Bayesian Inference

OCNC 2019

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Coin toss

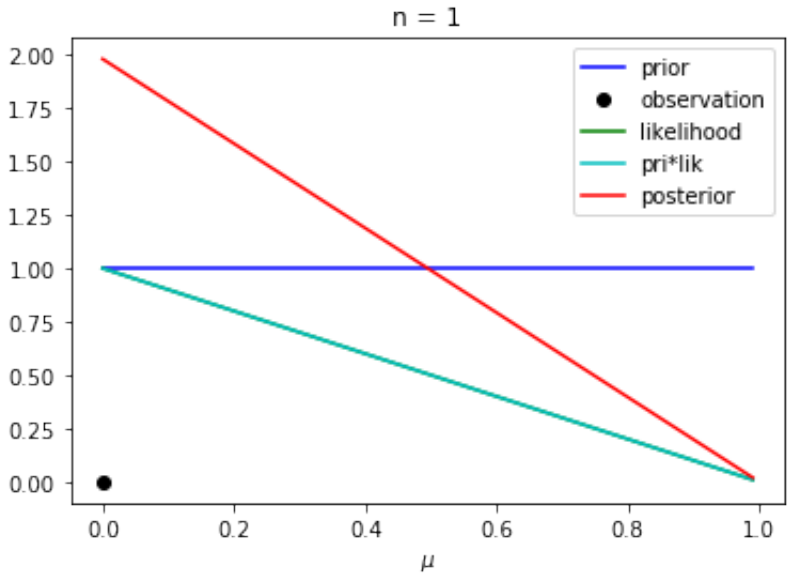
Estimate the parameter μ, probability for a coin to land head up, after n tosses.

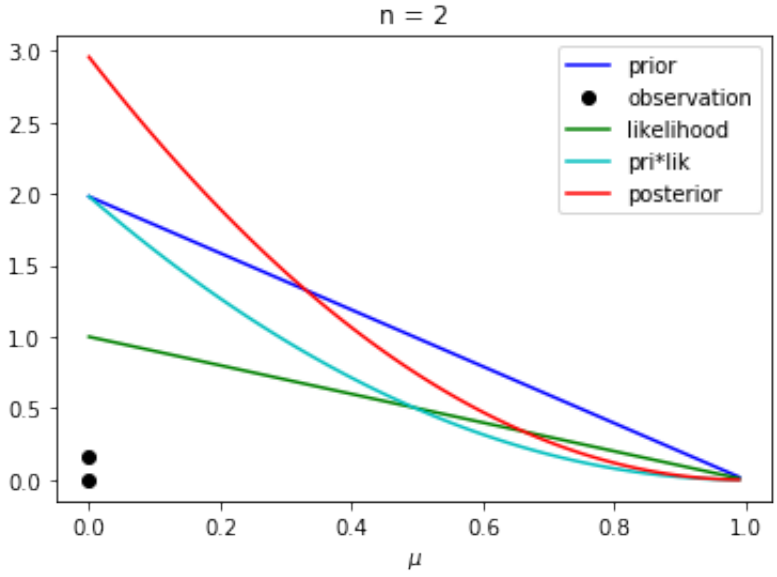
In [2]:
```
mu = 0.3  # probability of head
N = 6   # number of samples
y = np.random.choice(2, N, p=[1-mu, mu]) # binary observation sequ
y
```

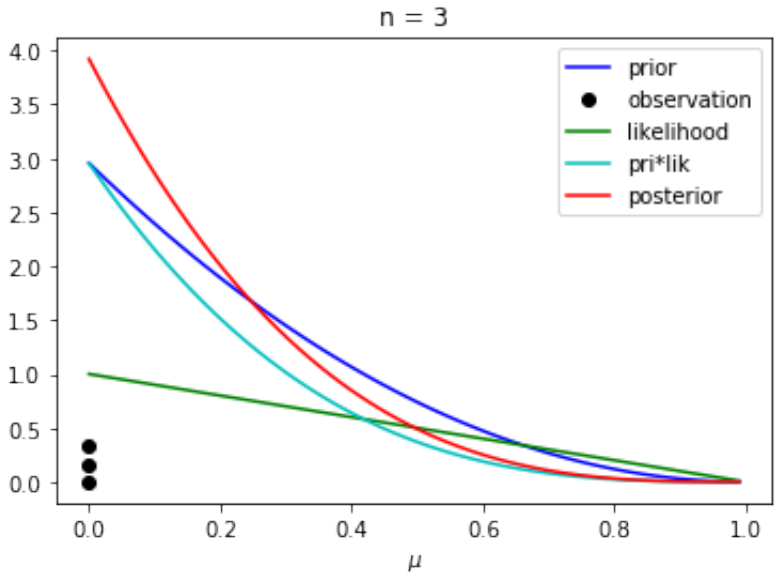
Out[2]:
```
array([0, 0, 0, 0, 0, 0])
```

Assume a flat prior $P(\mu) = 1$ for $0 \leq \mu \leq 1$.
Multiply prior and likelihood, and normalize to make the posterior.
Make posterior as the new prior.

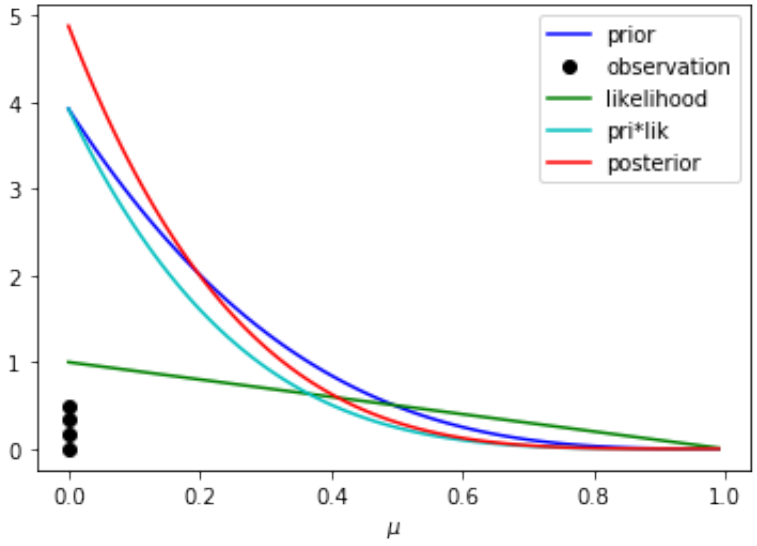
In [3]:
```
dx = 0.01  # plot step
x = np.arange( 0, 1, dx)  # range of the parameter
prior = np.ones( len(x))  # uniform prior
for n in range(N):
    plt.figure()  # a new figure
    # prior
    plt.plot( x, prior, 'b')
    # observation
    plt.plot( y[0:n+1], np.arange(n+1)/N, 'ko')
    # likelihood
    likelihood = x*y[n] + (1-x)*(1-y[n]) # theta if head, 1-theta
    plt.plot( x, likelihood, 'g')
    # posterior
    prilik = prior*likelihood
    plt.plot( x, prilik,'c')
    marginal = sum(prilik)*dx    # integrate over the parameter ra
    posterior = prilik/marginal  # normalize
    plt.plot( x, posterior, 'r')
    plt.xlabel('$\mu$')
    plt.legend(('prior', 'observation', 'likelihood', 'pri*lik', '
    plt.title('n = {0}'.format(n+1))
    plt.show()
    prior = posterior  # new prior
```


n = 1
prior
observation
likelihood
pri*lik
posterior
μ

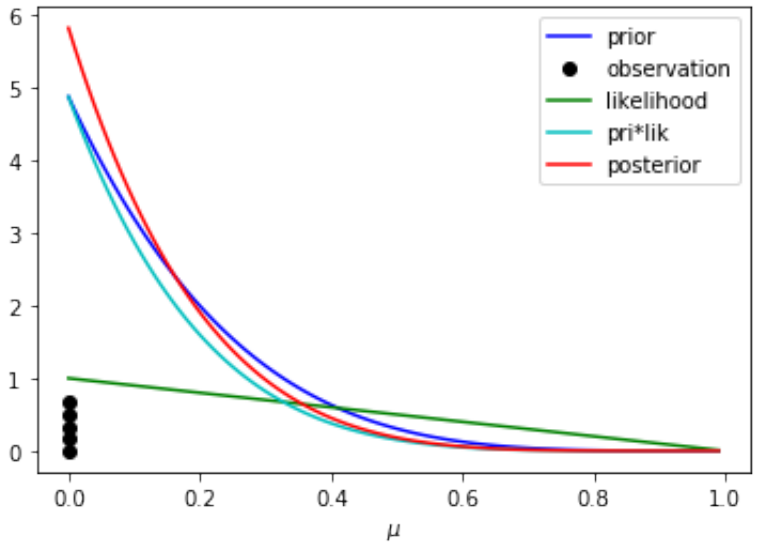

n = 2
prior
observation
likelihood
pri*lik
posterior
μ


n = 3
prior
observation
likelihood
pri*lik
posterior
μ

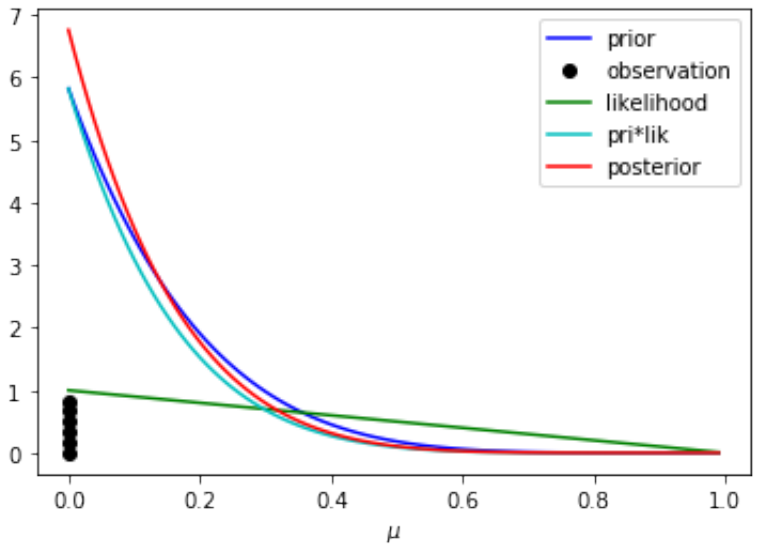
n = 4


prior
observation
likelihood
pri*lik
posterior
μ

n = 5


prior
observation
likelihood
pri*lik
posterior
μ

n = 6


prior
observation
likelihood
pri*lik
posterior
μ

In [ ]:
```
1
```

Noisy observation

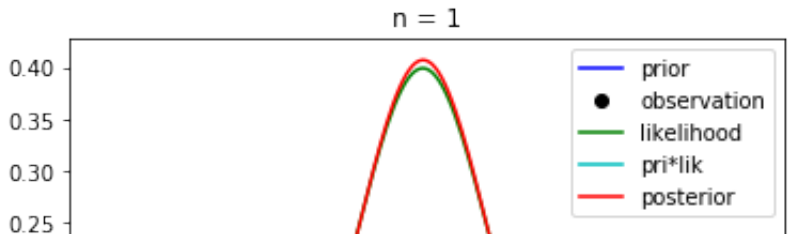
Estimate the mean μ from noisy observations.

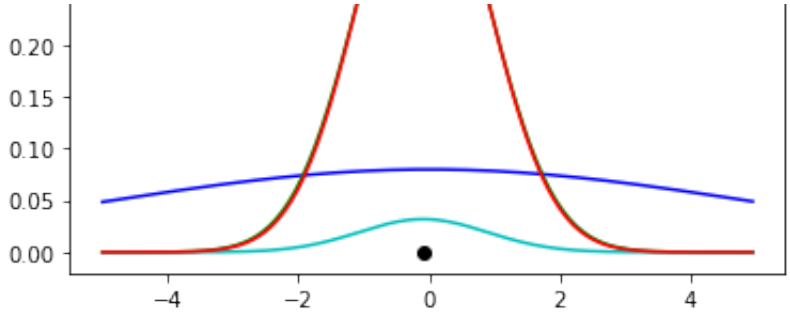
In [4]:
```
# Noisy observation: y = N(mu,sigma)
mu = 2
sigma = 1
N = 6
y = mu + sigma*np.random.randn(N)
y
```

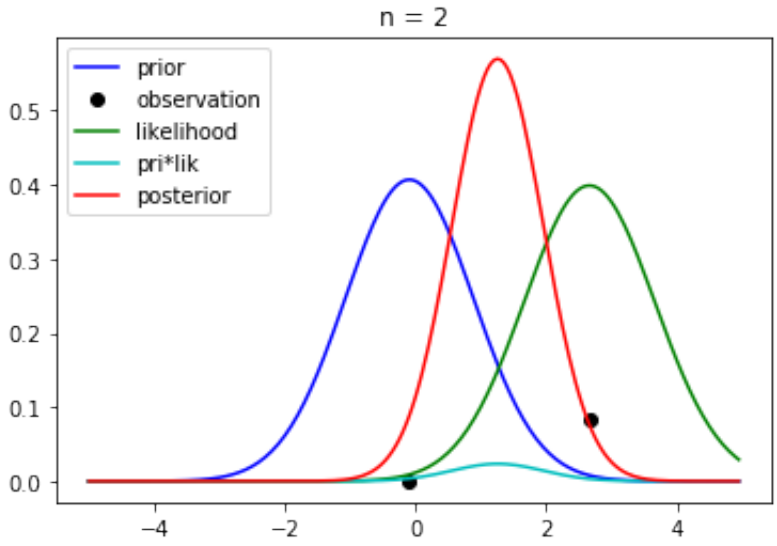
Out[4]:
```
array([-0.0966059 ,  2.66229795,  2.49849008,  2.81790195,  0.753068
86,
        1.38832428])
```

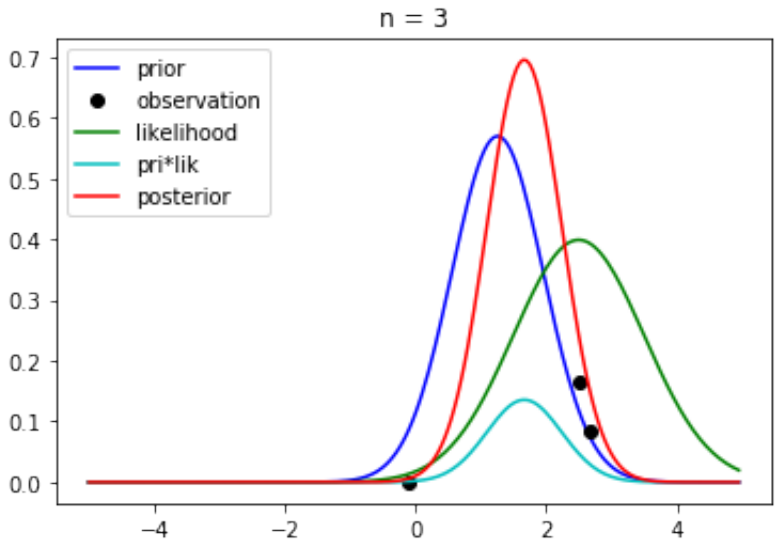
For simplicity, assume that we know the noise level σ.
Assume a wide Gaussian prior $N(\mu_0, \sigma_0)$ for μ.

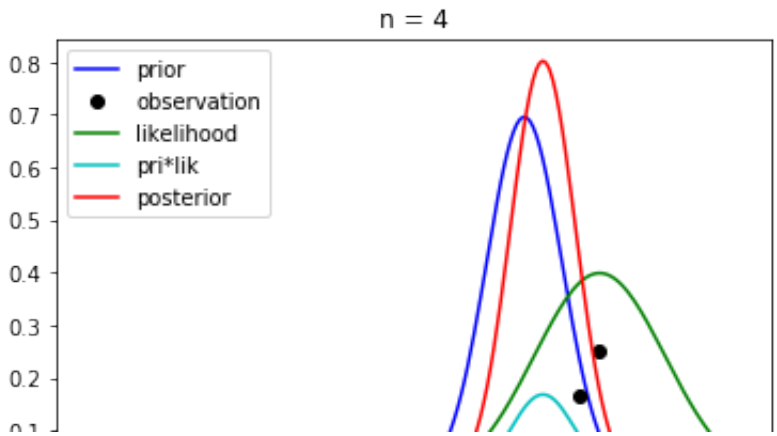
In [5]:
```
mu0 = 0
sigma0 = 5
dx = 0.05   # plot step
x = np.arange( -5, 5, dx)
# prior of the mean
prior = np.exp(-(x/sigma0)**2/2)/(np.sqrt(2*np.pi)*sigma0)
for n in range(N):
    plt.figure()
    # prior
    plt.plot( x, prior, 'b')
    # observation
    plt.plot( y[0:n+1], 0.5*np.arange(n+1)/N, 'ko')
    # likelihood
    likelihood = np.exp(-((y[n]-x)/sigma)**2/2)/(np.sqrt(2*np.pi)*s
    plt.plot( x, likelihood, 'g')
    # posterior
    prilik = prior*likelihood
    plt.plot( x, prilik,'c')
    marginal = sum(prilik)*dx
    posterior = prilik/marginal
    plt.plot( x, posterior, 'r')
    plt.legend(('prior', 'observation', 'likelihood', 'pri*lik', 'p
    plt.title('n = {0}'.format(n+1))
    prior = posterior  # new prior
```

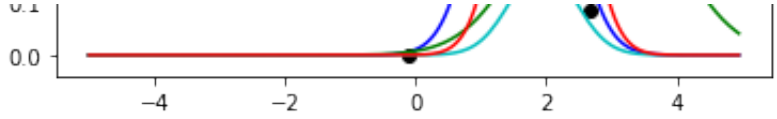


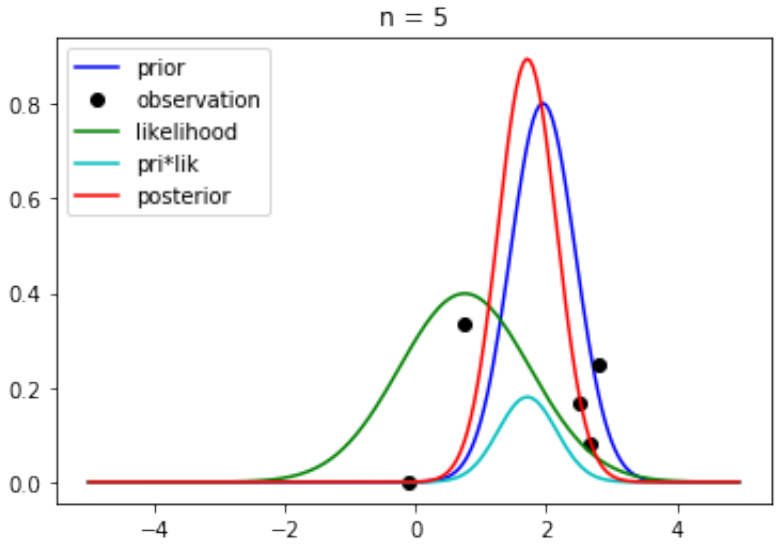

0.20
0.15
0.10
0.05
0.00
−4
−2
0
2
4

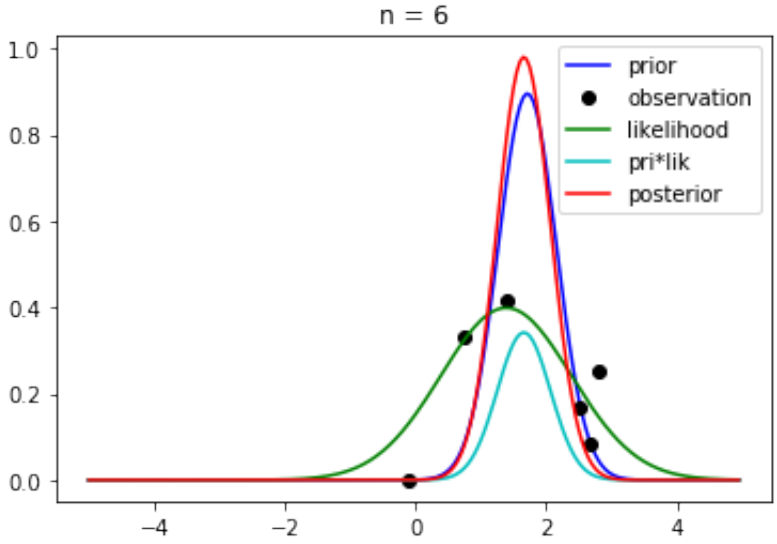

n = 2
prior
observation
likelihood
pri*lik
posterior
0.5
0.4
0.3
0.2
0.1
0.0
−4
−2
0
2
4


n = 3
prior
observation
likelihood
pri*lik
posterior
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−4
−2
0
2
4


n = 4
prior
observation
likelihood
pri*lik
posterior
0.8
0.7
0.6
0.5
0.4
0.3
0.2







For Gaussian prior and likelihood, the posterior mean is a Gaussian $P(\mu|Y) = N(\mu_n, \sigma_n)$ with

$$\mu_n = \frac{\sigma^2 \mu_0 + \sigma_0^2 \sum_i^n y_i}{n\sigma_0^2 + \sigma^2}$$

$$\frac{1}{\sigma_n^2} = \frac{1}{\sigma_0^2} + \frac{n}{\sigma^2}$$

In [ ]:
```
1
```

Dynamic Bayesian Inference

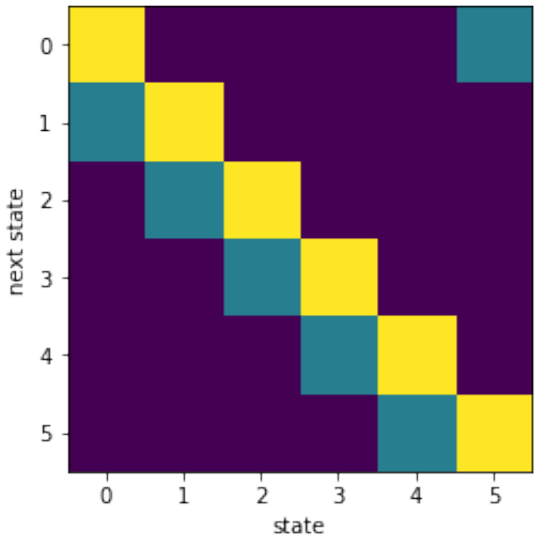
Here is a simple example of target tracking by a Hidden Markov model

In [6]:

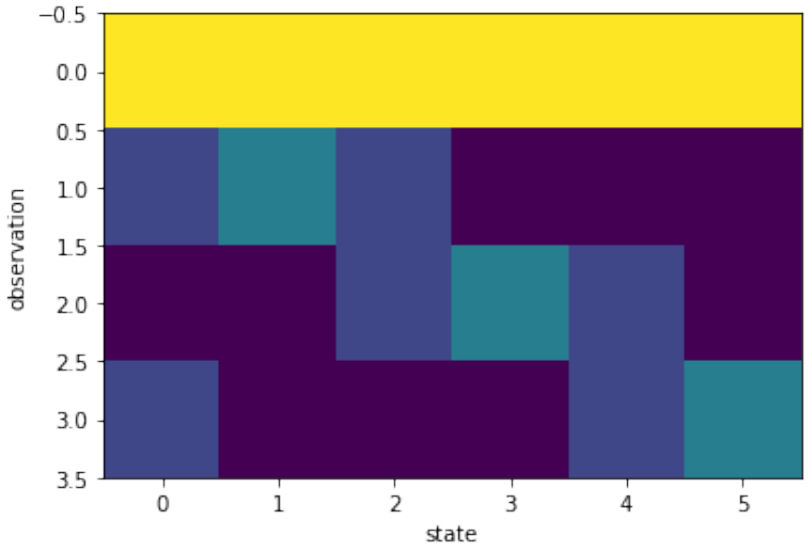
```
class HMM():
    """Hidden Markov model"""

    def __init__(self, ptr, pobs):
        """Create HMM with transition and observation models"""
        self.ptr = ptr    # transition matrix p(x'|x)
        self.pobs = pobs    # observation model
        self.Ns = len(ptr)  # number of states
        self.No = len(pobs)  # number of observations
        self.pst = np.ones(self.Ns)/self.Ns  # state distribution
        self.pred = np.zeros(self.Ns)  # predictive distribution

    def sample(self, x0=0, step=10):
        """generate a sample sequence from x0"""
        xt = np.zeros(step, dtype=int) # state sequence
        yt = np.zeros(step, dtype=int) # observation sequence
        xt[0] = x0
        po = self.pobs[:, x0] # prob. of observation
        yt[0] = np.random.choice(self.No, p=po) # observe
        for t in range(1, step):
            ps = self.ptr[:, xt[t-1]]  # prob. of new states
            xt[t] = np.random.choice(self.Ns, p=ps) # transit
            po = self.pobs[:, xt[t]]   # prob. of observation
            yt[t] = np.random.choice(self.No, p=po) # observe
        return xt, yt

    def predict(self):
        """predictive prior by transition model"""
        self.pred = self.ptr @ self.pst

    def update(self, obs):
        """update posterior by observation"""
        prl = self.pobs[obs]*self.pred # likelihood*prior
        self.pst = prl/sum(prl)  #normalize

    def reset(self):
        """reset state probability"""
        self.pst = np.ones(self.Ns)/self.Ns  # uniform

    def step(self, obs):
        """one step of dynamic bayesian inference"""
        self.predict()
        self.update(obs)
        return self.pst  # new prior
```

Here is an example of directed random walk on a ring.

In [7]:
```
# random walk on a ring
ns = 6   # ring size
ps = 0.3  # shift probability
Ptr = np.zeros((ns, ns))  # transition matrix
for i in range(ns):
    Ptr[i,i] = 1 - ps
    Ptr[(i+1)%ns, i] = ps
plt.imshow(Ptr)
plt.xlabel("state"); plt.ylabel("next state");
```

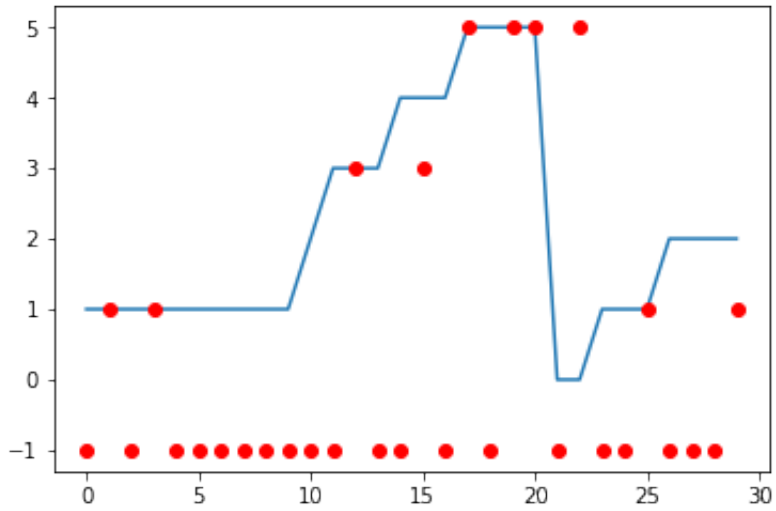


In [8]:
```
# Blurred intermittent observation model
no = 4
po = 0.3
Pobs = np.zeros((no, ns))  # p(obs|state)
Pobs[0,:] = 1 - po  # no information
Pobs[1,1] = Pobs[2,3] = Pobs[3,5] = po
Pobs[1,0] = Pobs[3,0] = Pobs[1:3,2] = Pobs[2:4,4] = po/2
plt.imshow(Pobs)
plt.xlabel("state"); plt.ylabel("observation");
```

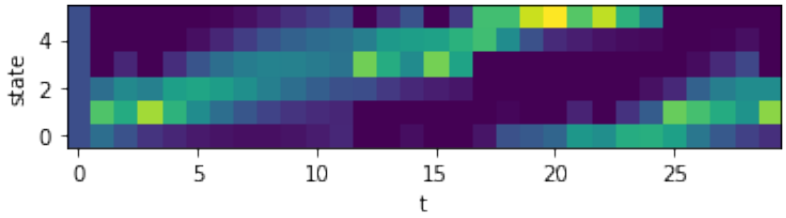


In [9]:
```
# crate a HMM
ring = HMM(Ptr, Pobs)
```

In [10]:
```
# sample a trajectory
T = 30
xt, yt, = ring.sample(1, T)
plt.plot(xt)
plt.plot(2*yt-1, 'ro');
```



In [11]:
```
# Dynamic Bayesian inference
post = np.zeros((T, ns))  # posterior trajectory
ring.reset()
for t in range(T):
    post[t] = ring.step(yt[t])
plt.imshow(post.T, origin='lower')
plt.xlabel('t'); plt.ylabel('state');
```



In [ ]:
```
1
```

In [ ]:
```
1
```