# Computational Finance

## Series 4 Redo ¶

Tientso Ning

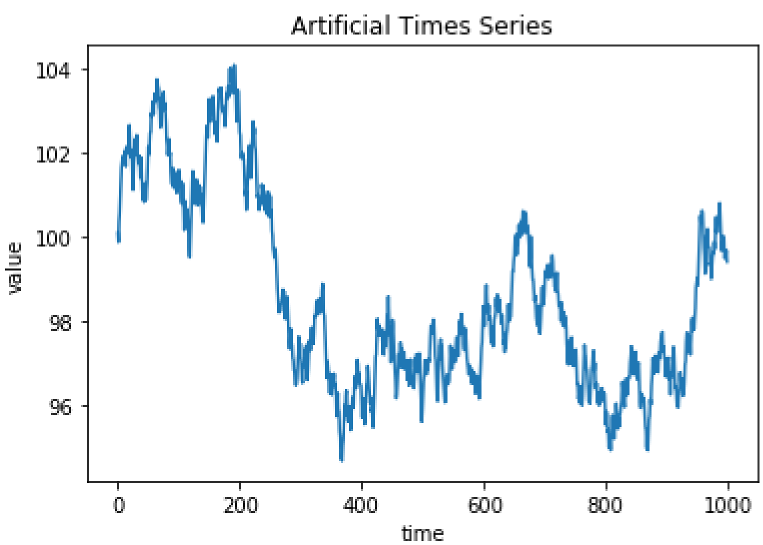
In [1]:
```
import numpy as np
import pandas as pd
import time
import matplotlib.pyplot as plt
```

### Time Average

In [41]:
```
#artificial time series
np.random.seed(1)
x = 100 + np.cumsum(0.5-np.random.random(1000))
```

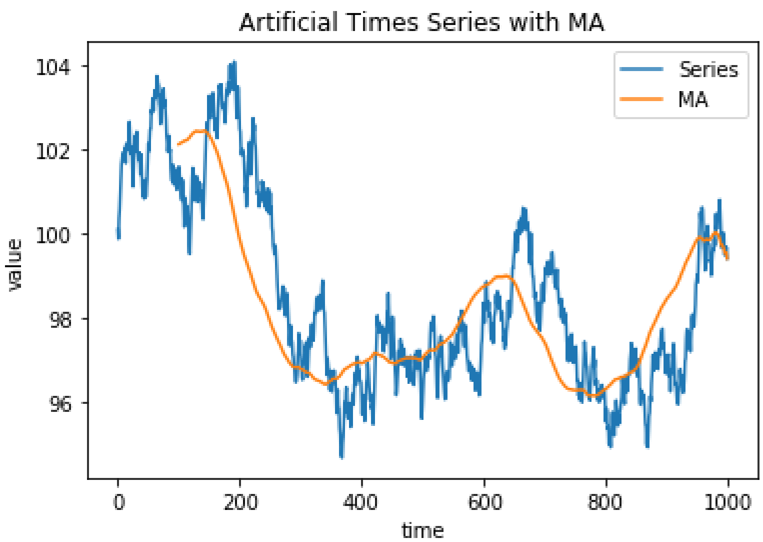
In [42]:
```
#visualize the time series
plt.plot(x)
plt.xlabel("time")
plt.ylabel("value")
plt.title("Artificial Times Series")
```

Out[42]:
```
Text(0.5, 1.0, 'Artificial Times Series')
```



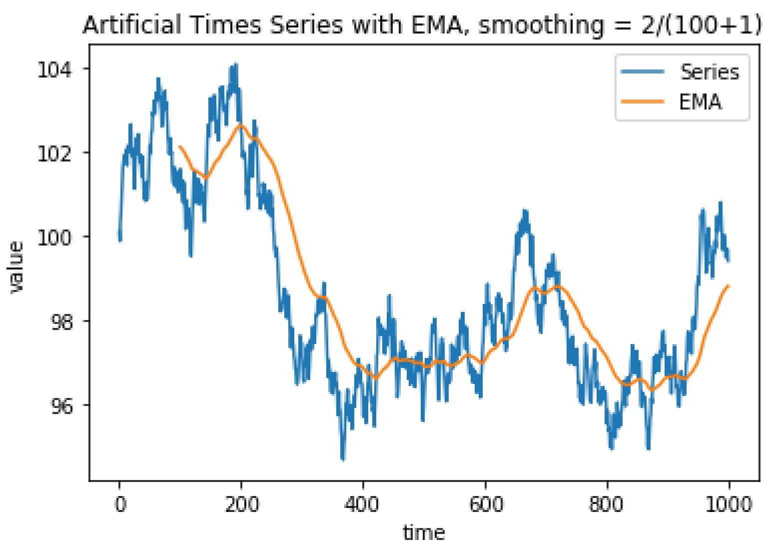
In [76]:
```
#Moving Average for N = 100
N = 100

def MA(ts, N):
    L = len(ts)
    assert L%N == 0 #we want to make sure even blocks
    ma = []

    for i in range(0,N): #account for necessary data
        ma.append(np.nan)

    for i in range(N,L): #for each data point starting from N
        ma.append(np.ma.average(ts[i:i+N]))

    return ma
```

In [77]:
```
plt.plot(x, label = "Series")
plt.xlabel("time")
plt.ylabel("value")
plt.title("Artificial Times Series with MA")
plt.plot(MA(x,N), label="MA")
plt.legend()
plt.show()
```



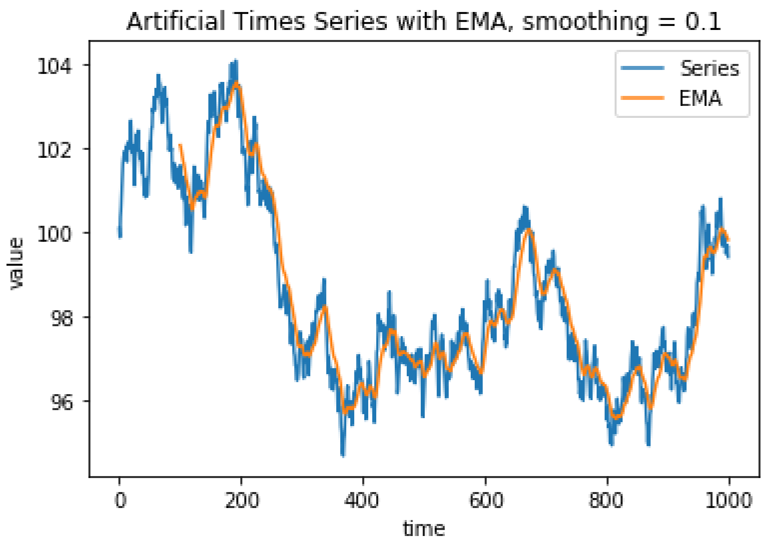
In [83]:
```
def EMA(ts, N, smoothing):
    L = len(ts)
    assert L%N == 0 #we want to make sure even blocks
    ema_init = np.ma.average(ts[N:N+N]) #init with SMA

    ema = []
    for i in range(0,N): #lag
        ema.append(np.nan)

    ema_pre = ema_init #first run
    for i in range(N, L):
        ema_curr = ts[i]*(smoothing) + ema_pre*(1-smoothing)
        ema.append(ema_curr)
        ema_pre = ema_curr

    return ema
```

In [85]:
```
plt.plot(x, label = "Series")
plt.xlabel("time")
plt.ylabel("value")
plt.title("Artificial Times Series with EMA, smoothing = 2/(100+1)")
plt.plot(EMA(x,N,2/(100+1)), label="EMA")
plt.legend()
plt.show()
```



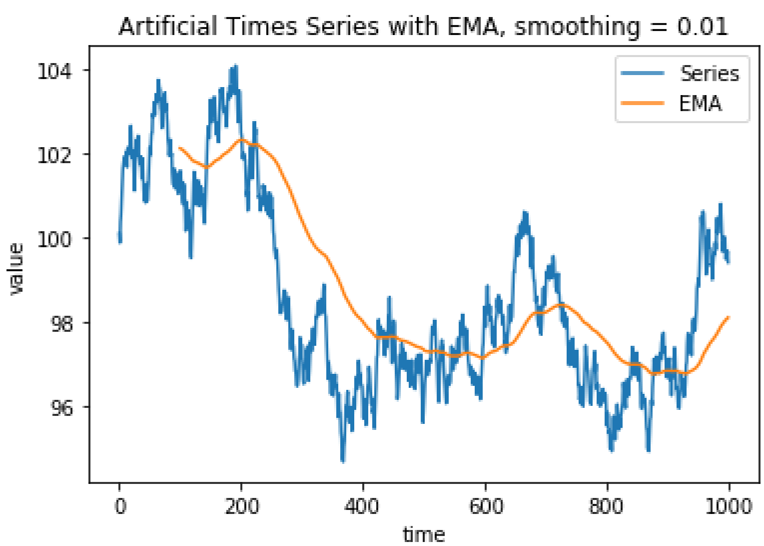
In [87]:

```
plt.plot(x, label = "Series")
plt.xlabel("time")
plt.ylabel("value")
plt.title("Artificial Times Series with EMA, smoothing = 0.1")
plt.plot(EMA(x,N,0.1), label="EMA")
plt.legend()
plt.show()
```




In [88]:

```
plt.plot(x, label = "Series")
plt.xlabel("time")
plt.ylabel("value")
plt.title("Artificial Times Series with EMA, smoothing = 0.01")
plt.plot(EMA(x,N,0.01), label="EMA")
plt.legend()
plt.show()
```



In the case of smoothing `smoothing = 0.1` and `smoothing = 0.01` , we can see that a smaller value (meaning more smoothing) means that prices are less affected by sudden shifts compared to the higher value.0.019801980198019802 For the case of 2/(N+1), we can see that `2/(N+1) = 0.019801980198019802` and so we should get a smoothing somewhere between 0.1 and 0.01, and that reflects the shape of the data somewhat.

## Scaling Law

In [105]:
```
def CDC (ts, delta):

    L = len(ts)
    direction = -1
    x = 0
    dc = 1

    for i in range (1, L): #cycle through the ts

        if np.abs(ts[x]-ts[i]) > delta*ts[x]: #scaling law

            x = i #set new

            #calculate directional change/overshoot
            if np.sign(ts[x]-ts[i]) != np.sign(direction):
                direction = direction * -1 #change direction
                dc +=1
            else:
                #overshoot
                continue


    return dc
```

In [118]:
```
CDC(x, 0.01)
```

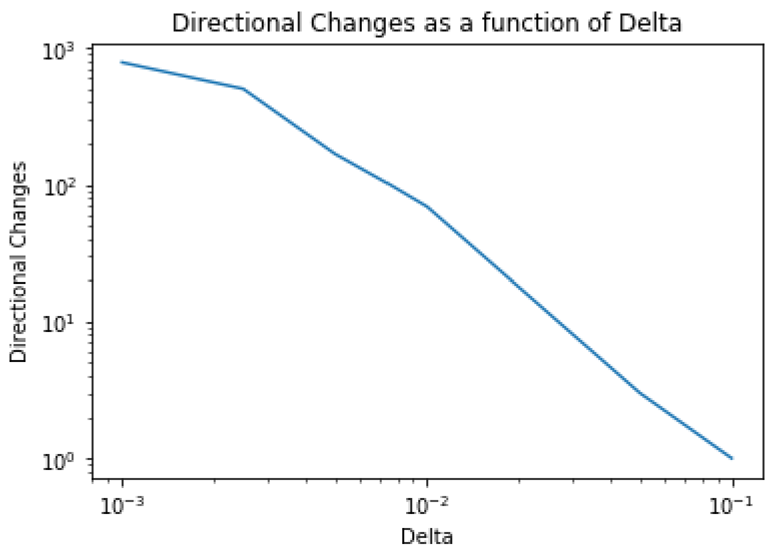
Out[118]: 69

In [119]:
```
deltas = [0.1, 0.05, 0.01, 0.0075, 0.005, 0.0025, 0.001]
dcs = []
for i in range(0, len(deltas)):
    dcs.append(CDC(x,deltas[i]))
print(dcs)
```

```
[1, 3, 69, 101, 168, 503, 786]
```

In [116]:
```
plt.loglog(deltas,dcs)
plt.title("Directional Changes as a function of Delta")
plt.xlabel("Delta")
plt.ylabel("Directional Changes")
```

Out[116]: `Text(0, 0.5, 'Directional Changes')`




Here we observe that the number of directional changes decreases as the choice for delta gets bigger, which makes sense since the bigger the delta value, the bigger change needs to occur in the prices.