stochastic sim coursework 3

tmp120 01854740

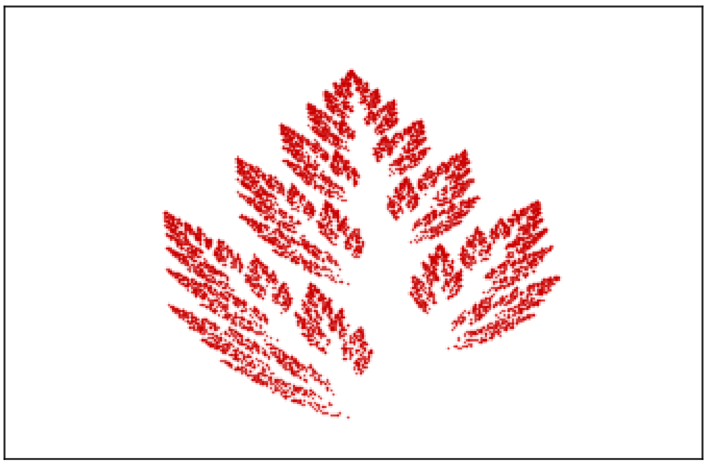
December 2022

1 sample an interesting chain

```
import numpy as np
import matplotlib.pyplot as plt
w1 = 0.2993
w2 = 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 np.matmul(A1,x) + b1
def f2(x):
    return np.matmul(A2,x) + b2
N = 10000
p = np.array([w1,w2])
s = np.array([1,2])
cdf = np.cumsum(p)
samples = []
for i in range(N):
    u = np.random.uniform(0, 1)
    for k in range(len(cdf)):
        if cdf[k] > u:
            samples.append(s[k])
            break
X  = np.zeros((len(samples)+1,2))
x0 = np.array([[0],[0]])
X[0][0]=x0[0]
X[0][1]=x0[1]
x =  x0
for n,i in enumerate(samples):
    #print(n)
    if i==1:
        x = f1(x)
    elif i==2:
        x=f2(x)
    X[n+1][0]=x[0]
    X[n+1][1]=x[1]
plt.scatter(X[20:,0], X[20:,1], s=0.1, color = [0.8, 0, 0])
plt.gca ().spines['top']. set_visible (True)
plt.gca ().spines['right']. set_visible (True)
plt.gca ().spines['bottom']. set_visible (True)
```

```
plt.gca ().spines['left']. set_visible (True)
plt.gca (). set_xticks ([])
plt.gca (). set_yticks ([])
plt.gca ().set_xlim(0, 1.05)
plt.gca ().set_ylim(0, 1)
plt.show ()
```

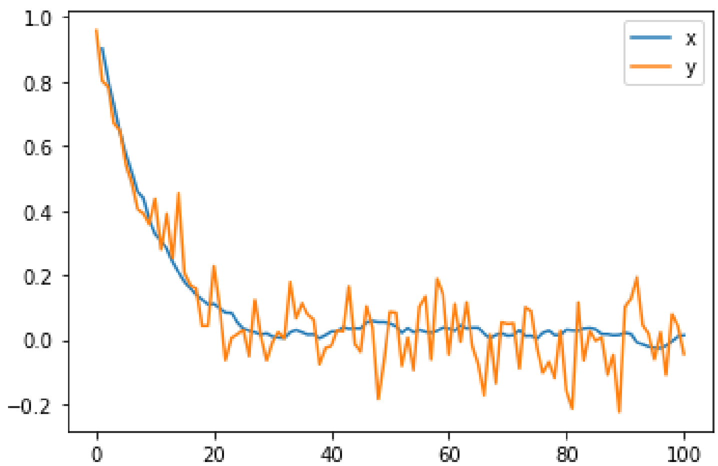
We get the following resulting scatter plot:



2 sample a state-space model

2.1 Simulate a Gaussian time-series corrupted by noise

```
x0 = 1
a = 0.9
sigma_x  = 0.01
sigma_y =  0.1
T = 100
X =  np.zeros(T+1)
Y = np.zeros(T+1)
x = x0
for t  in range(1,T+2):
    y = np.random.normal(x,sigma_y)
    Y[t-1]=y
    X[t-1]=x
    x = np.random.normal(a*x,sigma_x)
times = list(range(1,T+1))
plt.plot(times,X[1:],Y)
plt.legend(["x","y"])
```



This models an underlying x and hidden state space y where y is x corrupted by noise, and both slowly decay towards 0 and it is possible to go negative.
An example might be a reversible chemical reaction starting with all A in which the forward and backward rates gradually converge to equilibrium where the concentration of A - concentration of B is 0. x would represent the actual concentrations over time while y is the noisy measurements of concentration difference made by a scientist. Another example is of a metallic object put between 2 magnets so that it will stabilise at some at distance 0, the equilibrium in the middle. The metal starts more to one end. x is the actual position at time t of the metal while y is the inaccurately measured distance over time.

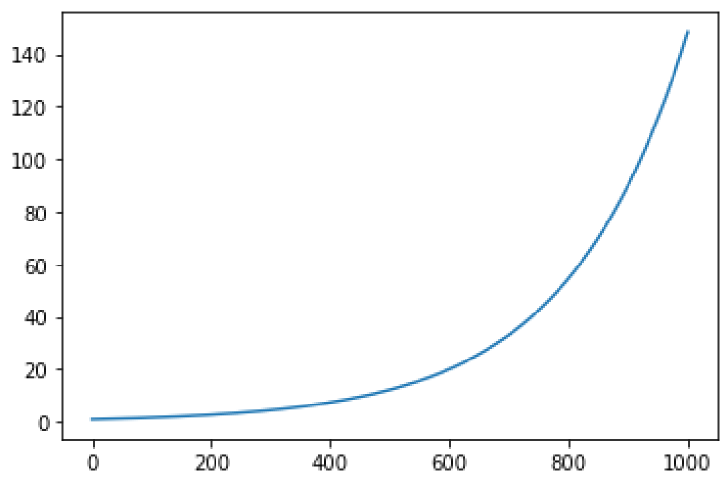
2.2 Develop a stochastic volatility model and simulate

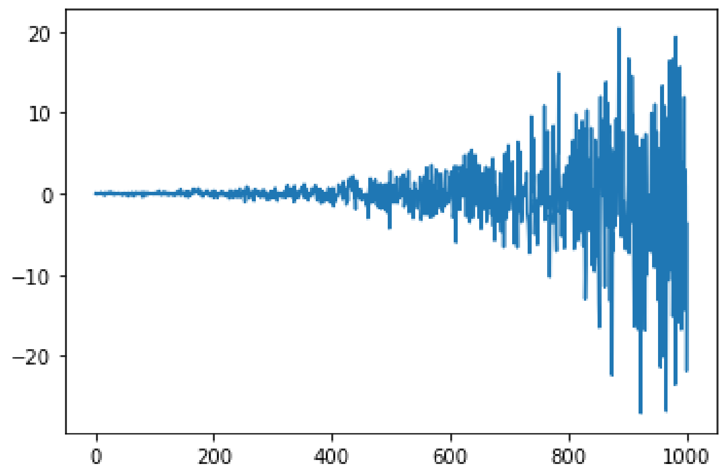
I define the Markov transition kernel as the exponential of:
$N(\alpha + \phi(\log(x_{t-1} - \alpha), \sigma_x)$. The exponential ensures the volatility does not go negative while the logarithm combined with the exponential maintains $x_t|x_{t-1}$ of order x_{t-1} and the σ_x allows for noise.
I define the likelihood as $N(\mu_y, x_t\sigma_y)$ so that the variance of the returns increases with the volatility at time t.

```
x0 = 1
a = 1
sigma_x  = 0.05
sigma_y =  0.1
miu_y=0
alpha = 2
phi = 1
T = 1000
#sanity check with x growing
X =  np.zeros(T+1)
Y = np.zeros(T+1)
x = x0
for t  in range(1,T+2):
```

```
    y = np.random.normal(miu_y,x*sigma_y)
    X[t-1]=x
    Y[t-1]=y
    x = np.exp(t*0.005)
times = list(range(T+1))
plt.plot(times,X)
plt.plot(times,Y)
#sanity check x decaying
X =  np.zeros(T+1)
Y = np.zeros(T+1)
x = x0
for t  in range(1,T+2):
    y = np.random.normal(miu_y,x*sigma_y)
    X[t-1]=x
    Y[t-1]=y
    x = np.exp(t*-0.005)
times = list(range(T+1))
plt.plot(times,X)
plt.plot(times,Y)
#real model
X =  np.zeros(T+1)
Y = np.zeros(T+1)
x = x0
for t  in range(1,T+2):
    y = np.random.normal(miu_y,x*sigma_y)
    X[t-1]=x
    Y[t-1]=y
    x = np.exp(np.random.normal(alpha+phi*(np.log(x)-alpha),sigma_x
                                    ))
times = list(range(T+1))
plt.plot(times,X)
plt.plot(times,Y)
```

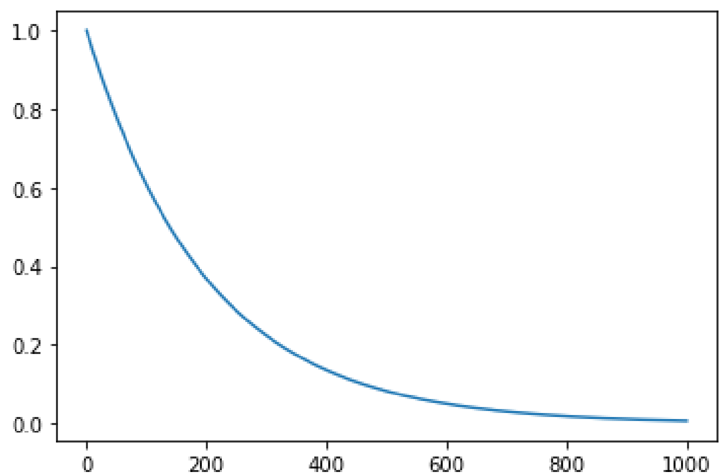
I plot the returns with increasing and decreasing volatility plots respectively and the variance increase with the volatility as expected.



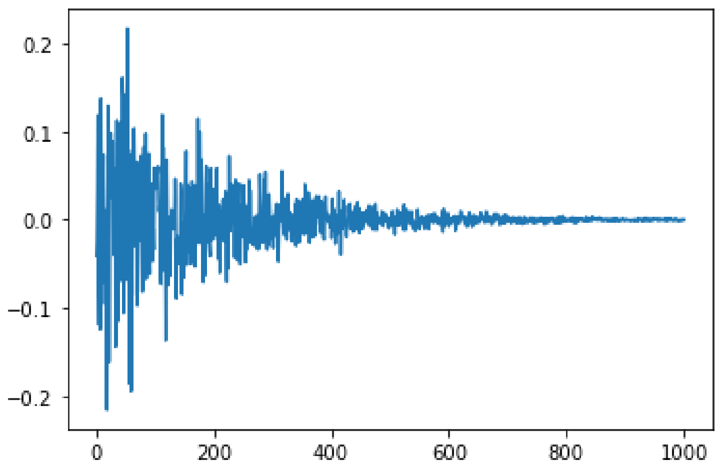


20
10
0
−10
−20
0
200
400
600
800
1000

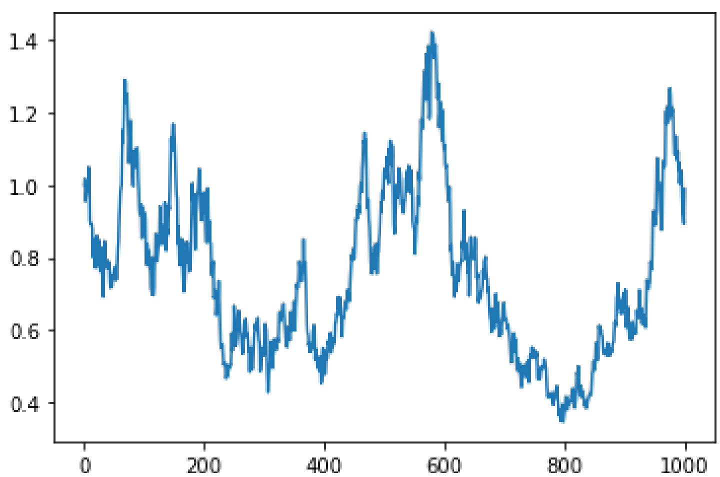
Decreasing:

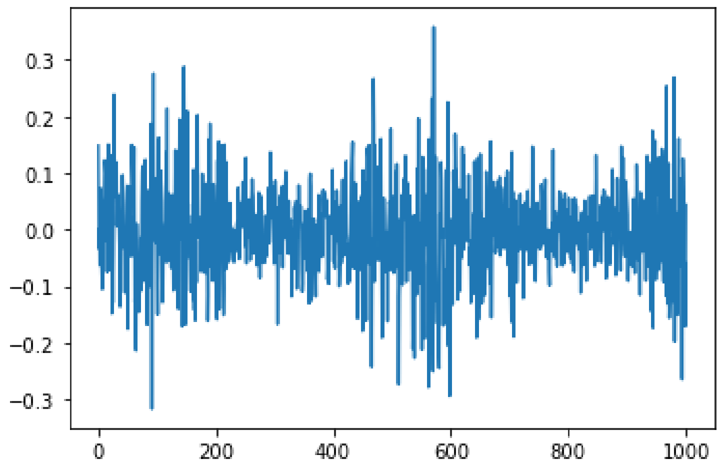


1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800
1000



Finally, I plot x and then y for the model.




0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0
200
400
600
800
1000