

Computer Practical: Markov Chains — Model answers

General comments

In task 5 it is important not to sample from the marginal distribution $\lambda'K^{(m)}$, but from the conditional distribution of $X_{t+1}|X_t = x_t$. Otherwise, the simulated sample path has an incorrect dependency structure.

Model code in Python

We start with importing the necessary libraries and defining the function `discrete_sample` (from the handout):

```

1 from scipy import *
2 from scipy import linalg
3 from numpy import random
4 import pylab
5
6 # Draws one realisation from the discrete distribution given by probs
7 def discrete_sample(probs):
8     return sum(cumsum(probs)<random.random_sample())

```

Next, we define a class `MarkovChain`, which carries out all the required computations. The class methods `invariant_distribution` and `invariant_distribution_alternative` both compute the invariant distribution: the former uses the eigen decomposition and the latter solves a system of equations.

```

9 # Definition of a class MarkovChain
10 class MarkovChain:
11     # Constructor: creates a new MarkovChain object
12     def __init__(self, K, initial):
13         self.K = K
14         self.initial = initial
15
16     # Compute m-step transition kernel
17     def K_m_step(self, m):
18         Km = eye(self.K.shape[0]) # Create identity matrix of correct size
19         for i in xrange(m):
20             Km = dot(Km, self.K) # Repeatedly multiply it by K
21         return Km
22
23     # Compute distribution of states at time m
24     def distribution_at_m(self, m):
25         return dot(self.initial, self.K_m_step(m))
26         # Compute lambda'K(m)
27
28     # Compute the invariant distribution
29     def invariant_distribution(self):
30         eigen = linalg.eig(self.K.T) # Find eigenvals and vecs
31         idx = eigen[0].argmax() # Find which eigenval is largest (i.e. 1)
32         mu = eigen[1][:,idx] # Extract corresponding eigenvec
33         mu = mu / sum(mu) # Normalise distribution
34         return mu
35
36     # Compute the invariant distribution (alternative implementation)
37     def invariant_distribution_alternative(self):
38         n = self.K.shape[0] # Get number of rows
39         A = concatenate((self.K.T - eye(n), [ones(n)]))
40         # Compute K'-I and add row of ones
41         b = zeros(n+1) # Set b = [0 ... 0 1]
42         b[n] = 1
43         mu = linalg.lstsq(A, b)[0] # Solve this system of equations
44         return mu
45

```

```

46 # Draw path of length n from the Markov chain
47 def sample_chain(self, n):
48     sample_path = zeros(n)           # Set up vector to hold results
49     sample_path[0] = discrete_sample(self.initial)
50                                     # Draw X[0] from the initial distribution
51     for t in xrange(1,n):           # Draw remaining X[t] from conditional
52                                     # distribution if X[t+1] | X[t]=x[t]
53         sample_path[t] = discrete_sample(K[sample_path[t-1],:])
54
55     return sample_path

```

We can now use the class we have defined to answer tasks 1 to 6.

```

58 # Task 1
59
60 K = array([[0.9, 0.1],              # Define exaple transition kernel
61           [0.3, 0.7]])
62 lam = array([0.2, 0.8])            # Set (arbitrary) initial distribution
63
64 myMC = MarkovChain(K, lam)         # Create MarkovChain object
65
66 print "2-step_Transition_Kernel:"
67 print myMC.K_m_step(2)            # Print 2-step transition kernel
68
69 # Task 2
70
71 print "Distribution_at_Time_t=2:"
72 print myMC.distribution_at_m(2)    # Print distribution of states at t=2
73
74 # Task 3
75
76 print "Invariant_Distribution"
77 print myMC.invariant_distribution() # Print invariant distribution
78
79 # Task 4
80 n = 20
81 distn = array([myMC.distribution_at_m(t) for t in xrange(n+1)])
82                                     # Compute distribution of states
83                                     # for t=0..n
84 pylab.figure()
85 pylab.plot(distn[:,0], 'bs-', distn[:,1], 'rs-')
86 pylab.show()
87
88 # Task 5
89
90 n = 100
91 mySample= myMC.sample_chain(n)     # Draw sample path of length n
92 pylab.figure()
93 pylab.plot(mySample, 'gs-')        # Plot the path
94 pylab.show()
95
96 # Task 6
97
98 print "Proportion_of_0's:"
99 print float(sum(mySample==0))/n
100 print "Proportion_of_1's:"
101 print float(sum(mySample==1))/n

```

Model code in R

We start by defining our transition kernel K and one initial distribution:

```
1 K <- rbind(c(0.9, 0.1),           # Define exaple transition kernel
2           c(0.3, 0.7))
3 lam <- c(0.2,0.8)                 # Set (arbitrary) initial distribution
```

We start by defining a function for computing the m -step transition kernel.

```
4 # Task 1
5
6 # Compute m-step transition kernel
7 K.m.step <- function(K, m=1) {
8   K.m <- diag(nrow(K))           # Create identity matrix of correct size
9   if (m>0)
10    for (i in 1:m)
11      K.m <- K.m %**% K           # Repeatedly multiply it by K
12   K.m
13 }
14
15 K.m.step(K,2)                    # Print 2-step transition kernel
```

Next, we compute the distribution of states at time m .

```
16 # Task 2
17
18 # Compute distribution of states at time m
19 distribution.at.m <- function(K, lambda, m=1) {
20   t(lambda)%**K.m.step(K,m)     # Compute lambda'K(m)
21 }
22
23 distribution.at.m(K, lam, 2)     # Print distribution of states at t=2
```

We can compute the invariant distribution using an eigen decomposition of K' ...

```
24 # Task 3
25
26 # Compute the invariant distribution
27 invariant.distribution <- function(K) {
28   mu <- eigen(t(K))$vectors[,1]  # Find the eigenvector corresponding
29                                   # to the largest eigenvalue (i.e. 1)
30   mu <- mu / sum(mu)             # Normalise distribution
31   mu
32 }
33
34 mu <- invariant.distribution(K)   # Compute invariant distribution
35 mu
```

...or by solving a system of linear equations

```
36 # Compute the invariant distribution (alternative implementation)
37 invariant.distribution.alternative <- function(K) {
38   n <- nrow(K)                   # Get number of rows
39   A <- rbind(t(K)-diag(n),1)     # Compute K'-I and add row of ones
40   b <- rep(0,n+1)                # Set b = [0 ... 0 1]
41   b[n+1] <- 1
42   mu <- qr.solve(A,b)           # Solve this system of equations
43   mu
44 }
45
46 mu <- invariant.distribution.alternative(K)
47 mu                               # Compute invariant distribution
```

We now create the plot shown on the handout.

```
48 # Task 4
49
50 n <- 20
51 distn <- matrix(nrow=n+1,ncol=2) # Create empty matrix to store result
```

```

52 for (i in 1:nrow(distn))
53   distn[i,] <- distribution.at.m(K, lam, i-1)
54                               # Compute distribution of states
55                               # for t=0..n
56 matplot(0:n,distn, type="b", pch=15:16, col=1:2, lty=1, ylim=0:1)
57 abline(h=mu,col=1:2,lty=3)

```

Finally we write the code to simulate one sample path.

```

58 # Task 5
59
60 # Draw path of length n from the Markov chain
61 sample.chain <- function(K, lambda, n) {
62   sample.path <- numeric(n)           # Set up vector to hold results
63   sample.path[1] <- sample(nrow(K), 1, prob=lambda)
64                                   # Draw X[0] from the initial distribution
65   for (t in 2:n)                   # Draw remaining X[t] from conditional
66                                   # distribution if X[t+1] | X[t]=x[t]
67     sample.path[t] <- sample(nrow(K), 1, prob=K[sample.path[t-1],])
68   sample.path
69 }
70
71 mySample <- sample.chain(K, lam, 1000)
72
73 plot(mySample, type="b")
74
75 # Task 6
76
77 table(mySample)

```