi_j, j \in \mathbb{Z}^+$, with the initial condition $v_0 \sim N(m_0, C_0)$. This time, we have an added noise element $\xi = \{\xi_j\}_{j\in\mathbb{Z}^+}$, which is an i.i.d. sequence with $\xi_0 \sim N(0, \Sigma), \Sigma > 0$. In this case, we must evaluate the probability distribution of the entire sequence of v (not just the initial condition), given the data y.

In both the stochastic case and the deterministic case, the conditioned random variable $v|y$ is called the smoothing distribution. The derivation of the prior and posterior distributions on $v|y$ and the associated log likelihood function is presented in Sec. 2.3 of Law et. al. (2015).

In real-world applications, the vectors v and y are often quite long (e.g. temperature measurements taken from many different points on the Earth). However, this paper will present only one- and two-dimensional maps. These examples will illustrate the basic principles of MCMC data assimilation, which can be generalized to higher-dimensional systems.

1. Deterministic Logistic Map

The first system to be analyzed is the familiar one-dimensional logistic map, where $\psi(v) = rv(1 - v)$. Because this is a deterministic map (without noise), the entire orbit of the system will be determined by the initial condition v_0. Therefore, it will suffice to find the posterior distribution of the initial condition, which will be in $\mathbb{R}$ for this one-dimensional system.

The Random Walk Metropolis MCMC algorithm is implemented in a MATLAB program (Program 1) borrowed from p3.m on page 118 of Law et. al. (2015). The script consists of four parts: setup, truth, solution, and plots to display the results.

The setup section specifies the number of iterations of the map (J=5), the parameter value (r=4), the square root of the observational noise variance (gamma), the mean and variance of the initial condition (m_0 and C_0, respectively). It also seeds the random number generator.

The truth section sets the actual initial condition, along with the sequences $\{v_j\}$ (the signal) and $\{y_j\}$ (the data). This section also initializes the log likelihood function, which will be used to compute the optimum initial condition value in the solution section.

The solution section sets up a random walker which gives a proposal sample w for the initial condition of the Markov Chain v. Depending on the log likelihood value associated with the Markov Chain generated by w, it will either be accepted or rejected. The algorithm then iterates this process over a large number of samples (the Monte Carlo component of MCMC).

Finally, we can analyze the histogram which shows the distribution of optimal initial condition values over all Monte Carlo trials. The results of running Program 1 are shown in Figure 1.

The left-hand peak clearly corresponds to the true value of v_0=0.3. However, there is a symmetrical peak near the value 0.7, due to the dynamics of the logistic map when r=4. This logistic map takes the points 0.3 and 0.7 to the same point under one iteration. Because the data only begins after the first iteration, the algorithm cannot determine whether 0.3 or 0.7 was the actual initial condition. It is important to note that the MCMC method will not necessarily identify a global optimum (as in this case). Also, the algorithm can be quite sensitive to various values set in the program, including r, J, gamma, or even the seed for the random number generator used in the random walker. Some more sophisticated algorithms can help to account for these issues, but it is still generally necessary to tune the various parameters, and to make sure that the results make sense.

2. Stochastic Sine Map

The sine map is defined as $\psi(v) = \alpha \sin(v)$. In the stochastic case, the Markov Chain will be $v_{j+1} = \psi(v_j) + \xi_j, j \in \mathbb{Z}^+$. The noise element ξ_j will be an i.i.d. sequence with $\xi_0 \sim N(0, \sigma^2)$.

The Independence Dynamics Sampler MCMC algorithm is implemented in Program 2, borrowed from p4.m on page 120 of Law et. al. (2015). Each section of the program is similar to the corresponding section in Program 1, with some modifications. In the setup, there is an added term sigma, which is the variance of the signal noise. The parameter in this map has the value

$\alpha = 2.5$. In the truth section, the initial condition is generated randomly, along with the corresponding truth signal and data (both signal and data have noise this time).

Rather than generating a single value for the proposed initial condition (as in the deterministic case) the solution section of Program 2 generates an entire proposed sequence $\{w_j\}$ using the stochastic map. The log likelihood of each sample w is computed and is either accepted or rejected.

The first plot (Figure 2) generated by Program 2 illustrates how the acceptance ratio of proposed samples decreases as we run more Monte Carlo trials. As this ratio stabilizes (after about 50,000 trials), we see that the cumulative mean of the first elements of proposed samples w also begins to approach a value near 0.3. However, convergence has apparently not occurred after 10^5 samples. N would need to be adjusted upward to confirm convergence, but this would be done at the expense of computation time.

3. Deterministic Gauss Map

The Gauss map is another 1-D deterministic map defined by $\psi(v) = \exp(-av^2) + b$. The MCMC RWM algorithm from Program 1 was modified to use this new dynamical system in Program 3. The same methods applied for the logistic map were used in this program. The parameters were set with values $a = 4, b = 0$. The histogram in Figure 4 shows a local optimum value around 0.3 (the true value is 0.3), but there is more noise than there was in the logistic map histogram. As was mentioned earlier, it appears that the results of MCMC can be quite sensitive to the values chosen for gamma (data noise variance), J (number of steps), beta (random walker step size), or other variables. And the values selected for the parameters a and b will obviously change the dynamics of ψ, which will affect the final result.

4. Deterministic Tinkerbell Map

The final example under consideration is a two-dimensional deterministic map called the Tinkerbell Map. It is given by $x_{n+1} = x_n^2 - y_n^2 + ax_n + by_n$ and $y_{n+1} = 2x_n y_n + cx_n + dy_n$. The MCMC RWM algorithm from Program 1 was modified to work with this 2-D map in Program 4. The parameters were set with values a=0.9, b=-0.6013, c=2.0, d=0.5.

Program 4 works in a similar way to Programs 1 and 3, except that we now have two Markov chains (one for x and one for y). Note that in this case, y refers to the second component of the signal, not to the data (as in the theory section above). The data is only generated for the second chain (y), so the observation operator $h(x, y) = h(y) = y$. The histogram in Figure 5 shows the initial condition samples output by Program 4. The initial conditions (x_0, y_0) are partitioned into bins in the (x,y) plane, and the frequency of initial conditions in each bin is plotted in the z-direction. In this example, the truth initial condition was set as $x_0 = -0.72$ and $y_0 = -0.64$. The histogram in Figure 5 does appear to have a peak in the vicinity of the true initial condition, but it may be necessary to perform more trials or tweak the noise variance to get a better approximation.

Conclusion

In this paper, MCMC algorithms were applied to the smoothing problem for four discrete dynamical systems. Data assimilation methods encompass far more than these limited examples. Continuous-time systems, the filtering problem, various alternative algorithms, higher-dimensional systems, and a variety of other aspects of data assimilation were not addressed in this brief introduction. However, the examples here show how we might apply these techniques in a relatively straightforward way to various models and real-world data.

References

Zobitz, J.M. et. al. (2011). A Primer for Data Assimilation with Ecological Models Using Markov Chain Monte Carlo (MCMC). *Oecologia 167*, Pages 599-611.

Law, K., Stuart, A., & Zygalakis, K. (2015). Data assimilation: A Mathematical Introduction (Vol. 62). Cham, Switzerland: Springer. ISBN 978-3-319-20325-6.

MATLAB Programs (adapted Law et. al., 2015)

Program 1: MCMC RWM Algorithm for Logistic Map (from p3.m: p. 118, Law)

```
%% setup
 J=5;% number of steps
 r=4;% dynamics determined by alpha
 gamma=0.2;% observational noise variance is gamma^2
 C0=0.01;% prior initial condition variance
 m0=0.5;% prior initial condition mean
 sd=10;rng(sd);% choose random number seed

 %% truth
 vt=0.3;vv(1)=vt;% truth initial condition
 Jdet=1/2/C0*(vt-m0)^2;% background penalization
 Phidet=0;% initialization model-data misfit functional
 for j=1:J
 % can be replaced by Psi for each problem
 vv(j+1)=r*vv(j)*(1-vv(j));% create truth
 y(j)=vv(j+1)+gamma*randn;% create data
 Phidet=Phidet+1/2/gamma^2*(y(j)-vv(j+1))^2;% misfit functional
 end
 Idet=Jdet+Phidet;% compute log posterior of the truth

 %% solution
 % Markov Chain Monte Carlo: N forward steps of the
 % Markov Chain on R (with truth initial condition)
 N=1e5;% number of samples
 V=zeros(N,1);% preallocate space to save time
 beta=0.05;% step-size of random walker
 v=vt;% truth initial condition (or else update I0)
 n=1; bb=0; rat(1)=0;
 while n<=N
 w=v+sqrt(2*beta)*randn;% propose sample from random walker
 vv(1)=w;
 Jdetprop=1/2/C0*(w-m0)^2;% background penalization
 Phidetprop=0;
 for i=1:J
 vv(i+1)=r*vv(i)*(1-vv(i));
 Phidetprop=Phidetprop+1/2/gamma^2*(y(i)-vv(i+1))^2;
 end
 Idetprop=Jdetprop+Phidetprop;% compute log posterior of the proposal

 if rand<exp(Idet-Idetprop)% accept or reject proposed sample
 v=w; Idet=Idetprop; bb=bb+1;% update the Markov chain
 end
 rat(n)=bb/n;% running rate of acceptance
 V(n)=v;% store the chain
 n=n+1;
 end
 dx=0.0005; v0=[0.01:dx:0.99];
 Z=hist(V,v0);% construct the posterior histogram
 figure(1), plot(v0,Z/trapz(v0,Z),'k','Linewidth',2)% visualize posterior
```

MATLAB Programs (adapted from Law et. al., 2015)

Program 2: MCMC Independence Dynamics Sampler for Sine Map (from p4.m: p. 120, Law)

```
%% setup
J=10;% number of steps
alpha=2.5;% dynamics determined by alpha
gamma=1;% observational noise variance is gamma^2
sigma=1;% dynamics noise variance is sigma^2
C0=1;% prior initial condition variance
m0=0;% prior initial condition mean
sd=0;rng(sd);% choose random number seed

%% truth
vt(1)=m0+sqrt(C0)*randn;% truth initial condition
Phi=0;
for j=1:J
vt(j+1)=alpha*sin(vt(j))+sigma*randn;% create truth
y(j)=vt(j+1)+gamma*randn;% create data
% calculate log likelihood of truth, Phi(v;y) from (1.11)
Phi=Phi+1/2/gamma^2*(y(j)-vt(j+1))^2;
end

%% solution
% Markov Chain Monte Carlo: N forward steps of the
% Markov Chain on R^{J+1} with truth initial condition
N=1e5;% number of samples
V=zeros(N,J+1);% preallocate space to save time
v=vt;% truth initial condition (or else update Phi)
n=1; bb=0; rat(1)=0;
while n<=N
w(1)=sqrt(C0)*randn;% propose sample from the prior
Phiprop=0;
for j=1:J
w(j+1)=alpha*sin(w(j))+sigma*randn;% propose sample from the prior
Phiprop=Phiprop+1/2/gamma^2*(y(j)-w(j+1))^2;% compute likelihood
end
if rand<exp(Phi-Phiprop)% accept or reject proposed sample
v=w; Phi=Phiprop; bb=bb+1;% update the Markov chain
end
rat(n)=bb/n;% running rate of acceptance
V(n,:)=v;% store the chain
n=n+1;
end
% plot acceptance ratio and cumulative sample mean
figure;plot(rat)
figure;plot(cumsum(V(1:N,1))./[1:N]')
xlabel('samples N')
ylabel('(1/N) \Sigma_{n=1}^N v_0^{(n)}')
```

MATLAB Programs (adapted from Law et. al., 2015)

Program 3: MCMC RWM Algorithm for Gauss Map (modified Program 1)

```
%% setup
 J=5;% number of steps
 a=4;% dynamics determined by parameter a
 b=0;% and by parameter b
 gamma=0.2;% observational noise variance is gamma^2
 C0=0.01;% prior initial condition variance
 m0=0.5;% prior initial condition mean
 sd=10;rng(sd);% choose random number seed

 %% truth
 vt=.3;vv(1)=vt;% truth initial condition
 Jdet=1/2/C0*(vt-m0)^2;% background penalization
 Phidet=0;% initialization model-data misfit functional
 for j=1:J
 % can be replaced by Psi for each problem
 vv(j+1)=exp(-a*vv(j)^2)+b;% create truth
 y(j)=vv(j+1)+gamma*randn;% create data
 Phidet=Phidet+1/2/gamma^2*(y(j)-vv(j+1))^2;% misfit functional
 end
 Idet=Jdet+Phidet;% compute log posterior of the truth

 %% solution
 % Markov Chain Monte Carlo: N forward steps of the
 % Markov Chain on R (with truth initial condition)
 N=1e5;% number of samples
 V=zeros(N,1);% preallocate space to save time
 beta=0.05;% step-size of random walker
 v=vt;% truth initial condition (or else update I0)
 n=1; bb=0; rat(1)=0;
 while n<=N
 w=v+sqrt(2*beta)*randn;% propose sample from random walker
 vv(1)=w;
 Jdetprop=1/2/C0*(w-m0)^2;% background penalization
 Phidetprop=0;
 for i=1:J
 vv(i+1)=exp(-a*vv(i)^2)+b;
 Phidetprop=Phidetprop+1/2/gamma^2*(y(i)-vv(i+1))^2;
 end
 Idetprop=Jdetprop+Phidetprop;% compute log posterior of the proposal

 if rand<exp(Idet-Idetprop)% accept or reject proposed sample
 v=w; Idet=Idetprop; bb=bb+1;% update the Markov chain
 end
 rat(n)=bb/n;% running rate of acceptance
 V(n)=v;% store the chain
 n=n+1;
 end
 dx=0.0005; v0=[0.01:dx:0.99];
 Z=hist(V,v0);% construct the posterior histogram
 figure(1), plot(v0,Z/trapz(v0,Z),'k','Linewidth',2)% visualize posterior
```

MATLAB Programs (adapted from Law et. al., 2015)

Program 4: MCMC RWM Algorithm for 2-D Tinkerbell Map (modified Program 1)

```
%% setup
 J=5;% number of steps
 a=0.9;% parameter a
 b=-0.6013;% parameter b
 c=2.0;% parameter c
 d=0.50;% parameter d
 gamma=0.1;% observational noise variance is gamma^2
 C0=0.01;% prior initial condition variance
 m0=-0.7;% prior initial condition mean
 sd=10;rng(sd);% choose random number seed

 %% truth
 pp = [];
 qq = [];
 pt=-0.72;pp(1)=pt;% truth initial condition: p
 qt=-0.64;qq(1)=qt;% truth initial condition: q
 Jdet=1/2/C0*(pt-m0)^2+1/2/C0*(qt-m0)^2;% background penalization
 Phidet=0;% initialization model-data misfit functional
 for j=1:J
 % can be replaced by Psi for each problem
 pp(j+1)=(pp(j)^2-qq(j)^2)+a*pp(j)+b*qq(j);% create truth for p
 qq(j+1)=2*pp(j)*qq(j)+c*pp(j)+d*qq(j);% create truth for q
 y(j)=pp(j+1)+qq(j+1)+gamma*randn;% create data (only for q component)
 Phidet=Phidet+1/2/gamma^2*(y(j)-pp(j+1)-qq(j+1))^2;% misfit functional
 end
 Idet=Jdet+Phidet;% compute log posterior of the truth

 %% solution
 % Markov Chain Monte Carlo: N forward steps of the
 % Markov Chain on R (with truth initial condition)
 N=1e4;% number of samples
 V=zeros(N,1);% preallocate space to save time
 beta=0.05;% step-size of random walker
 p=pt+gamma*randn;% truth initial condition for p
 q=qt+gamma*randn;% truth initial condition for q
 n=1; bb=0; rat(1)=0;
 while n<=N
 w=p+sqrt(2*beta)*randn;% propose sample from random walker for p
 x=q+sqrt(2*beta)*randn;% random walker sample for q
 pp(1)=w;
 qq(1)=x;
 Jdetprop=1/2/C0*(w-m0)^2+1/2/C0*(x-m0)^2;% background penalization
 Phidetprop=0;
 for j=1:J
 pp(j+1)=(pp(j)^2-qq(j)^2)+a*pp(j)+b*qq(j);
 qq(j+1)=2*pp(j)*qq(j)+c*pp(j)+d*qq(j);
 Phidetprop=Phidetprop+1/2/gamma^2*(y(j)-pp(j+1)-qq(j+1))^2;
 end
 Idetprop=Jdetprop+Phidetprop;% compute log posterior of the proposal
 if rand<exp(Idet-Idetprop)% accept or reject proposed sample
 p=w; q=x; Idet=Idetprop; bb=bb+1;% update the Markov chain
```

```
 end
 rat(n)=bb/n;% running rate of acceptance
 P(n)=p;% store the chain
 Q(n)=q;
 Z=[P;Q];
 n=n+1
 end
 Z
 % dx=0.0005; v0=[0.01:dx:0.99];
hist3(Z',[25 25]);% construct the posterior histogram
% visualize posterior
```

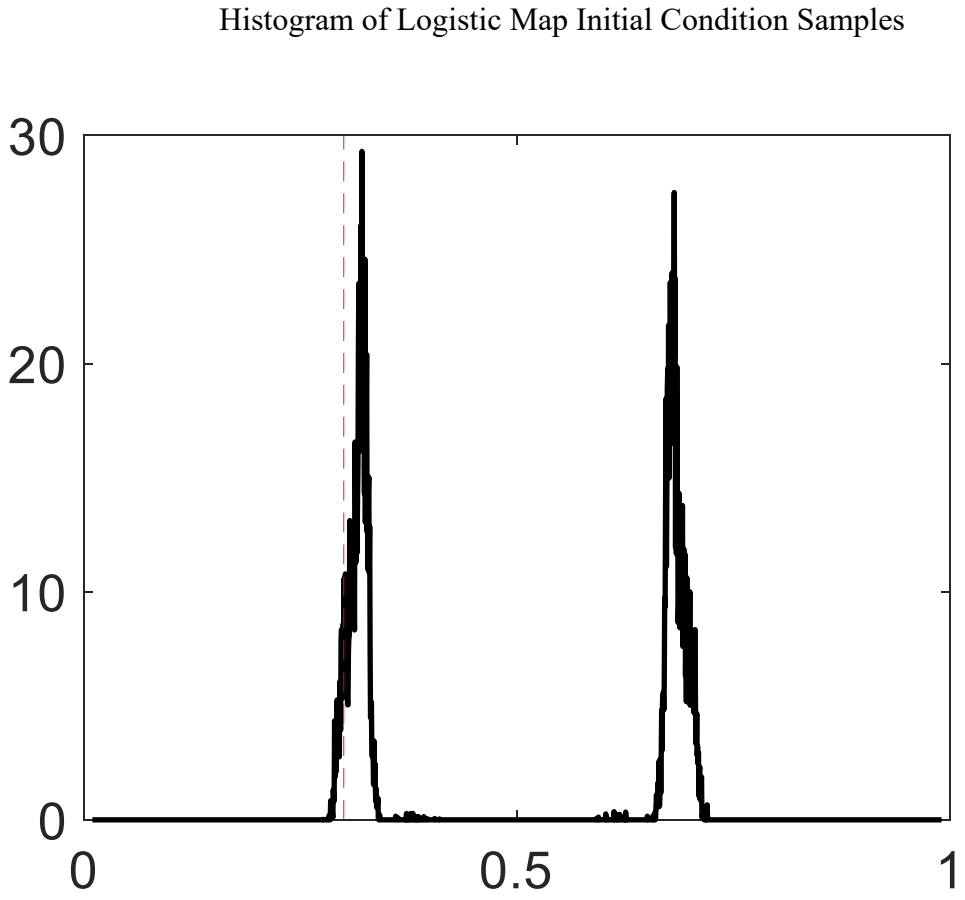

Figure 1. The true initial condition value in this case is v_0=0.3 (red dashed line).

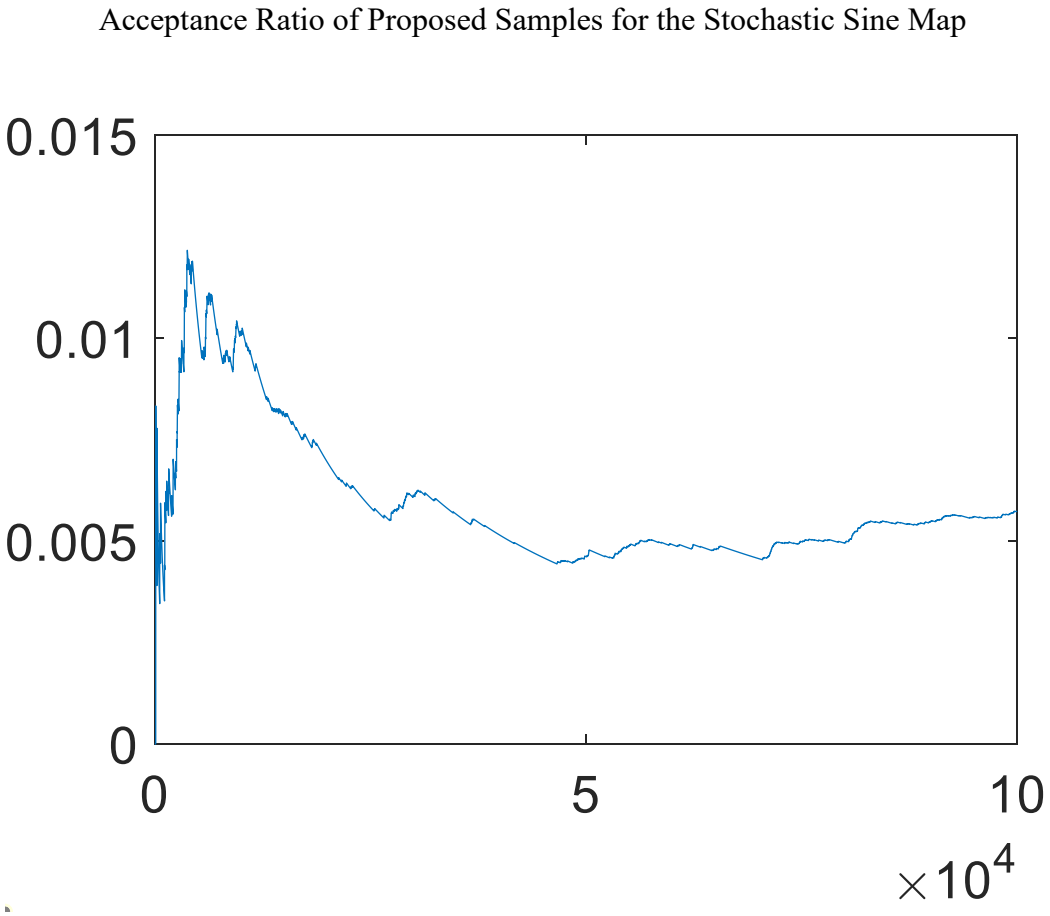

Figure 2: Acceptance ratio of proposed samples decreases over time.

Cumulative Sample Mean of Proposed First Elements for the Stochastic Sine Map

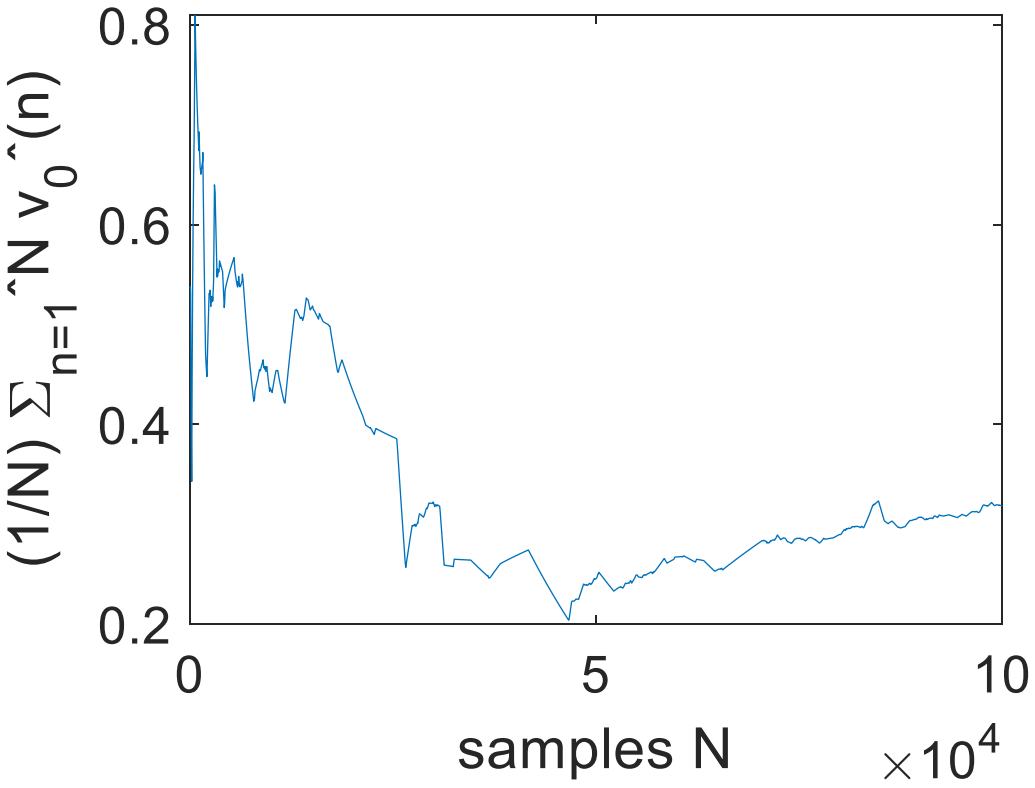

Figure 3: Running average of the first element in proposed samples w.

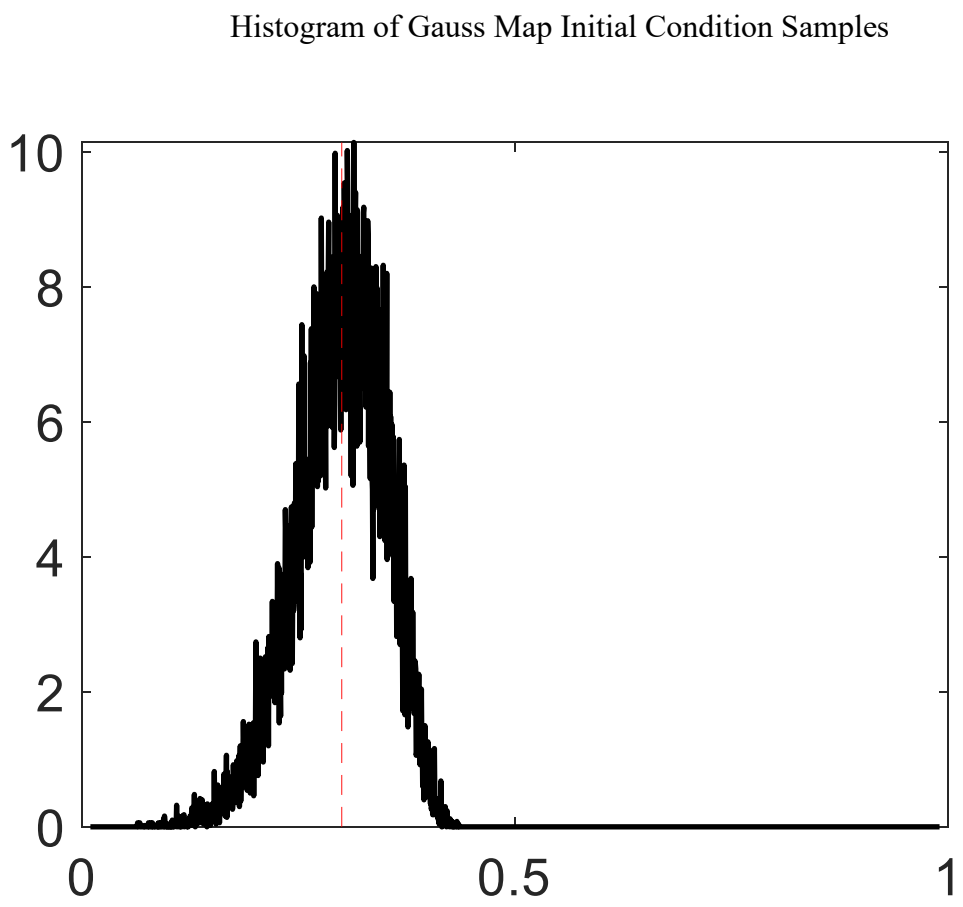

Figure 4: The true initial condition value in this case is $v_0 = 0.3$ (red dashed line). The mean value of the initial condition samples is 0.298126.

Histogram of 2-D Tinkerbell Map Initial Condition Samples

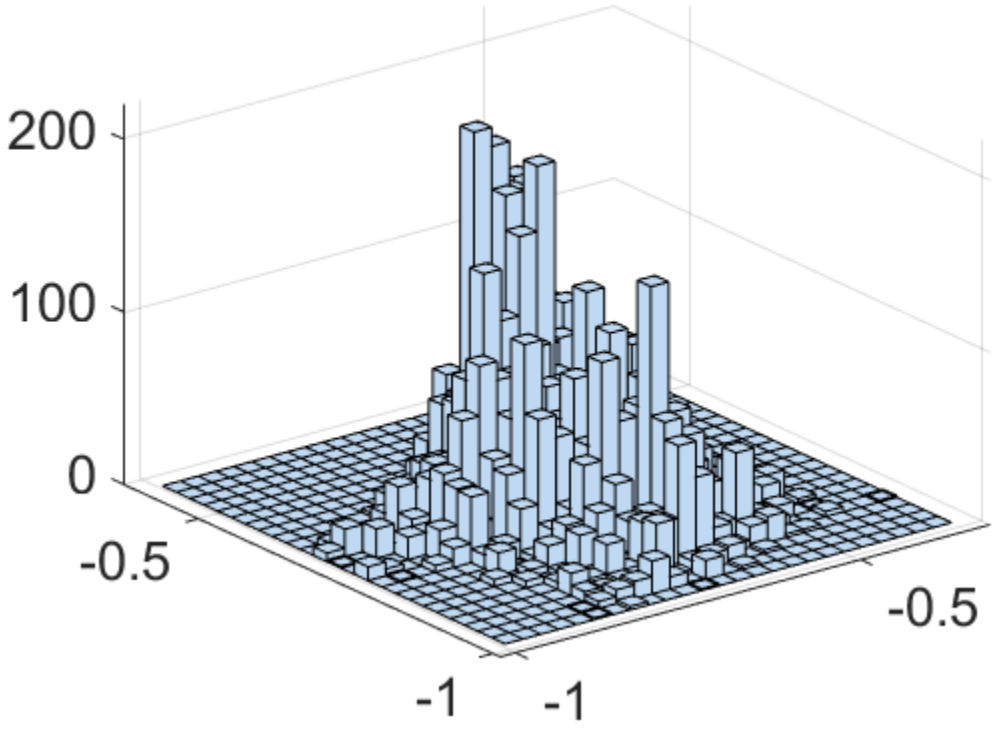

Figure 5: The true initial condition in this case is $x_0 = -0.72, y_0 = -0.64$. The mean value of the initial condition samples is (x,y)=(-0.693226,-0.719511).