Parameter Estimation in Models with Complex Dynamics

Manas Lohani

2019B4A70109G

1 Introduction

1.1 Aim

This project aims to study the estimate the parameters for mathematical models. We use Monte Carlo Markov Chains (Metropolis Hastings Algorithm)for sampling from a distribution proportional to the posterior distribution of these parameters.

1.2 General Mathematical Formalism for the posterior

We will consider a deterministic dynamical model with parameters θ, and any information, I defined as:

$$x_{n+1} = M(x_n, \theta, I) \tag{1}$$

We consider a time series of length t of noisy and partial observations, y_n, defined by:

$$y_n = h(x_n) + \eta_n, n = 1, 2, ..., t, \tag{2}$$

here, $h(x_n)$ is a function of x_n and η_n is the experimental noise. We aim to estimate θ based on the observations $\mathrm{Y} = y_1, ..., y_t$.
We assume that η_n are iid random variables derived from a normal distribution $N(0, \sigma_r^2)$ where σ_r is called the recovery noise. It is also assumed that the prior probability distribution on the parameters θ is uniform within a physically motivated range. The posterior probability distribution would be taken as:

$$p(\theta|Y, I) \propto exp\Big[-\sum_{n=1}^{t} \frac{\|y_n - h(x_n)\|^2}{2\sigma_r^2}\Big] \tag{3}$$

To sample the above posterior the Metropolis Hastings Algorithm is used. We choose initial conditions $x_0 = x_{inj}$ and parameters θ_{inj} to generate a trajectory:

$$x'_{n+1} = M(x'_n, \theta_{inj}, I) \tag{4}$$

Then the data set is given by:

$$y_n = h(x'_n) + \eta_{inj,n}, n = 1, 2, ..., t, \tag{5}$$

Here, η_{inj} are iid random variables derived from a normal distribution $N(0, \sigma^2_{inj})$ where σ_{inj} is called the injected noise.

1.3 Metropolis Hastings Algorithm

We have been given a function proportional to the target distribution in equation (3).
Choose a arbitrary point x_n and sample the next point x* from from a distribution g(x*—x_{tn}). g can be assumed to be a normal distribution centered around x_n. The Acceptance probability of taking a point b after point a is given by : $A(a \to b) = min\Big(1, \frac{f(b)g(a|b)}{f(a)g(b|a)}\Big)$.Also, Since normal distribution is symmetric,$g(a|b) = g(b|a)$.
Now generate a uniform random variable u in [0,1].Now the next point x_{n+1} can be given by :
x* if $u \leq A(x_n \to x*)$,
x_n if $u \geq A(x_n \to x*)$
This is the Metropolis Hastings Algorithm

2 The Logistic Map

The logistic map is an one dimensional discrete-time dynamical model. The population value at any time step t is mapped to the population at the next time step (hence the term 'map') as:

$$x_{n+1} = r(1 - x_n)x_n \qquad x_t \in [0, 1], r \in [1, 4]. \tag{6}$$

Here x_n represents the population value at any time-step n and r is called the growth-rate.

2.1 Bifurcation Diagram

We can generate a bifurcation diagram for the logistic map by plotting the values of x_n as a function of r using a fixed initial value x_0.
Here, as a demonstration, we choose 10000 values of r distributed uniformly on [1,4]. Then we generate a series of x for 200 time-steps for each of these values. Now, we take the x_{200} value and plot it against the r value. We notice that for every $r \in [1, 3]$ x approaches some fixed value. As we increase the value of r, we begin to see period doubling cycles for these values i.e the system oscillates between two values.
As the value of r is increased, one gets 4-period and even 8-period bifurcations. On increasing the value of r even further beyond, it seems that the population can land on any random value and it leads to chaos. One can observe that

different values of r_{inj} will give different series that can be periodic or chaotic. The following code generates the bifurcation diagram for the logistic map:

```
import numpy as np
import matplotlib.pyplot as plt
R = np.linspace(1,4,10000) #For setting 10000 values of r
X = []
Y = []
for r in R:
    X.append(r)
    x =np.random.random()
    for n in range(100):
        x = r*x*(1-x)
    for n in range(100):
        x = r*x*(1-x)
    Y.append(x)
plt.plot(X,Y, ls='', marker = ',', c ='#4CAF50' )
plt.show
```

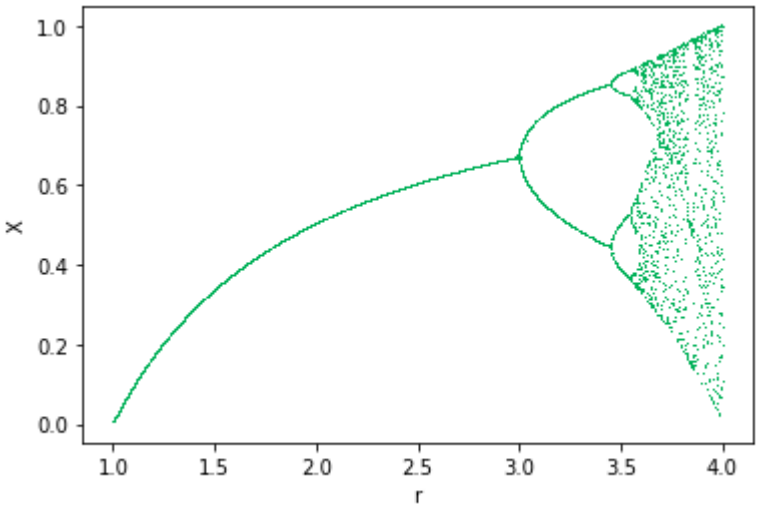
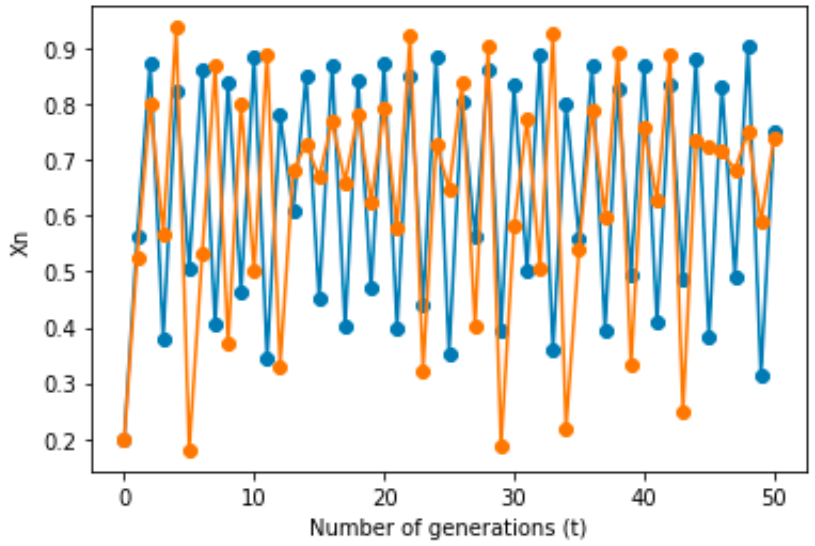


Figure 1: Bifurcation Diagram for logistic map

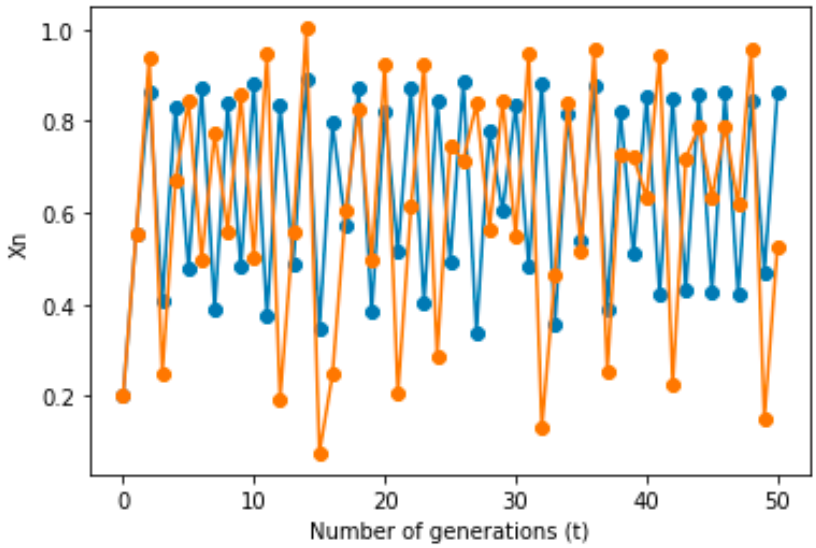
2.2 Generating Periodic and Chaotic data

Figure (1) shows us that different values of r_{inj} can give rise to either a periodic or a chaotic data series. We shall use $r_{inj} = 3.5$ for generating periodic data and $r_{inj} = 3.7$ for chaotic data in our parameter estimation problem. We generate the data set using Equation 5. We choose the initial value $x_0 = 0.2$. In the generated plot for the same, we use blue color to represent $\sigma_{inj} = 0.01$ and orange color to represent $\sigma_{inj} = 0.05$.

```
#Code for plotting Periodic data generated with r = 3.5
from matplotlib import pyplot as plt
import numpy as np

# initialize an array of 0s and specify starting values and r constant
steps = 50
x = np.zeros(steps + 1) #array for small noise
y = np.zeros(steps + 1)
x[0], y[0] = 0, 0.2

x_2 = np.zeros(steps + 1) #initializing array for large noise
y_2 = np.zeros(steps + 1)
x_2[0], y_2[0] = 0, 0.2
r = 3.5

# loop over the steps and replace array values with calculations
for i in range(steps):
   y[i+1] = 3.5 * y[i] * (1 - y[i]) + np.random.normal(0,0.01)
   x[i+1] = x[i] + 1

for i in range(steps):
   y_2[i+1] = r * y_2[i] * (1 - y_2[i]) + np.random.normal(0,0.05)
   x_2[i+1] = x_2[i] + 1

# plot the figure!
plt.plot(x,y,'-o')
plt.plot(x_2,y_2,'-o')
plt.xlabel("Number of generations (t)")
plt.ylabel("Xn")
plt.show()
```



(a) Periodic data generated with r = 3.5



(b) Chaotic data generated with r = 3.7

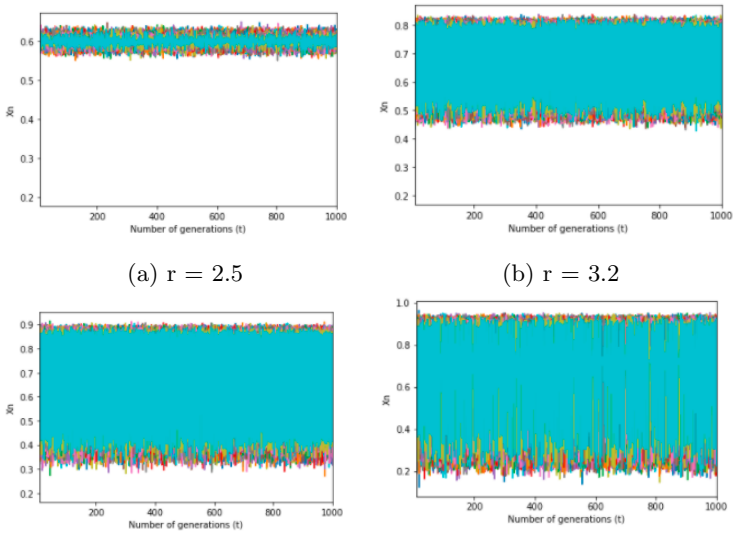
Figure 2: The blue dots connected by the blue line) represent the data Y for the injected noise $\sigma_{inj} = 0.01$, and the orange dots (connected by the orange line) represent data Y for the injected noise $\sigma_{inj} = 0.05$

2.3 Probability distribution for Generated Trajectory

Now, we will plot the probability distribution for x'_n generated using different values of r. To do so, we generate 100 time series of 1000 time-steps for x_n .

The values of r are selected as follows:
Using the initial condition of $x_0 = 0.2$ we select values of r_{inj} that would generate different forms of trajectories for our analysis. We know that the trajectory for the logistic map settles on a fixed value for $1 < r < 3$.
Hence let us pick $r_{inj} = 2.4$. We can also check that for $r_{inj} = 3.2$ we will get a two period cycle on which the generated trajectory x'_{n+1} would settle down on. Similarly, $r_{inj} = 3.5$ gives us a 4-period cycle and $r_{inj} = 3.7$ would generate chaotic data.
We can plot the 100 stochastic data series that we have thus generated and plot them for different values of r. We shall plot the values following x_{10} in order to get cleaner graphs for better analysis. The given code generates the required plot:

```
#Code for plotting Periodic data generated with r = 3.5
from matplotlib import pyplot as plt
import numpy as np
from scipy.stats import norm
# initialize an array of 0s and specify starting values and r constant
steps = 100
x = np.zeros(steps + 1) #array for small noise
y = np.zeros(steps + 1)
x[0], y[0] = 0, 0.2

x_2 = np.zeros(100) #initializing array
r = 3.5
#
# loop over the steps and replace array values with calculations
for j in range(100):
    for i in range(steps):
        y[i+1] = 3.55 * y[i] * (1 - y[i]) + np.random.normal(0,0.01)
        x[i+1] = x[i] + 1
    plt.plot(x,y)
    plt.xlim(10,100)
    plt.xlabel('Xn')
    plt.ylabel('Number of generations (t)')
    x_2[j] = y[steps]
```



(a) r = 2.5 (b) r = 3.2

(c) r = 3.5 (d) r = 3.7

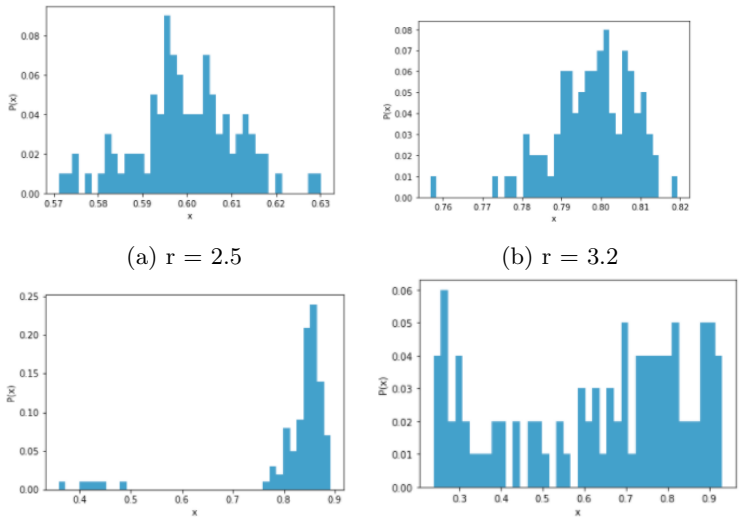
Figure 3: The Stochastic data series plots for different values of r.

Now, in order to plot the probability distribution of X, wee take up the values at a particular time-step for a particular value of r_{inj} and plot the probability distribution of x_n using a histogram.
We do so for different injected parameters that give us periodic or chaotic data. We can look at X values retrieved by this method and plot the required histogram by adding the following snippet by the addition of the following code:

```
print(x_2)
# plotting the graph
weights = np.ones_like(x_2)/float(len(x_2))
plt.hist(x_2, bins= 30, weights=weights)
plt.xlabel('x_n')
plt.ylabel('P(x)')
plt.show()
```

```
# The above code will generate the x values for example:
[0.58353144 0.83654638 0.85343205 0.81112687 0.82639518 0.88304384
 0.8808908 0.780906  0.87067182 0.76774117 0.8802967 0.81923283
 0.8685524 0.85844897 0.88578574 0.85589801 0.87761139 0.85385802
 0.50210435 0.89795969 0.80378327 0.87784731 0.90013043 0.85273173
 0.80786313 0.73651642 0.8630926 0.85040307 0.89484014 0.86728292
 0.88164587 0.76794253 0.80941637 0.82206856 0.91161461 0.76518639
 0.85087021 0.8882165 0.87144958 0.88265098 0.84645462 0.86714246
 0.36935019 0.8730082 0.77685998 0.85354389 0.85981544 0.87129979
 0.87502538 0.7729177 0.87091161 0.86639091 0.85151529 0.8406177
```

```
0.40331157 0.84632313 0.46954514 0.84205759 0.81035666 0.82582576
0.87423233 0.86844131 0.82699276 0.88406356 0.88737952 0.86769883
0.82873294 0.78857545 0.76544375 0.85413525 0.89310512 0.84924796
0.88183761 0.85455272 0.81612347 0.37611805 0.84341461 0.84201978
0.82431704 0.82923864 0.80678051 0.88440198 0.88406234 0.82211454
0.82774494 0.84343617 0.85973461 0.8812268 0.90370664 0.84207948
0.86385214 0.80817292 0.88469397 0.82728802 0.88355583 0.83687046
0.76602178 0.83337709 0.8270606 0.88452573]
```



(a) r = 2.5 (b) r = 3.2

(c) r = 3.5 (d) r = 3.7

Figure 4: The plot for the distribution of x for various values of r.

We observe that for r = 2.5, the generated trajectory for x_n is centered at around 0.6. For r = 3.2, we have a broader trajectory with the mean value falling at around 0.8. In the case of r = 3.5, We get the bulk of our x_n values to be around 0.82 . The chaotic data generated using r = 3.7 gives us values spread out throughout the range [1,4] .

2.4 Sampling using Metropolis Hastings Algorithm

We use the posterior probability distribution for the parameter r as defined in equation (3). We then use the initial conditions of $x_0 = 0.2, r_0 = 3.2, \sigma_{inj} = \sigma_r = 0.01$

We shall propose a Normal distribution centered around r_n for picking up values of r_* in the Markov chain and then decide the values of r_{n+1} accordingly for the Metropolis Hastings Algorithm. We generate 1000 samples and then plot the posterior probability distribution for r. The given code aims at sampling from the logistic map using the Metropolis Hastings Algorithm as defined above:

```
import matplotlib.pyplot as plt
import numpy as np
import math
# this is the distribution proportional to the posterior
def posterior(r):
    sigma_inj, sigma_r = 0.01, 0.01
    steps = 50
    y = np.zeros(steps + 1,dtype = 'int64')
    y_2 = np.zeros(steps +1,dtype = 'int64')
    y[0], y_2[0] = 0.2, 0.2
    sum = 0
    for i in range(steps):
        n_inj = np.random.normal(0, sigma_inj)
        y[i + 1] = 3.5 * y[i] * (1 - y[i]) + n_inj #r random uniform +
            some normal noise
        y_2[i + 1] = r * y_2[i] * (1 - y_2[i]) #r constant = 3.5
        sum = sum + (y[i + 1] - y_2[i + 1])** 2 #(y-y_2 squared)
    return np.exp(- (sum / (2.0 * (sigma_r ** 2))))

N = 1000 #1000 samples
r_0 = 3.2 #initial values
p = posterior(r_0)
samples = []
#Metropolis Algorithm
for i in range(N):
    rn = np.random.normal(r_0,1)
    if(rn < 1 or rn > 4):
        rn = r_0
    pn = posterior(rn)
    if pn >= p:
        p = pn
        r_0 = rn
    else:
        u = np.random.rand()
        if u < pn/p:
            p = pn
            r_0 = rn
```

```
    samples.append(r_0)
plt.hist(samples, bins=100, density = True)
plt.xlim(2,4)

plt.title('Metropolis Hastings sampling')
plt.grid()
plt.show()
plt.close()
```

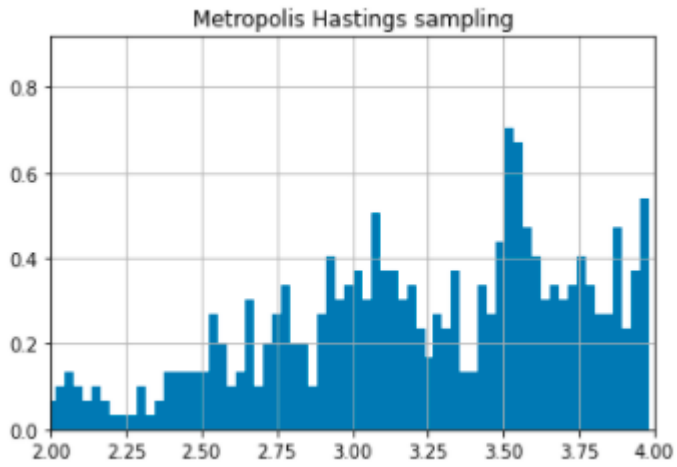


Figure 5: The posterior probability distribution for r on sampling using periodic data.

We can also use r = 3.7 and sample the posterior distribution for r using chaotic data instead. It has been observed that in case of periodic data the mode is close to the $r_{inj} = 3.5$. No such observation is made for the chaotic data which makes parameter estimation for the same difficult.
It is observed that changing the initial conditions doesn't bring about a significant affect in the posterior analysis using chaotic data.