COURSEWORK 3

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Stochastic Simulation

Author:
Richard Fu (CID: 01852979)

Date: December 13, 2022

Q1: SAMPLING AN INTERESTING CHAIN

Below is the code used to generate the plot in Figure 1.

Code Listing 1: Code for question 1

```
import numpy as np
import matplotlib.pyplot as plt

N = 10000
x0 = np.array([0,0])
w = np.array([0.2993, 0.7007])
A1 = np.array([[0.4, -0.3733], [0.06, 0.6]])
A2 = np.array([[-0.8, -0.1867], [0.1371, 0.8]])
b1 = np.array([0.3533, 0.0])
b2 = np.array([1.1, 0.1])

def f1(x):
    return A1 @ x + b1

def f2(x):
    return A2 @ x + b2

def sample_discrete(w):
    cw = np.cumsum(w) # cdf of weights
    u = np.random.uniform(0, 1) # random uniform
    for k in range(len(cw)): # samples the index from the discrete
                                   distribution
        if u <= cw[k]:
            return k

x_array = np.array([x0])
x = x0
for n in range(N):
    i_n = sample_discrete(w)
    if i_n == 0:
        x = f1(x)
    else:
        x = f2(x)
    x_array = np.vstack((x_array, x))

plt.figure(figsize=(10,7))
plt.scatter(x_array[20:, 0], x_array[20:, 1], s=0.1, color=[0.8, 0, 0
                                   ])
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.gca().spines['bottom'].set_visible(False)
plt.gca().spines['left'].set_visible(False)
plt.gca().set_xticks([])
plt.gca().set_yticks([])
plt.gca().set_xlim(0, 1.05)
plt.gca().set_ylim(0, 1)
plt.show()
```

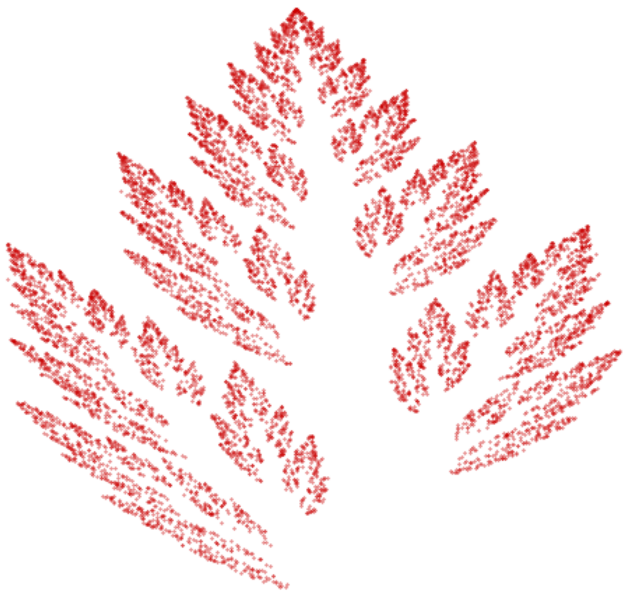

Figure 1

Q2: SAMPLE A STATE-SPACE MODEL

Part a: Simulate a Gaussian time-series corrupted by noise

Below is the code used to generate the plot in Figure 2.

Code Listing 2: Code for question 2a

```python
import numpy as np
import matplotlib.pyplot as plt

x0 = 1
a = 0.9
sigma_x = 0.01
sigma_y = 0.1
T = 100
x_vals = np.array([x0])
y_vals = np.array([])
x = x0
for t in range(T):
    x = np.random.normal(a*x, sigma_x)
    y = np.random.normal(x, sigma_y)
    x_vals = np.append(x_vals, x)
    y_vals = np.append(y_vals, y)

plt.figure(figsize=(10,7))
plt.plot(x_vals, "k-", label="x")
plt.plot(np.arange(1, T+1), y_vals, "r-", label="y")
plt.xlabel("time")
plt.legend()
plt.show()
```

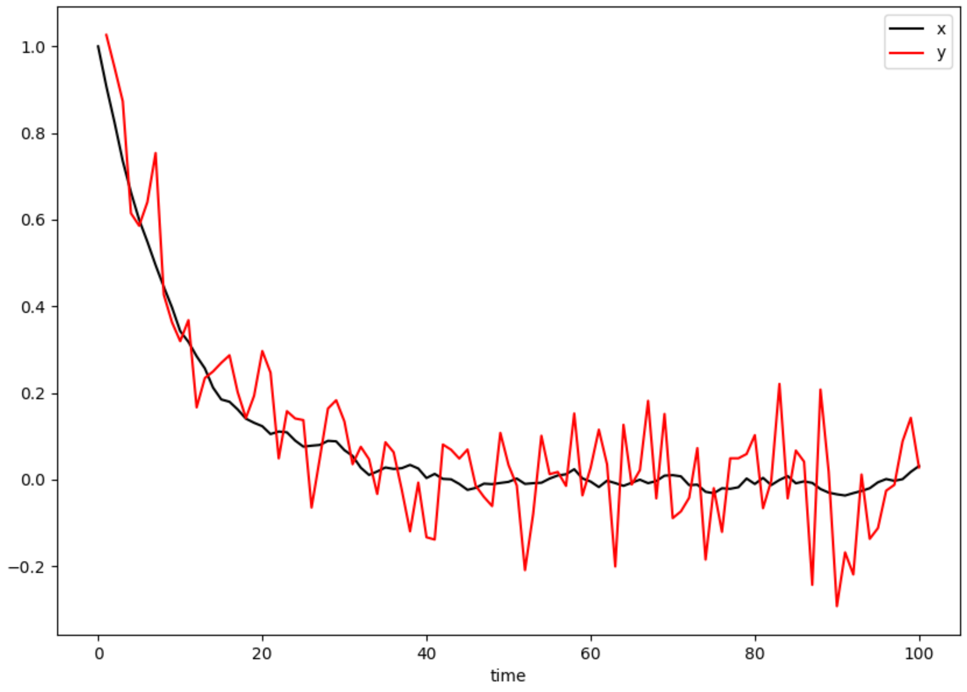

Figure 2

This model output could represent true wind velocity as x_t which is decreasing, and our observed/measured wind velocity with noise as y_t.
It could also represent a true temperature of a fluid as x_t which is decreasing, and our noisy observed temperature as y_t.

Part b: Develop a stochastic volatility model and simulate

Define our model as:

$$h_t | x_{t-1} \sim \mathcal{N}(h_t;\ ax_{t-1},\ \sigma_x^2), \quad (1)$$

$$x_t | h_t = |h_t|, \quad (2)$$

$$y_t | x_t \sim \mathcal{N}(y_t;\ 0,\ x_t^2). \quad (3)$$

Initial volatility x_0 can be taken as any non-negative value for ease. In this model, our x_t is the volatility of the stock, and y_t is the returns from the observed stock. From (1) and (2) it follows that we have a folded normal Markov transition kernel:

$$x_t | x_{t-1} \sim \begin{cases} \frac{1}{\sqrt{2\pi\sigma_x^2}} \exp\left(-\frac{(x_t - ax_{t-1})^2}{2\sigma_x^2}\right) + \frac{1}{\sqrt{2\pi\sigma_x^2}} \exp\left(-\frac{(x_t + ax_{t-1})^2}{2\sigma_x^2}\right) & \text{for } x_t \geq 0 \\ 0 & \text{otherwise.} \end{cases} \quad (4)$$

The parameter a is a centering factor of our volatility x_t given x_{t-1}.
$a < 1$ causes overall decay in volatlity, $a > 1$ causes overall growth in volatility, and

$a = 1$ gives more random volatility behaviour.
σ_x represents the standard deviation in our random volatility. Higher σ_x leads to more varied volatility through time.

To obtain x as a decaying system we can take $a = 0.99,\ x_0 = 10,\ \sigma_x = 0.01$.
To obtain x as a growing system we can take $a = 1.01,\ x_0 = 0.1,\ \sigma_x = 0.01$.
These situations can be seen in Figure 3.

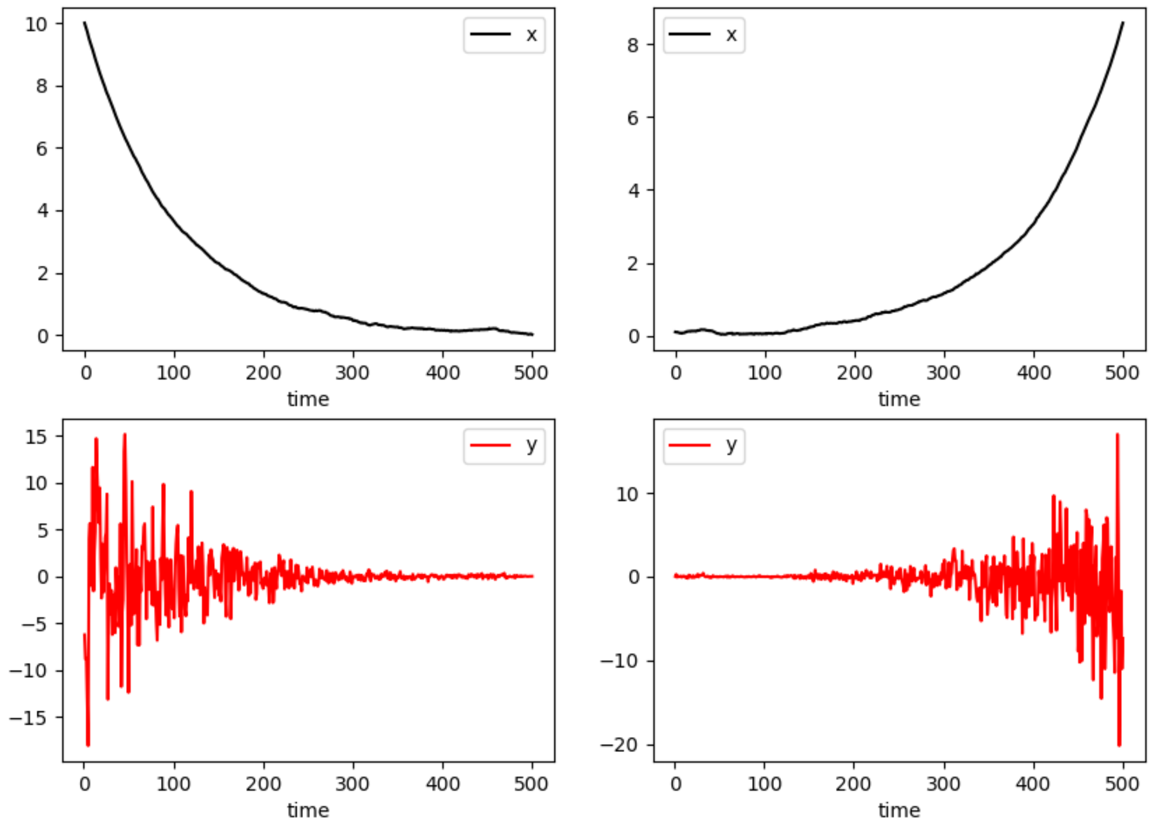

Figure 3

We can clearly see that as volatility decreases, so do the fluctuations in our returns. Similarly, as volatility increases, our returns fluctuate more. This is the behaviour we would want the model to exhibit.

To get some nice data for x and y, we can take $a = 1,\ x_0 = 10,\ \sigma_x = 1$.
These parameter choices exhibit more realistic and meaningful behaviour, as the volatility displays a lot more randomness as it likely would in practice.
Once again, we see that in the time portions of higher volatility, we see larger variations in our returns, and lower variations at the time portions of lower volatility.
Some examples are shown in Figure 4, 5, 6.

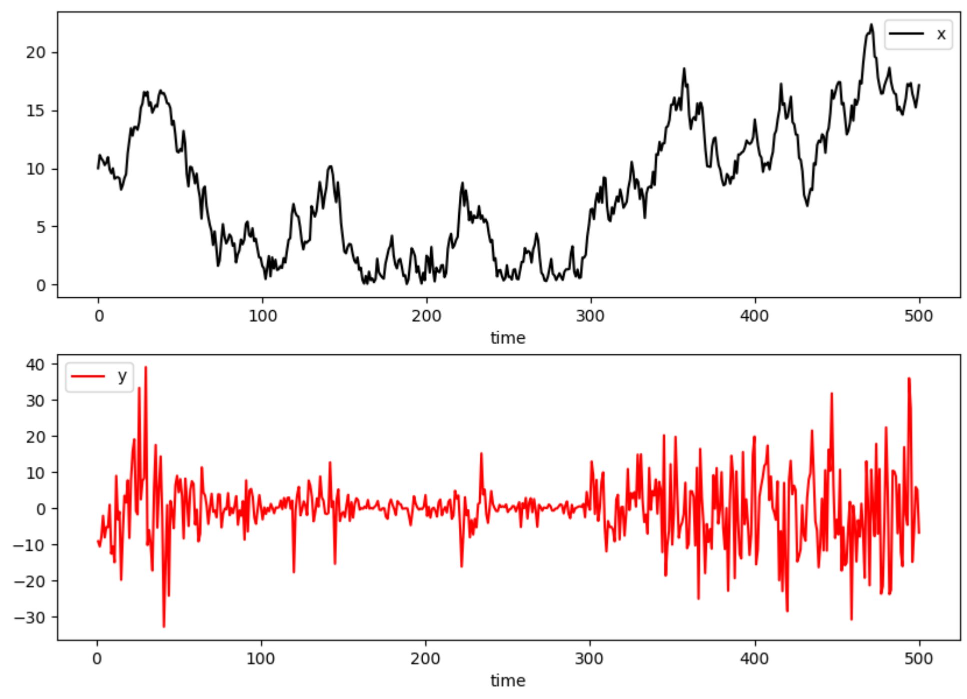

Figure 4

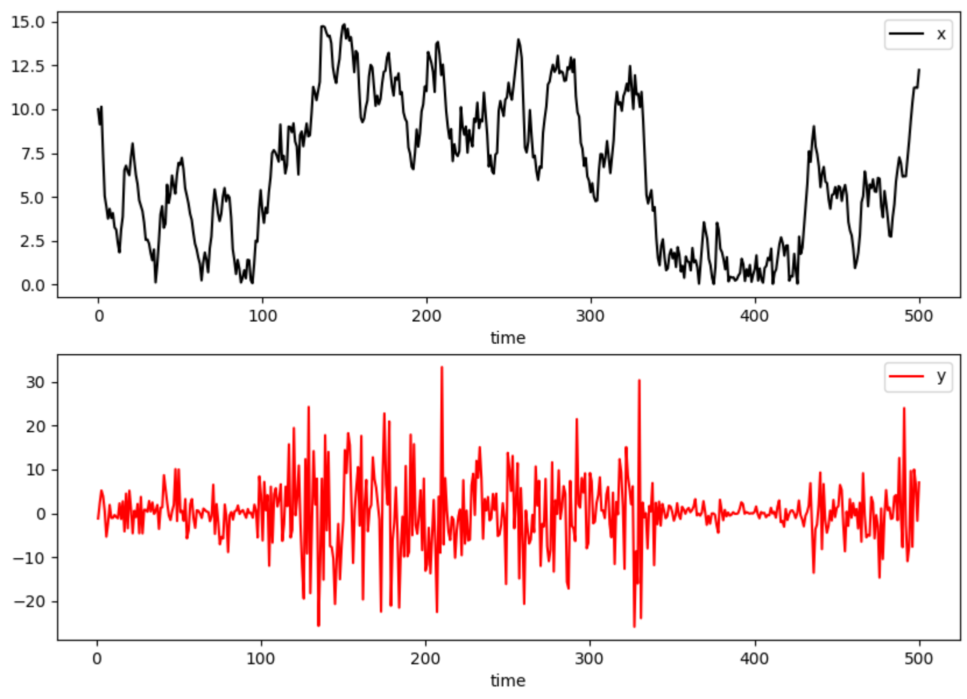

Figure 5

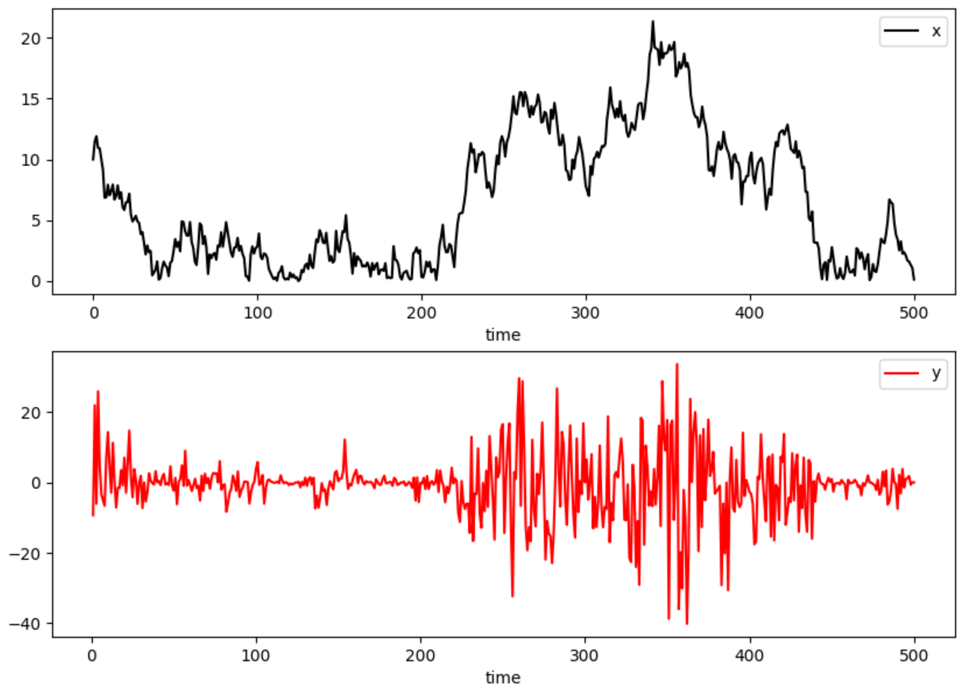

Figure 6

Below is the code that generated the above plots.

Code Listing 3: Code for question 2b

```
T = 500

### simulate data for x decaying
x0 = 10
a = 0.99
sigma_x = 0.01
x_decay = np.array([[x0]])
y_decay = np.array([])
x = x0
for t in range(T):
    x = np.abs(np.random.normal(a*x, sigma_x))
    y = np.random.normal(0, x)
    x_decay = np.append(x_decay, x)
    y_decay = np.append(y_decay, y)

### simulate data for x growing
x0 = 0.1
a = 1.01
sigma_x = 0.01
x_grow = np.array([[x0]])
y_grow = np.array([])
x = x0
for t in range(T):
    x = np.abs(np.random.normal(a*x, sigma_x))
    y = np.random.normal(0, x)
    x_grow = np.append(x_grow, x)
    y_grow = np.append(y_grow, y)

### plot x decaying and growing and corresponding ys
fig, axs = plt.subplots(2, 2, figsize=(10,7))
axs[0, 0].plot(x_decay, "k-", label="x")
axs[1, 0].plot(np.arange(1, T+1), y_decay, "r-", label="y")
axs[0, 1].plot(x_grow, "k-", label="x")
axs[1, 1].plot(np.arange(1, T+1), y_grow, "r-", label="y")
for i in range(2):
    for j in range(2):
        axs[i, j].set_xlabel("time")
        axs[i, j].legend()

### simulate other data
x0 = 10
a = 1
sigma_x = 1
x_vals = np.array([x0])
y_vals = np.array([])
x = x0
for t in range(T):
    x = np.abs(np.random.normal(a*x, sigma_x))
    y = np.random.normal(0, x)
    x_vals = np.append(x_vals, x)
    y_vals = np.append(y_vals, y)

### plot simulation data
fig, axs = plt.subplots(2, 1, figsize=(10,7))
axs[0].plot(x_vals, "k-", label="x")
axs[1].plot(np.arange(1, T+1), y_vals, "r-", label="y")
axs[0].set_xlabel("time")
axs[1].set_xlabel("time")
axs[0].legend()
axs[1].legend()
plt.show()
```