Information Theory (UAI/500)

Lecture Notebook

Lecturer: Ivo Bukovsky

Something about Data and Systems that Generate them

- Data generating systems:
  - Stochastic (random process),
  - Deterministic
  - Chaotic (deterministic chaos,logistic eq. example)
  - Real (...omg)
- Anomaly detection: probabilistic or machine learning based ?

Note on Deterministic Chaos

It can be good to aware of what deterministic chaos is (i.e., an unpredictable behavior of data generated by a deterministic governing law), just be aware it exists, ...)

Example of a very complex dynamical systems, or a very complex governing lar in machine learning are reccurent neural networks (the ones with some kind of feedback, tapped delays,...). However do not panic, chaotic development in them can be very rare...

Example of Logistic Equation

The behavior of data (here scalar variable y) is driven by a deterministic governing law as follows

$$y(k) = a \cdot y(k-1) \cdot (1 - y(k-1), \tag{1}$$

where k is diskrete index of time, y is generated data (e.g., it has been demonstrated that extended to 3-D problem it works as an insect population [2]), a is called a bifurcattion parameter (it changes qualitative behavior of generated data).

- What happens if we change a?
- What happens if we keep $a = const.$ and change the initial condition $y(k=0)$ to $y(0) = y(0) + \text{"a butterfly"}$ where the "butterfly" is a very small number?

```
%matplotlib inline
#%matplotlib notebook
from numpy import *
from matplotlib.pyplot import *


a=4  # biffurcation parameter
N=10000
y=zeros(N)
y[0]=.1  # init. cond.
for k in range(1,N):
    y[k]=a*y[k-1]*(1-y[k-1])

z=zeros(N)
z[0]=0.1000001  # init. cond.
for k in range(1,N):
    z[k]=a*z[k-1]*(1-z[k-1])

figure(figsize=(10,4))
subplot(311)
plot(y,'.')
subplot(312)
plot(range(N-200,N),y[-200:],'.-'),xlabel('k')
subplot(313)
plot(range(N-200,N),z[-200:],'.-'),xlabel('k')
show()

```

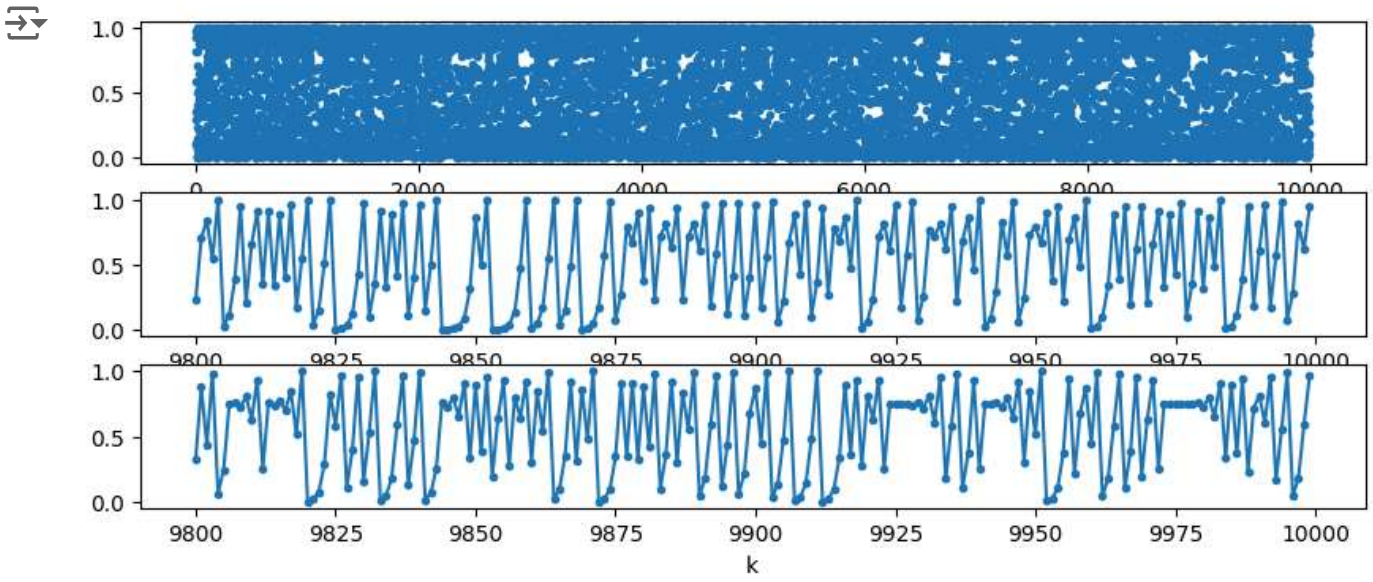
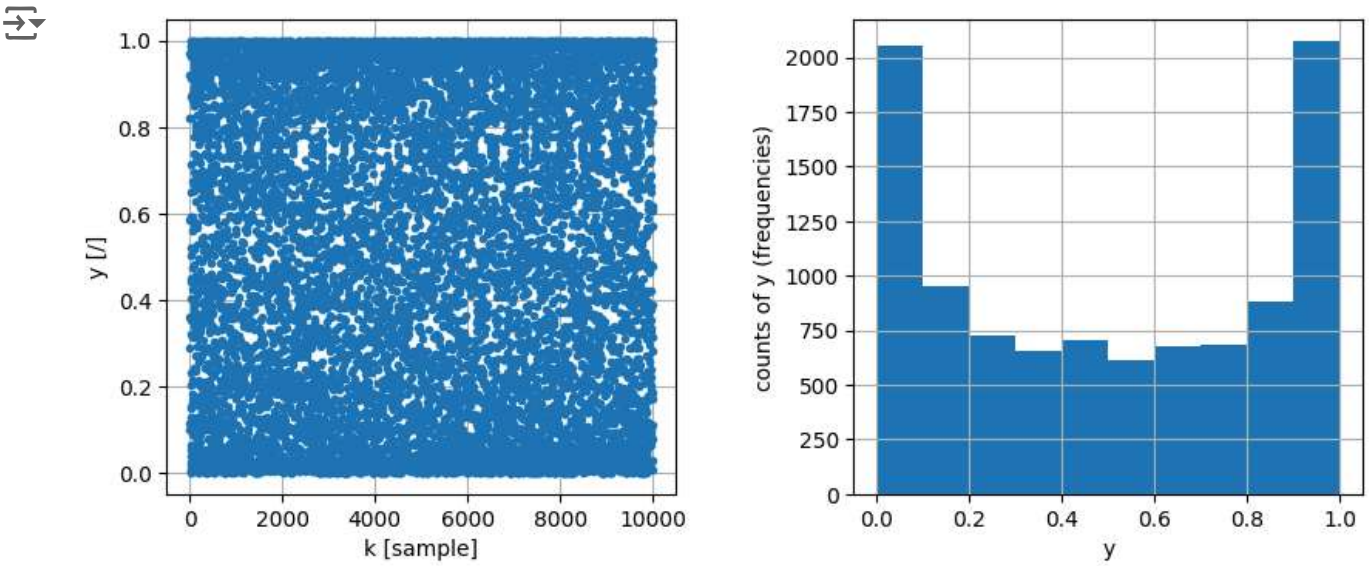


Fig. 1: The detail of last 200 samples differs due to a very small distinction in initial conditions ($y(k=0)=0.1, z(k=0)=y(k=0)+0.000001$, i.e. this is the butterfly efffect where 0.000001 is the "butterfly" here. Also, notice, that the flat data behavior around $k=9925$ or

$k = 9975$ may seems to be an anomaly in sense of probability, but not in sense of the governing law that is the same for all k. (and may be learned from data ...if done properly...)

```
figure(figsize=(10,4))
subplots_adjust(wspace=.35)
subplot(121)
xlabel("k [sample]"); ylabel("y [/]")
plot(y,'.');grid()
subplot(122)
hist(y,10)
grid( );xlabel("y");ylabel("counts of y (frequencies)")
show()
```




```
savetxt("y.txt",y)
```

Example L-7

Let's play a little bit with theoretical data generated by chaotic system (1) and practice some basic statistical techniques.

Generate or use those 10 000 samples by system (1) for random initial condition $0 < y(0) < 1$ with $a = 4$ (as done above).

Now, let's forget about how you generated time series y and consider y be a continuous random variable observed at every sample time $k = 0, 1, 2, \ldots, 9\ 999$.

Sliding window excersize:

1. The time serie is long $N = 10\ 000$ data samples. Calculate and draw mean trajectory and variance trajectory for a sliding window of lenght 5,10,30,100,1000 samples with 1-sample

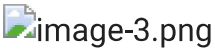
sliding. What is the shortest window where mean and variance are stationary? (stationaty in a loose sense, show and discuss your result and your opinion)**[0.6 Points]**

1-D probability:

2. Estimate the probability mass function via histogram $p(y)\ \forall y$ and discuss your selection of number of bins (reason for your choice, how does it affect $p(y)$? **[0.4 Points]**
3. Calculate and plot the probability of each individual sample point $y(k)\ \forall k$, i.e., horizontal axis is time index $k = 1, 2, \ldots, N$, vertical axis shows $p\big(y(k)\big)$ **[0.6 Points]**

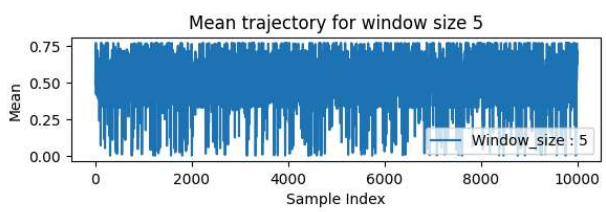
2-D probability:

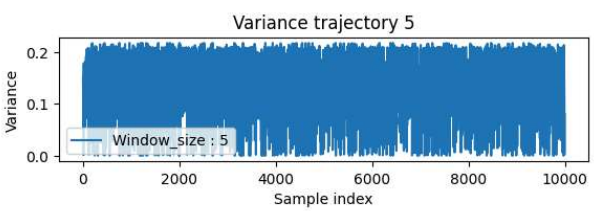
4. Estimate the 2-D probability mass function $p\big(y, y_{previous})\big)\ \forall y$, show the 2-D plot and briefly discuss your solution (you can use Numpy or some existing libraries(reference the library and briefly explain the principles that the library applies). **[0.8 Points]**
5. Estimate the 2-D conditional probability function $p\big(y|y_{previous})\big)\ \forall y$, show the 2-D plot and briefly discuss your solution (you can use Numpy or some existing libraries(reference the library and briefly explain the principles that the library applies):
   a) Estimate $p\big(y|y_{previous}\big)$ directly from the sequence of data (time series) **[0.8 Points]**
   a) Estimate $p\big(y|y_{previous}\big)$ by the Bayes rule using $p\big(y, y_{previous}\big)$ and $p(y_{previous})$ **[1 Points]**
6. Calculate and plot the probability of each individual data point $\mathbf{x}(k) = [y(k), y(k-1)]$ where horizontal axis is time index k, and the vertical axis shows $p([y(k), y(k-1)])$**[0.6 Points]**

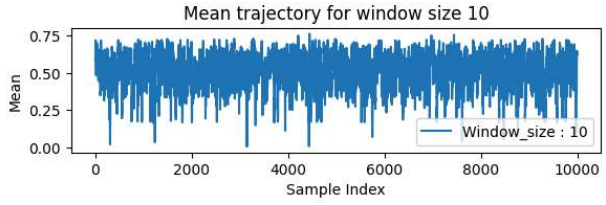


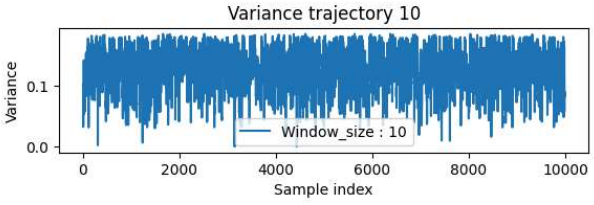

```
 1 import numpy as np
 2 import pandas as pd
 3 import matplotlib.pyplot as plt
 4
 5 """Sliding window excersize:
 6 1) The time serie is long  N=10 000  data samples. Calculate and draw mean trajectory
 7     What is the shortest window where mean and variance are stationary?"""
 8
 9 a=4  # biffurcation parameter
10 N=10000
11
12 initial_condition = np.random.rand()  # Random initial condition 0 < y(0) < 1
13 y = np.zeros(N)
14 y[0] = float(initial_condition)
15
16 for k in range(1,N):
17   y[k] = a * y[k-1] * (1-y[k-1])      # logistic map function
18
19 window_sizes = [5, 10, 30, 100, 1000]  # window sizes
20
21 # Function to calculate mean and variance for time series data
22 def window_sliding(data, window_size):
```

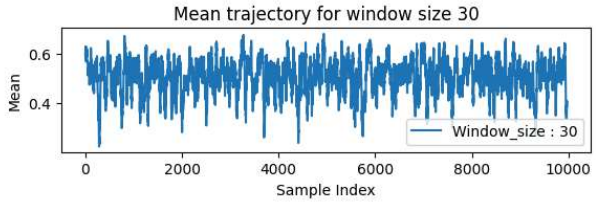
```
  series = pd.Series(data)
  mean_rolling = series.rolling(window = window_size).mean().dropna().values
  var_rolling = series.rolling(window = window_size).var().dropna().values
  return mean_rolling, var_rolling

result = {}
for i in window_sizes:
  mean, variance = window_sliding(y, i)
  result[i] = (mean, variance)


# plotting the function
plt.figure(figsize=(15, 12))

for i, window_size in enumerate(window_sizes):
  means, variances = result[window_size]

  plt.subplot(5,2,2*i+1)
  plt.plot(means, label = f'Window_size : {window_size}')
  plt.title(f'Mean trajectory for window size {window_size}')
  plt.xlabel('Sample Index')
  plt.ylabel('Mean')
  plt.legend()

  plt.subplot(5,2,2*i+2)
  plt.plot(variances, label = f'Window_size : {window_size}')
  plt.title(f'Variance trajectory {window_size}')
  plt.xlabel('Sample index')
  plt.ylabel('Variance')
  plt.legend()

plt.subplots_adjust(wspace=0.3, hspace=0.7)
plt.show()
```

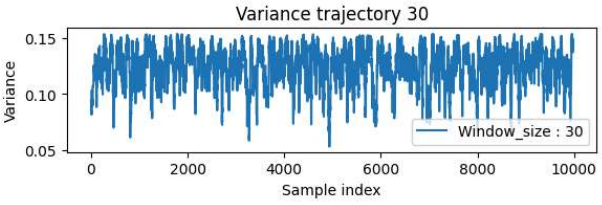

Mean trajectory for window size 5
Mean
Sample Index
Window_size : 5

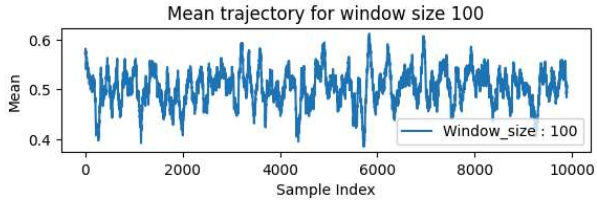

Variance trajectory 5
Variance
Sample index
Window_size : 5

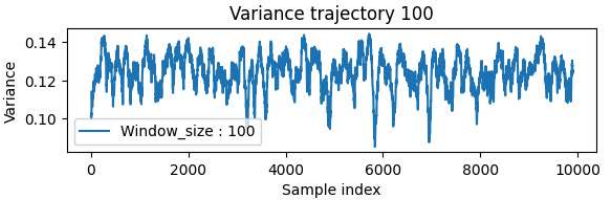

Mean trajectory for window size 10
Mean
Sample Index
Window_size : 10

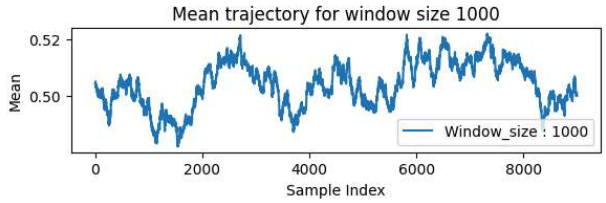

Variance trajectory 10
Variance
Sample index
Window_size : 10

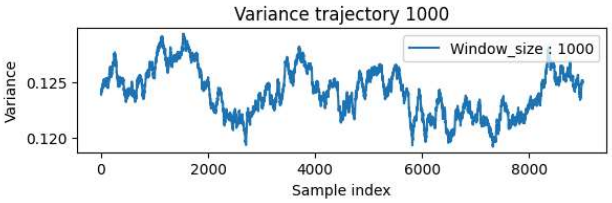

Mean trajectory for window size 30
Mean
Sample Index
Window_size : 30


Variance trajectory 30
Variance
Sample index
Window_size : 30


Mean trajectory for window size 100
Mean
Sample Index
Window_size : 100


Variance trajectory 100
Variance
Sample index
Window_size : 100


Mean trajectory for window size 1000
Mean
Sample Index
Window_size : 1000


Variance trajectory 1000
Variance
Sample index
Window_size : 1000

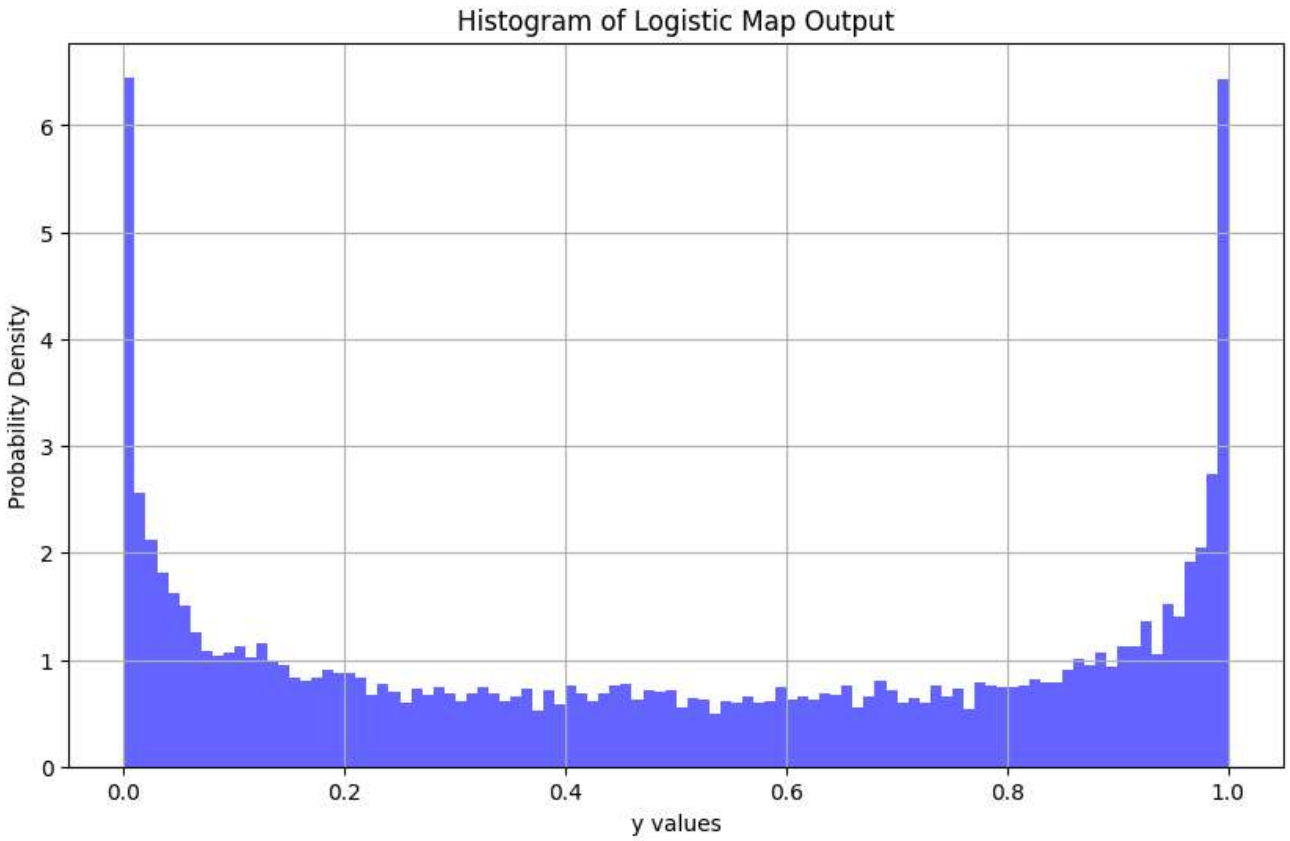
Window size 5 and 10 - The trajectories exhibit high fluctuations due to the small window size. The time series shows chaotic behavior and these small windows are not enough to smooth out the noise. The variance fluctuates wildly, indicating non-stationarity.

Window size 30 - The fluctuations in both mean and variance start to decrease, but there is still some noticeable variation. This suggests some initial stabilization, but the system has not fully reached stationarity.

Window size 100 - Both the mean and variance appear to be more stable compared to smaller window sizes. Although there are still minor variations, the system is much closer to stationarity

Window size 1000 - Here, the mean and variance show very little change over time. This large window size effectively smooths the data, suggesting that the system has become stationary.

```
"""2) Estimate the probability mass function via histogram  p(y) ∀y  and discuss your
"""

num_bins = int(np.sqrt(N))

plt.figure(figsize=(10, 6))
plt.hist(y, bins=num_bins, density=True, alpha=0.6, color='b')

# Adding titles and labels
plt.title('Histogram of Logistic Map Output')
plt.xlabel('y values')
plt.ylabel('Probability Density')
plt.grid()
plt.show()
```



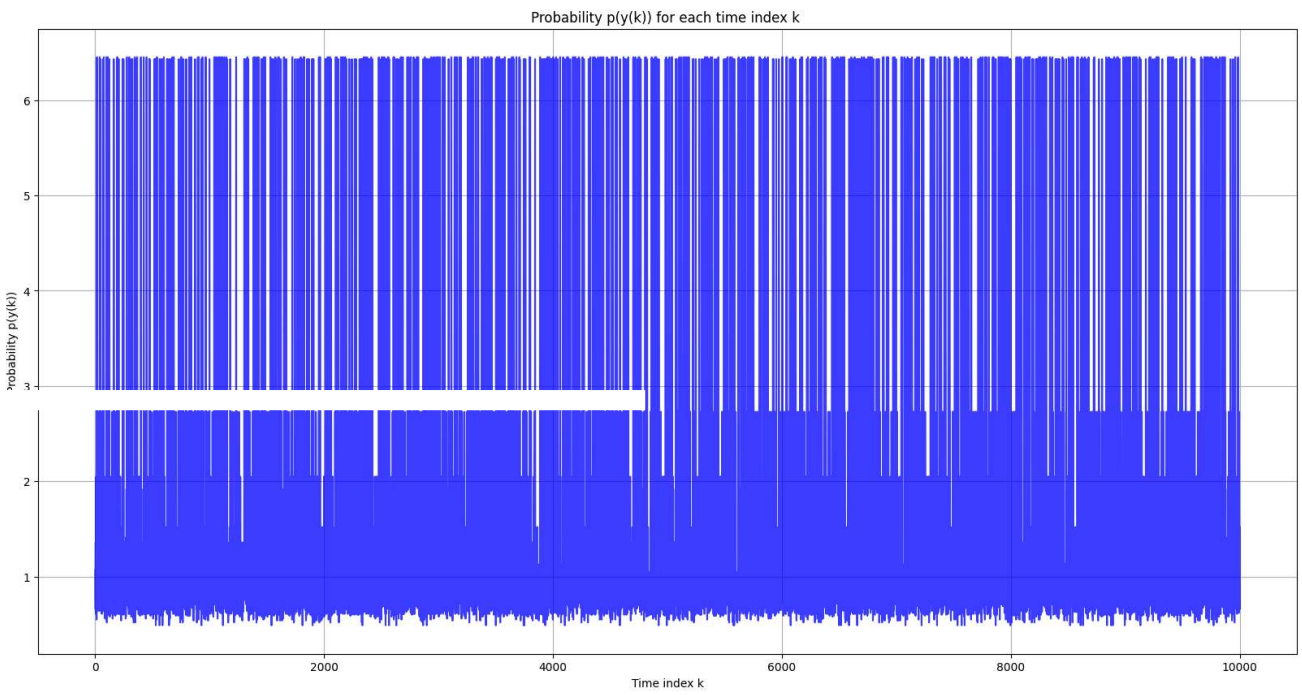
I have used the square root of the sample size as the appropriate choice for selecting the bins as it balances the detail and smoothness in the distribution.

If the number of bins is too low, the histogram will be overly simplistic, failing to capture the nuances of the distribution. Important features such as peaks and valleys may be lost, making it hard to discern the actual shape of the PMF.

Conversely, using too many bins can result in a noisy histogram that fluctuates widely. This is particularly problematic with limited data points, as random variations may appear as significant features of the distribution which can mislead interpretations.

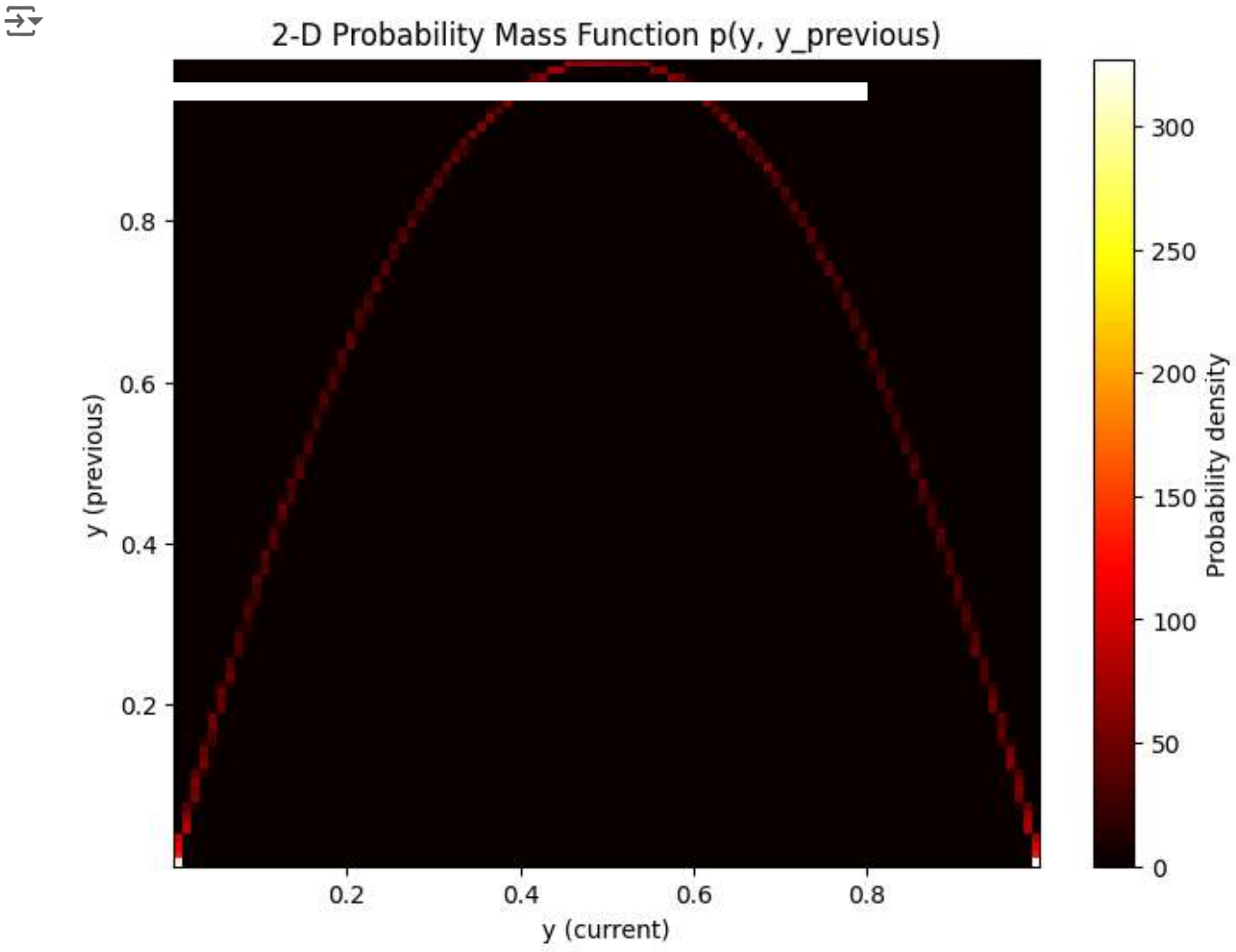
```
""" 3)Calculate and plot the probability of each individual sample point  y(k) ∀k , i.
"""

num_bins = int(np.sqrt(N))  # Number of bins
hist, bin_edges = np.histogram(y, bins=num_bins, density=True)

bin_indices = np.digitize(y, bin_edges[:-1]) - 1  # Bins are indexed from 0
bin_indices[bin_indices >= num_bins] = num_bins - 1  # Correct out-of-bound indices

# Get the probability for each y(k) based on the corresponding bin
prob_yk = hist[bin_indices]
```

```
12
13 # Step 3: Plot p(y(k)) for each k (time index)
14 plt.figure(figsize=(20, 10))
15 plt.plot(range(1, N+1), prob_yk, color='blue', alpha=0.75)
16 plt.title('Probability p(y(k)) for each time index k ')
17 plt.xlabel('Time index k')
18 plt.ylabel('Probability p(y(k))')
19 plt.grid(True)
20
21 plt.show()
```




```
1 """4) Estimate the 2-D probability mass function  p(y,yprevious)) ∀y  , show the 2-D p
2 """
3
```

```
 4 y_previous = y[:-1]  # y(k-1)
 5 y_current = y[1:]    # y(k)
 6
 7 # Step 2: Compute the 2-D histogram to estimate joint distribution p(y, y_previous)
 8 num_bins = int(np.sqrt(N))  # Set the number of bins
 9 hist, xedges, yedges = np.histogram2d(y_previous, y_current, bins=num_bins, density=Tr
10
11 # Step 3: Plot the 2-D PMF as a heatmap
12 plt.figure(figsize=(8, 6))
13 plt.imshow(hist.T, origin='lower', extent=[xedges[0], xedges[-1], yedges[0], yedges[-1
14 plt.colorbar(label='Probability density')
15 plt.title('2-D Probability Mass Function p(y, y_previous)')
16 plt.xlabel('y (current)')
17 plt.ylabel('y (previous)')
18 plt.grid(False)
19 plt.show()
```

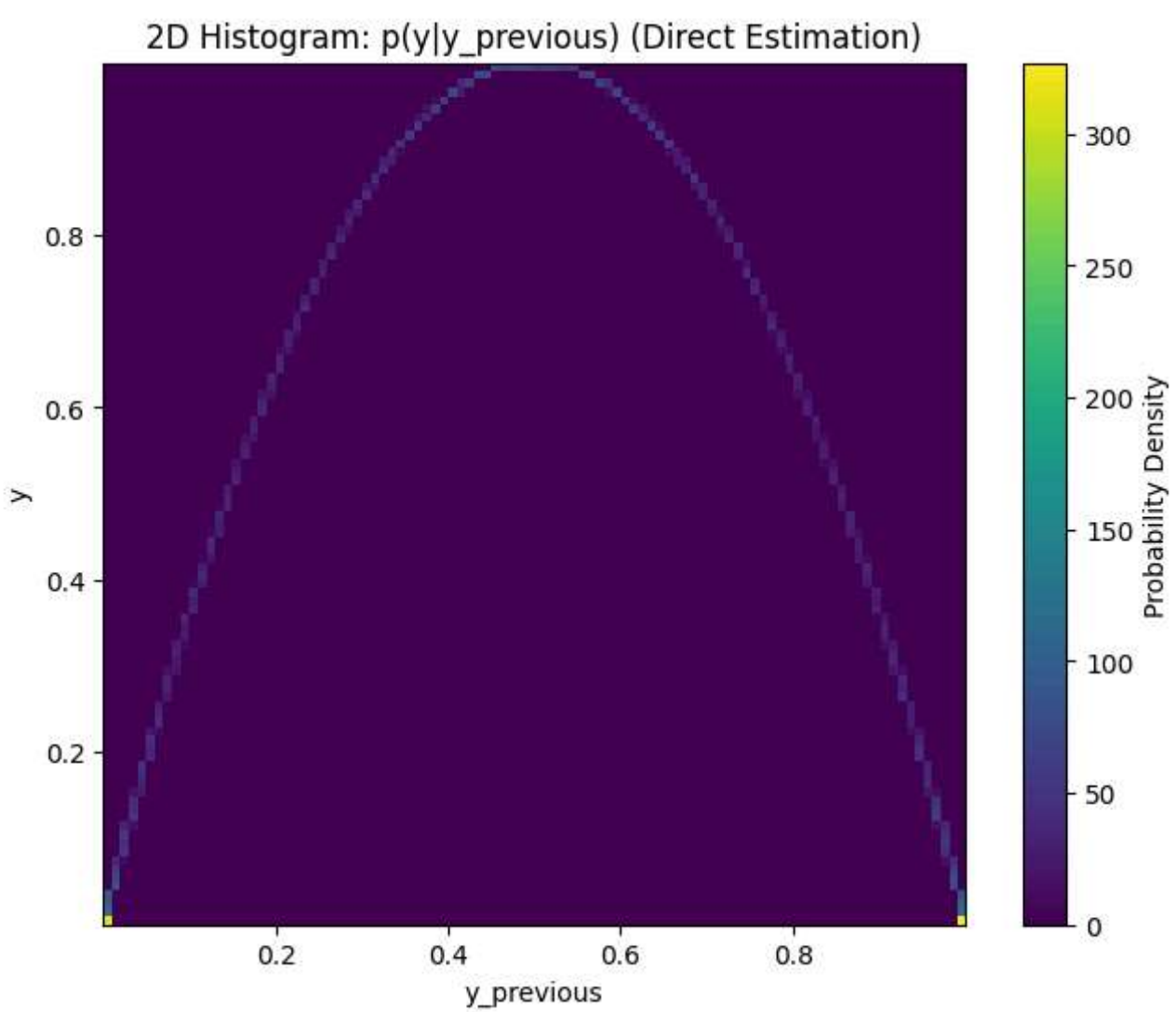


The plot exhibits a structured pattern, particularly with prominent regions in the lower-middle of the graph. This behavior is a result of the chaotic nature of the logistic map at a=4, where the system oscillates between different regions of the state space.

The relationship between y and yprevious shows significant density in certain areas, reflecting where the system spends more time.

We used numpy.histogram2d to compute the joint histogram of y and ypreviousBy normalizing the counts into probabilities, we obtain a 2D PMF, which is visualized with a heatmap using matplotlib. This approach allows us to estimate the density of points in the phase space, capturing the dynamics of the system.

```
""" 5) Estimate the 2-D conditional probability function  p(y|yprevious)) ∀y  , show t
a) Estimate  p(y|yprevious)  directly from the sequence of data (time series)

"""
y_previous = y[:-1]  # y(k-1)
y_current = y[1:]    # y(k)

num_bins = int(np.sqrt(N))  # Set the number of bins for histogram

# 2D histogram (joint distribution of y and y_previous)
hist, xedges, yedges = np.histogram2d(y_previous, y_current, bins=num_bins, density=Tr

# Plotting the 2D histogram
plt.figure(figsize=(8, 6))
plt.imshow(hist.T, origin='lower', cmap='viridis', extent=[xedges[0], xedges[-1], yedg
plt.colorbar(label='Probability Density')
plt.title('2D Histogram: p(y|y_previous) (Direct Estimation)')
plt.xlabel('y_previous')
plt.ylabel('y')
plt.show()
```



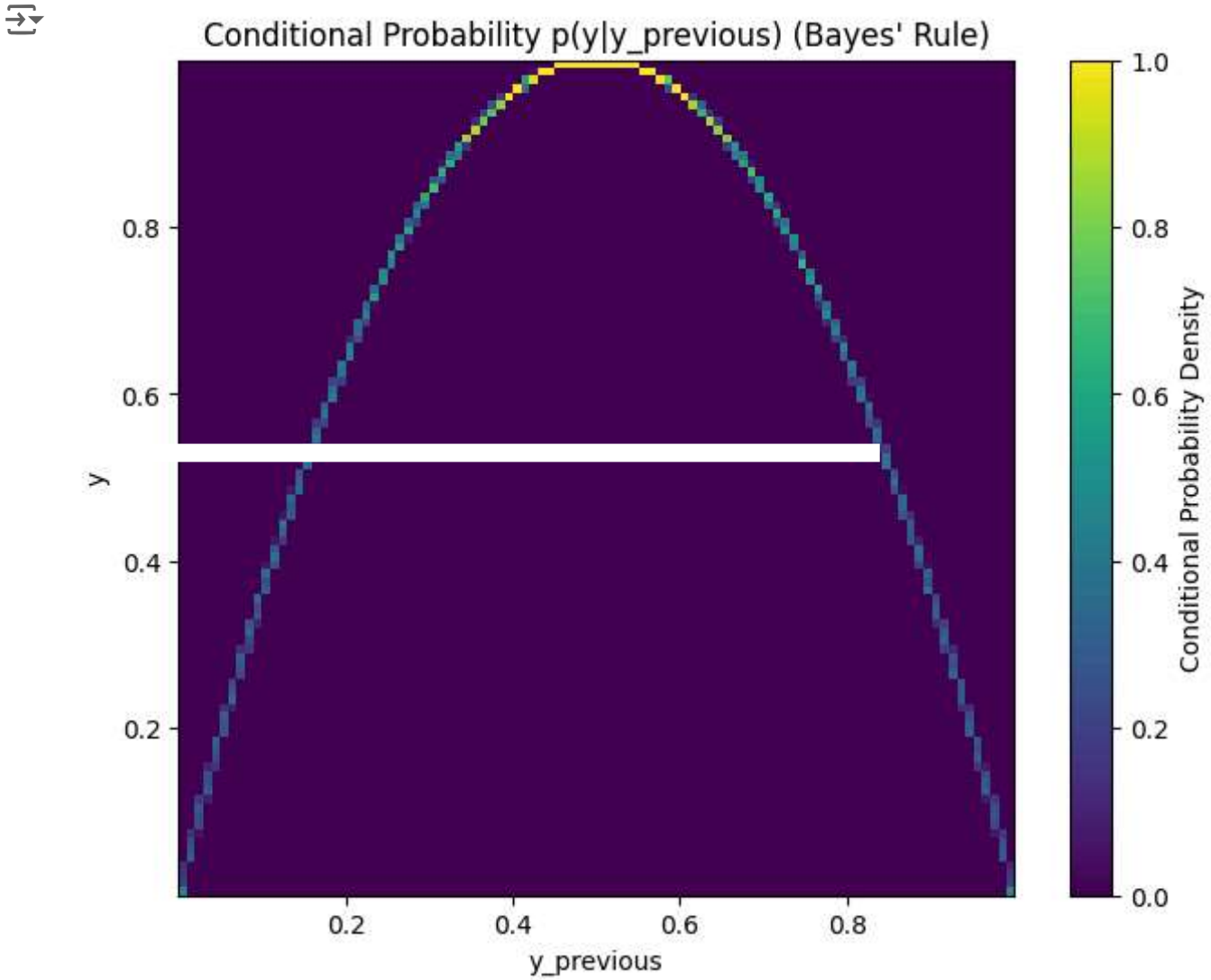
This method simply counts the occurrences of each pair (yprevious,y) and normalizes the counts to represent the probability density.

In the direct estimate, the conditional probabilities are approximated but may not sum to 1 for each yprevious. This could lead to inaccuracies in interpreting the probability distribution.

The direct estimate might reflect the raw frequencies more accurately, but it's also more susceptible to sampling noise.

```
"""5 b) Estimate  p(y|yprevious)  by the Bayes rule using  p(y,yprevious)  and  p(ypre
"""

# Calculate the marginal distribution p(y_previous)
p_y_previous = np.sum(hist, axis=1)  # Marginalize over y (sum across rows)

# Avoid division by zero by ensuring no zero values in p_y_previous
p_y_previous = np.where(p_y_previous == 0, 1e-10, p_y_previous)

# Use Bayes' rule to calculate p(y|y_previous)
p_y_given_y_previous = hist / p_y_previous[:, np.newaxis]

# Plot the conditional probability p(y|y_previous) using Bayes' rule
```

```
plt.figure(figsize=(8, 6))
plt.imshow(p_y_given_y_previous.T, origin='lower', cmap='viridis', extent=[xedges[0],
plt.colorbar(label='Conditional Probability Density')
plt.title('Conditional Probability p(y|y_previous) (Bayes\' Rule)')
plt.xlabel('y_previous')
plt.ylabel('y')
plt.show()
```



This method ensures that for each value of yprevious, the sum of the conditional probabilities across all possible values of y is 1.

By explicitly normalizing the joint distribution using the marginal p(yprevious), this method gives a more accurate and consistent representation of the conditional probabilities.

The resulting graph is smoothed and better conforms to the probabilistic interpretation, removing noise and better highlighting the regions where the probabilities are highest.

```
""" 6) Calculate and plot the probability of each individual data point  x(k)=[y(k),ɔ
      where horizontal axis is time index  k , and the vertical axis shows  p([y(k),ɔ
"""

y_previous = y[:-1]  # y(k-1)
y_current = y[1:]    # y(k)
```

```
 7
 8 # Step 2: Compute the 2-D histogram to estimate joint distribution p(y, y_previous)
 9 num_bins = int(np.sqrt(N))  # Set the number of bins
10 hist, xedges, yedges = np.histogram2d(y_current, y_previous, bins=num_bins, density=T
11
12 # Step 3: Assign probabilities to each data point x(k) = [y(k), y(k-1)]
13 # Find the bin indices for each (y(k), y(k-1)) pair
14 bin_x_indices = np.digitize(y_current, xedges[:-1]) - 1  # for y(k)
15 bin_y_indices = np.digitize(y_previous, yedges[:-1]) - 1  # for y(k-1)
16
17 # Correct out-of-bound indices
18 bin_x_indices[bin_x_indices >= num_bins] = num_bins - 1
19 bin_y_indices[bin_y_indices >= num_bins] = num_bins - 1
20
21 # Get the probability for each x(k) = [y(k), y(k-1)] based on the corresponding bin
22 prob_xk = hist[bin_x_indices, bin_y_indices]
23
24 # Step 4: Plot p([y(k), y(k-1)]) for each k (time index)
25 plt.figure(figsize=(20, 10))
26 plt.plot(range(1, N), prob_xk, color='blue', alpha=0.75)
27 plt.title('Probability p([y(k), y(k-1)]) for each time index k')
28 plt.xlabel('Time index k')
29 plt.ylabel('Probability p([y(k), y(k-1)])')
30 plt.grid(True)
31
32 plt.show()
```