Simulating a Markov Chain

Here we will simulate the simplest possible (2-state) Markov chain

In [1]:
```
import matplotlib.pylab as plt   # That gives plotting, and the next line makes plots appear inl
%matplotlib inline
import numpy as np  # That gives numerical arrays and tools for manipulating them
import scipy.optimize as opt
import scipy.linalg as la

from scipy.optimize import curve_fit
```

Initialize random number generator

In [2]:
```
rng = np.random.default_rng()
```

Define transition matrix

In [3]:
```
A = np.array([[0.98, 0.10, 0],
              [0.02, 0.70, 0.05],
              [0, 0.20, 0.95]])
print(A)
```

```
[[0.98 0.1  0.  ]
 [0.02 0.7  0.05]
 [0.   0.2  0.95]]
```

Set up for simulating realization of the Markov Chain

Define a list of states on this realization. states(k)=1 means in state 1 at timestep k, etc

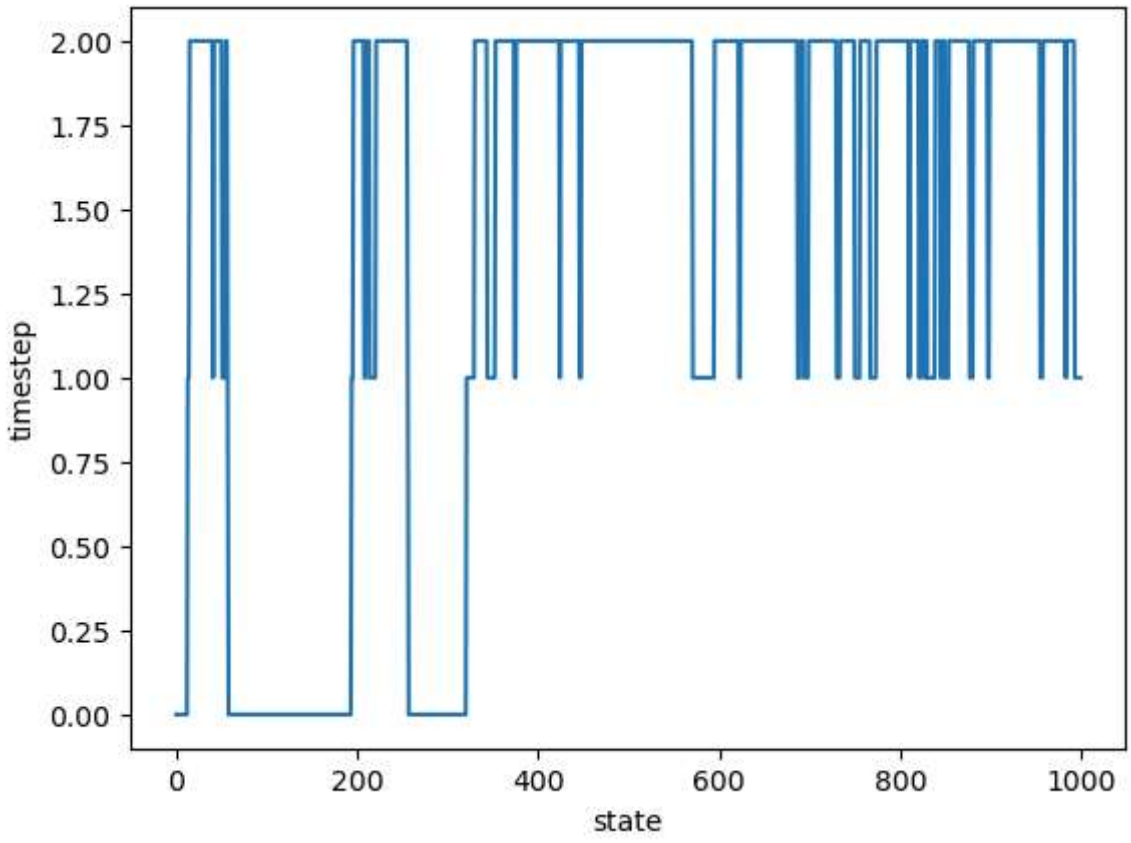
In [4]:
```
#number of timesteps
Tmax=1000
#intialize array of states; need to be integers as will use as indices below
states=np.zeros(Tmax,dtype=int)
#set initial state
states[0]=0
```

In [5]:
```
for t in np.arange(Tmax - 1):
    r = rng.uniform(0, 1) # draw random variable (uniformly distributed in 0,1)
    current_state = states[t]

    if states[t] == 0: # for transition FROM states[t] to state 0
        if r < A[0, 0]:
            states[t + 1] = 0
        else:
             states[t + 1] = 1

    elif states[t] == 1:
        if r < A[1, 1]: #transition 1 to 1.
            states[t + 1] = 1
        elif A[1, 1] < r < A[1, 1] + A[1, 0]: #transition 1 to 0.
            states[t + 1] = 0
        elif A[1, 1] + A[1, 0] < r: #transition 1 to 2
            states[t + 1] = 2
```

```
    elif states[t] == 2:
        if r < A[2, 2]:
            states[t + 1] = 2
        else:
            states[t + 1] = 1
```

In [6]:
```
#Plot of states
plt.plot(states)
plt.xlabel('state')
plt.ylabel('timestep')
```

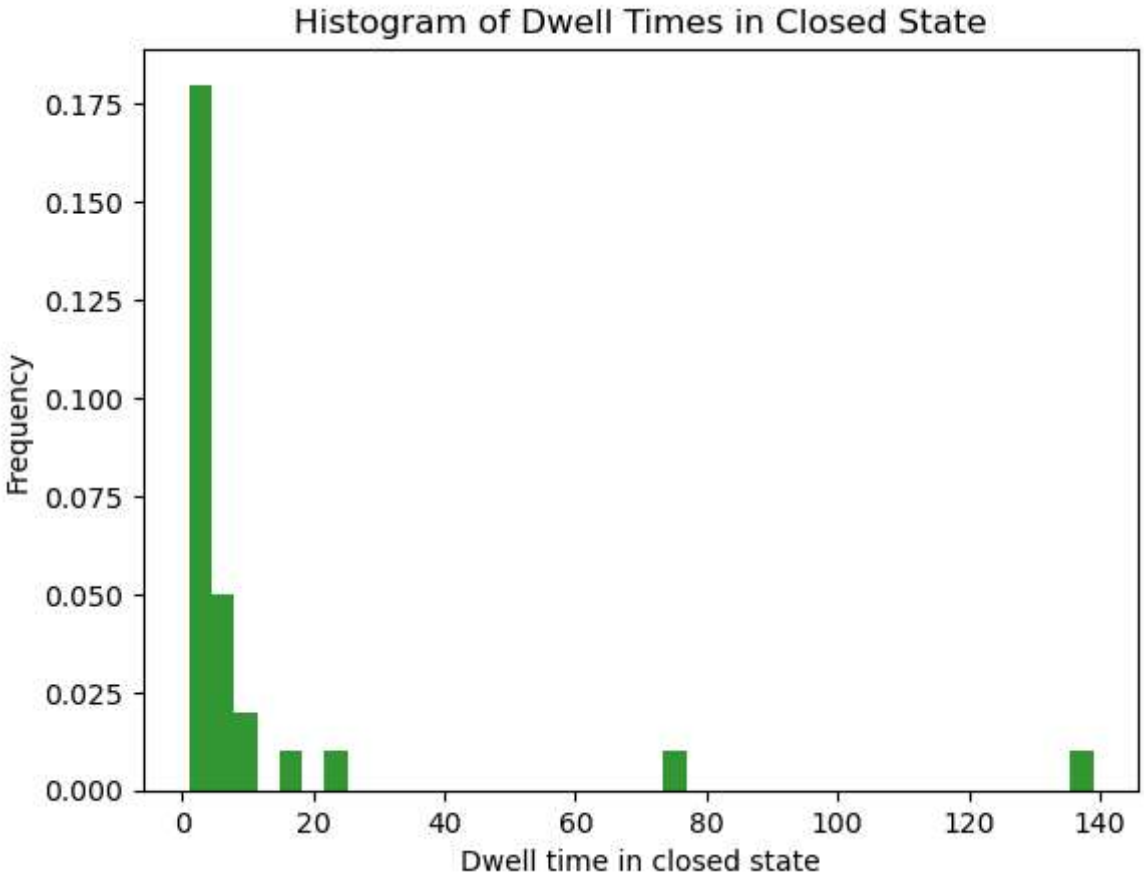
Out[6]: Text(0, 0.5, 'timestep')



In [7]:
```
# Reduce closed states rstates of 0 and 1
rstates = (states == 0) | (states == 1)
# print(states)
# print(rstates)

# Compute dwell times in the closed state
dwell_times = []
current_dwell = 0
for i in range(Tmax):
    if rstates[i]:
        current_dwell += 1
    else:
        if current_dwell > 0:
            dwell_times.append(current_dwell)
            current_dwell = 0
# Add ongoing dwell at the end
if current_dwell > 0:
    dwell_times.append(current_dwell)

print(dwell_times)
```

```
[15, 1, 4, 139, 2, 7, 74, 9, 2, 1, 2, 24, 2, 3, 5, 4, 6, 7, 1, 2, 2, 9, 2, 4, 4, 2, 3, 1, 7]
```

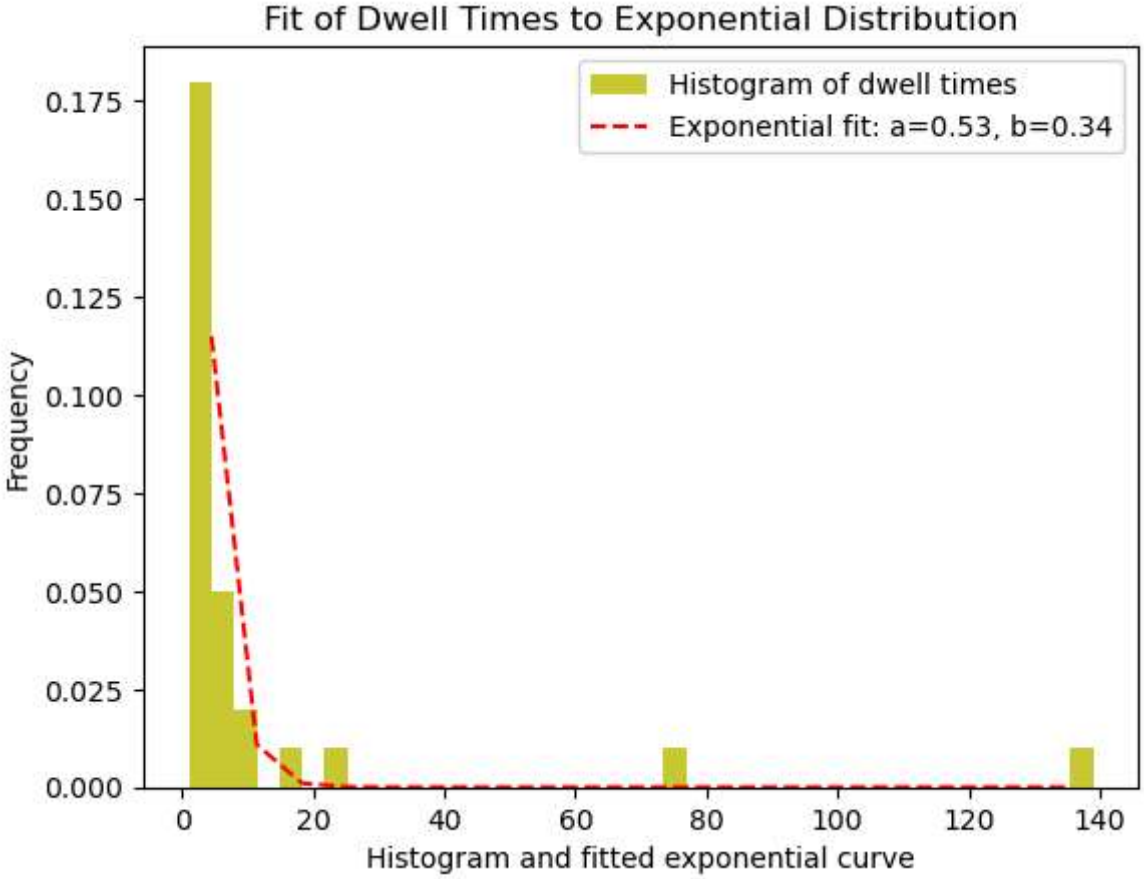
In [8]:
```
# Histogram of dwell times
plt.hist(dwell_times, bins=40, density=True, alpha=0.8, color='g')
plt.xlabel('Dwell time in closed state')
plt.ylabel('Frequency')
plt.title('Histogram of Dwell Times in Closed State')
plt.show()
```



In [9]:
```
# Exponential distribution function
def exp_func(x, a, b):
    return a * np.exp(-b * x)

# Generate histogram data for fitting
hist_data, bin_edges = np.histogram(dwell_times, bins=20, density=True)
bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2
popt, pcov = curve_fit(exp_func, bin_centers, hist_data) # curve fit
```

```
C:\Users\julie\AppData\Local\Temp\ipykernel_40864\4236958410.py:3: RuntimeWarning: overflow encou
ntered in exp
  return a * np.exp(-b * x)
```

In [10]:
```
# Plot of histogram and fitted exponential curve
plt.hist(dwell_times, bins=40, density=True, alpha=0.8, color='y', label='Histogram of dwell tim
plt.plot(bin_centers, exp_func(bin_centers, *popt), 'r--', label=f'Exponential fit: a={popt[0]:.
plt.xlabel('Histogram and fitted exponential curve')
plt.ylabel('Frequency')
plt.title('Fit of Dwell Times to Exponential Distribution')
plt.legend()
plt.show()
```


Fit of Dwell Times to Exponential Distribution
Histogram of dwell times
Exponential fit: a=0.53, b=0.34
Frequency
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
20
40
60
80
100
120
140
Histogram and fitted exponential curve