CE 530

Assignment #4 Solution

1. Evaluate the integral of $f(x) = 3x^2$ for $x = 0..1$ using Monte Carlo integration with importance sampling. Use the following weighting functions
   (a) $p(x) = 1$
   (b) $p(x) = 2x$
   (c) $p(x) = 4x^3$
   Pick a sample size and perform multiple runs of your program (taking care not to begin all runs with the same random-number seed), taking statistics to compute the variance of the values given by the runs. Compare these results with the variances you would expect from the formula given by Frenkel and Smit (and in class).

A Java code for completing this assignment follows below. It makes use of the `Quadrature`, `Function`, and `MyRandom` classes in the montecarlo package.

The Java code below simply sets up an instance of Quadrature with the desired integrand and specifying that it use 1000 sample points to evaluate the integral. It then sets up the weighting functions and random-number generators to sample on these distributions. It calls the sampling method of Quadrature 1000 times and it computes the standard deviation of the results. The following table summarizes the outcomes.

Weight	*Observed Average*	*Observed Std. Deviation*	*Standard deviation from Frenkel & Smit formula*
1.0	0.9997	0.028	0.028
2x	0.99997	0.011	0.011
$4x^3$	1.0002	0.011	0.011

The observed standard deviation is in perfect agreement with the theoretical value.

```
import montecarlo.*;

import java.awt.*;

import java.applet.*;


//Evaluate the integral of f(x) = 3x^2 for x = 0..1 using Monte Carlo integration with importance
sampling.

//Use the following weighting functions

//   (a) p(x) = 1

//   (b) p(x) = 2x

//   (c) p(x) = 4x^3

//Pick a sample size and perform multiple runs of your program (taking care not to begin all

//runs with the same random-number seed), taking statistics to compute the variance of the

//values given by the runs.  Compare these results with the variances you would expect from

//the formula given by Frenkel and Smit (and in class).


//Output goes to the console (nothing shows up in the applet window)


public class Assignment5_1 extends Applet {


  //The Quadrature class can perform various types of numerical integrations

  //on its function.  We create it here with its function defined as 3x^2

    Quadrature quad = new Quadrature(new Function() {

        public double f(double x) {return 3.0*x*x;}

    });


  //Set up the weight functions and random number generators based on them

    Function p1 = new Function() {public double f(double x) {return 2*x;}};

    Function p3 = new Function() {public double f(double x) {return 4*x*x*x;}};

    MyRandom r1 = new MyRandom(new Function() {public double f(double x) {return Math.sqrt(x);}});

    MyRandom r3 = new MyRandom(new Function() {public double f(double x) {return Math.pow(x,1./4.);}});
```

```
//Number of sample points in each MC calculation
  int nSample = 1000;

//Number of times MC calculation is repeated to measure error statistics
  int nRepeat = 1000;

  public void start() {
      quad.setN(nSample);

    //Unbiased MC calculation
      double sum1 = 0.0;
      double sum2 = 0.0;
      System.out.println("Unbiased MC calculation...");
      for(int i=0; i<nRepeat; i++) {
          double value = quad.simpleMC();
          sum1 += value;
          sum2 += value*value;
      }
      double average = sum1/(double)nRepeat;
      double std = Math.sqrt(sum2/(double)nRepeat - average*average);
      System.out.println("Unbiased average:  "+average);
      System.out.println("Unbiased std dev: "+std);

    //Linear-bias MC calculation
      System.out.println("");
      System.out.println("Linear-bias MC calculation...");
      sum1 = sum2 = 0.0;
      for(int i=0; i<nRepeat; i++) {
          double value = quad.importanceMC(r1, p1);
          sum1 += value;
          sum2 += value*value;
```

```
        }
        average = sum1/(double)nRepeat;
        std = Math.sqrt(sum2/(double)nRepeat - average*average);
        System.out.println("Unbiased average:  "+average);
        System.out.println("Unbiased std dev: "+std);

      //Cubic-bias MC calculation
        System.out.println("");
        System.out.println("Cubic-bias MC calculation...");
        sum1 = sum2 = 0.0;
        for(int i=0; i<nRepeat; i++) {
            double value = quad.importanceMC(r3, p3);
            sum1 += value;
            sum2 += value*value;
        }
        average = sum1/(double)nRepeat;
        std = Math.sqrt(sum2/(double)nRepeat - average*average);
        System.out.println("Unbiased average:  "+average);
        System.out.println("Unbiased std dev: "+std);
    }
}
```

2. Here is a transition probability matrix for a four-state system:

$$\begin{pmatrix} 0.5 & 0.1 & 0.1 & 0.3 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.1 & 0.2 & 0.3 & 0.4 \\ 0.2 & 0.1 & 0.35 & 0.35 \end{pmatrix}$$

(a) Compute analytically the limiting distribution corresponding to these transition probabilities. Use an algebraic approach, in which you solve the balance equations and normalization, as presented in class.

(b) Repeat (a), but by computing the appropriate normalized left eigenvector of the matrix.

(c) Do the transition probabilities and the limiting distribution obey detailed balance?

(d) Perform a random walk for this set of transition probabilities using the Markov process applet here: http://www.eng.buffalo.edu/~kofke/applets/MarkovApplet1.html. Take a screen shot of the distribution after many steps, and submit that with your assignment.

(a) We solve these equations:

```
0.5 x[1] + 0.25 x[2] + 0.1 x[3] + 0.2 x[4] == x[1]
0.1 x[1] + 0.25 x[2] + 0.2 x[3] + 0.1 x[4] == x[2]
0.1 x[1] + 0.25 x[2] + 0.3 x[3] + 0.35 x[4] == x[3]
x[1] + x[2] + x[3] + x[4] == 1
```

where x[i] is the unknown π_i. The solution is

```
{x[1] → 0.259501, x[2] → 0.147936, x[3] → 0.257458, x[4] → 0.335104}
```

(b) The equation for the left eigenvectors is

$$x^T \Pi = \lambda x^T$$

If you prefer, this can be written as the conventional (right) eigenvectors of the transpose of the transition-probability matrix

$$x^T \Pi = \lambda x^T$$
$$\left(x^T \Pi\right)^T = \left(\lambda x^T\right)^T$$
$$\left(\Pi\right)^T \left(x^T\right)^T = \lambda \left(x^T\right)^T$$
$$\Pi^T x = \lambda x$$

The eigenvalues and eigenvectors are

```
In[102]:= pi // TableForm

Out[102]//TableForm=
0.5     0.1     0.1     0.3
0.25    0.25    0.25    0.25
0.1     0.2     0.3     0.4
0.2     0.1     0.35    0.35
```

```
In[109]:= eig = Eigensystem[Transpose[pi]];
          eig // TableForm

Out[110]//TableForm=
1.           0.341632     0.144475     -0.086107
-0.501453    -0.837976    0.293871     -0.218108
-0.285867    0.24632      -0.823125    0.294341
-0.497504    0.472029     0.0454808    -0.69496
-0.647545    0.119626     0.483774     0.618727
```

In the last matrix, the top row are the eigenvalues, and below each is its corresponding eigenvector. We are interested in the eigenvector corresponding to $\lambda = 1$. The eigenvectors are given to within an arbitrary multiplicative constant, so we can divide by the sum of its terms to get a set of normalized probabilities

```
In[114]:= norm = Sum[eig[[2, 1, k]], {k, Length[eig[[2, 1]]]}]
          limitDist = eig[[2, 1]] / norm

Out[114]= -1.93237

Out[115]= {0.259501, 0.147936, 0.257458, 0.335104}
```

These are consistent with the result from (a).

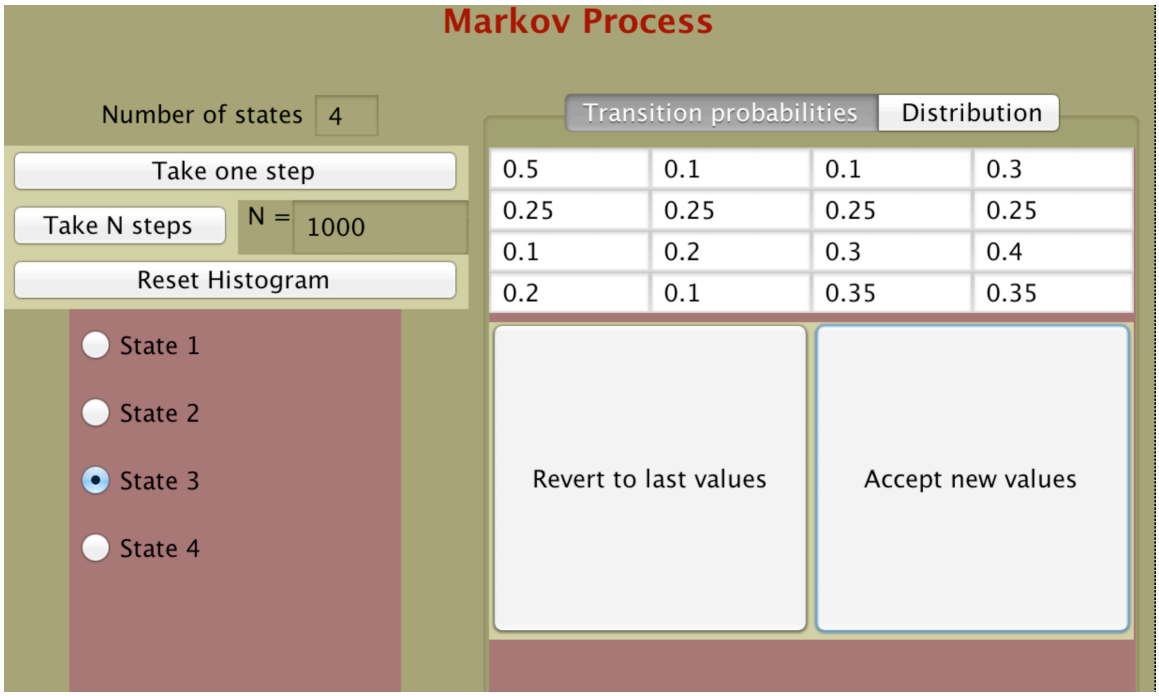
(c) We check whether $\pi_i \pi_{ij} = \pi_j \pi_{ji}$. This is shown by the symmetry of the matrix $P\Pi$ (see solution to problem 4). Here it is:

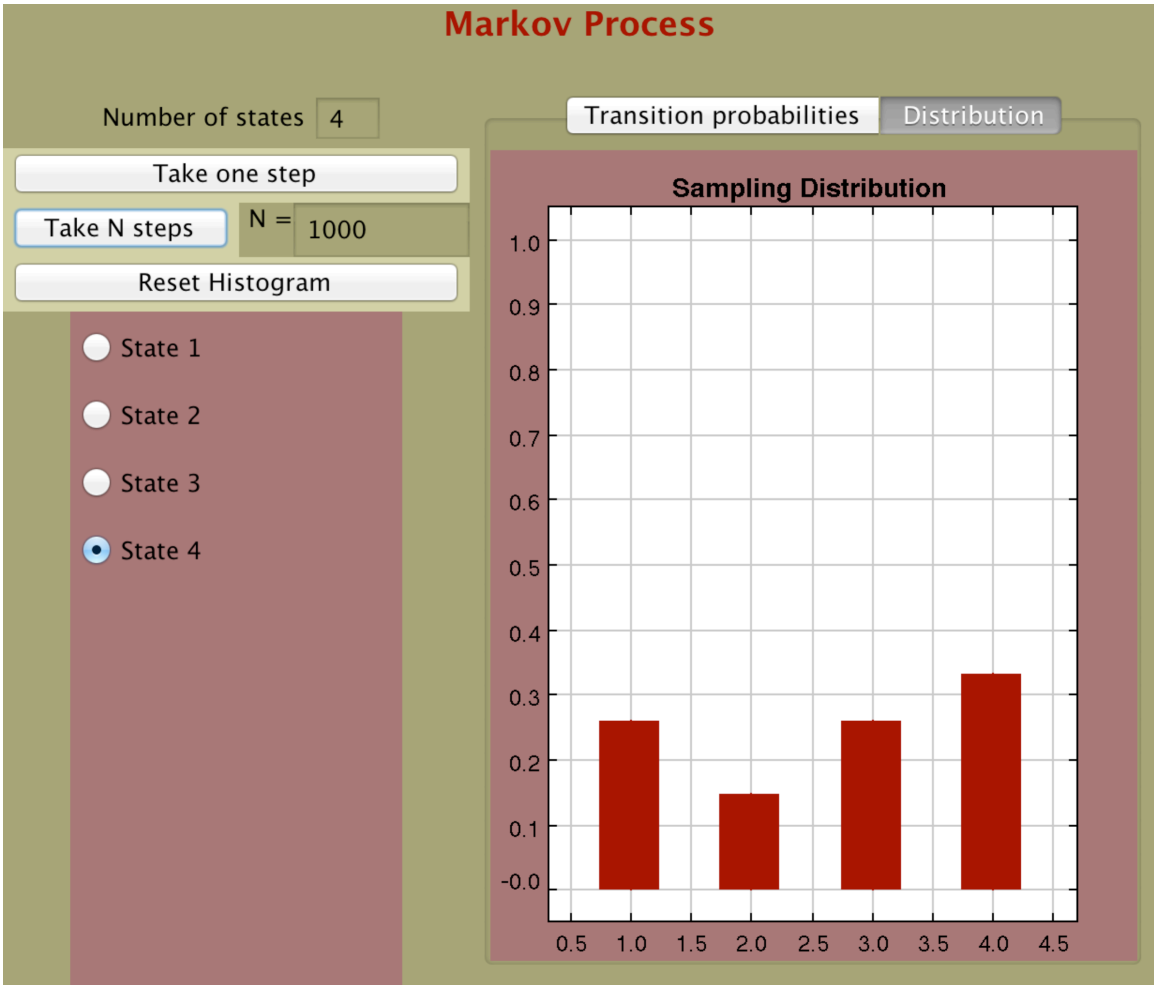
```
In[122]:= Ppi = DiagonalMatrix[limitDist].pi;
          TableForm[Ppi]

Out[123]//TableForm=
0.129751     0.0259501    0.0259501    0.0778504
0.0369841    0.0369841    0.0369841    0.0369841
0.0257458    0.0514916    0.0772374    0.102983
0.0670208    0.0335104    0.117286     0.117286
```

It is not symmetric, so detailed balance is not satisfied.

(d)


Markov Process
Number of states 4
Take one step
Take N steps
N = 1000
Reset Histogram
State 1
State 2
State 3
State 4
Transition probabilities
Distribution
0.5 0.1 0.1 0.3
0.25 0.25 0.25 0.25
0.1 0.2 0.3 0.4
0.2 0.1 0.35 0.35
Revert to last values
Accept new values



That looks like the limiting distribution we derived.

3. Here is a transition probability matrix for a four-state system:

$$\begin{pmatrix} 0.1 & 0.9 & 0 & 0 \\ 0.5 & 0.5 & 0 & 0 \\ 0 & 0 & 0.3 & 0.7 \\ 0 & 0 & 0.6 & 0.4 \end{pmatrix}$$

A Markov process that attempts to sample according to these transition probabilities will be flawed. What is wrong?

It is not ergodic. There is no way to get from states 1 or 2 to states 3 or 4, or vice versa.

4. Consider the following probability distribution for a 4-state system:

$$\pi_1 = 0.2 \quad \pi_2 = 0.1 \quad \pi_3 = 0.5 \quad \pi_4 = 0.2$$

(a) Derive a set of transition probabilities according to the Metropolis algorithm that will yield this as the limiting distribution of a corresponding Markov process on the four states.

Here's the Mathematica code that does this. It also evaluates the λ=1 eigenvector to show it does correspond to the desired distribution (you didn't have to do this for the problem).

```
In[128]:= pij[i_, j_, p_List] := Min[1, p[[j]] / p[[i]]] / (Length[p] - 1) /; i ≠ j
          pij[i_, j_, p_List] := 1 - Sum[pij[i, k, p], {k, 1, j - 1}] -
            Sum[pij[i, k, p], {k, j + 1, Length[p]}]
          metropolis[p_List] := Table[pij[i, j, p], {i, 1, Length[p]}, {j, 1, Length[p]}]

In[145]:= met = metropolis[{2 / 10, 1 / 10, 5 / 10, 2 / 10}];
          met // TableForm
          eig = Eigensystem[Transpose[met]];
          eig // TableForm // Chop
          eig[[2, 1]] / Sum[eig[[2, 1, k]], {k, Length[eig[[2, 1]]]}]
```

Out[146]//TableForm=

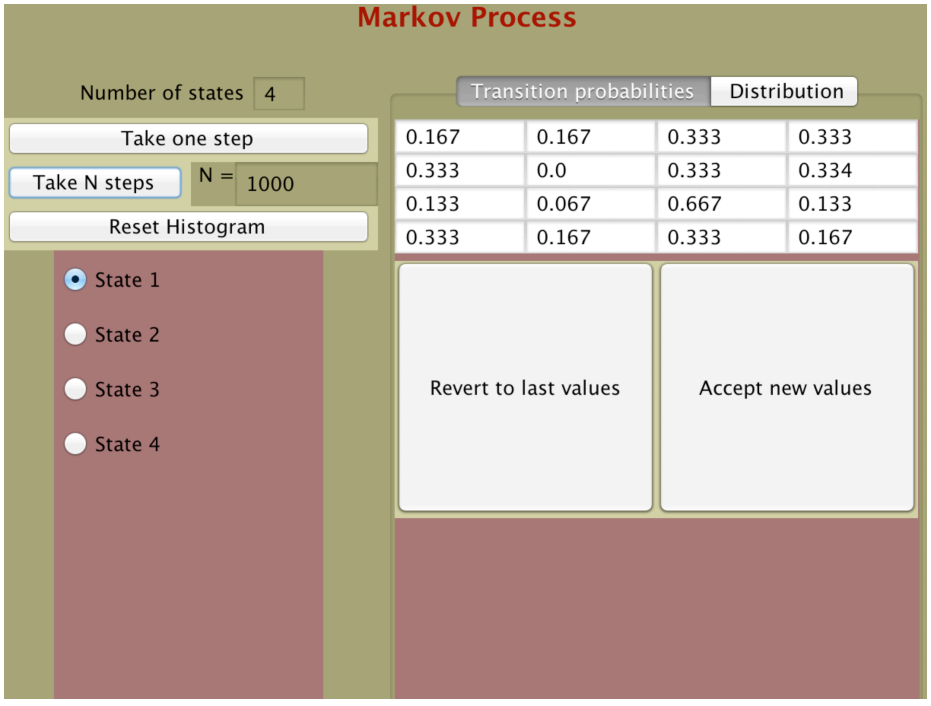
$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$
$\frac{1}{3}$	0	$\frac{1}{3}$	$\frac{1}{3}$
$\frac{2}{15}$	$\frac{1}{15}$	$\frac{2}{3}$	$\frac{2}{15}$
$\frac{1}{3}$	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{6}$

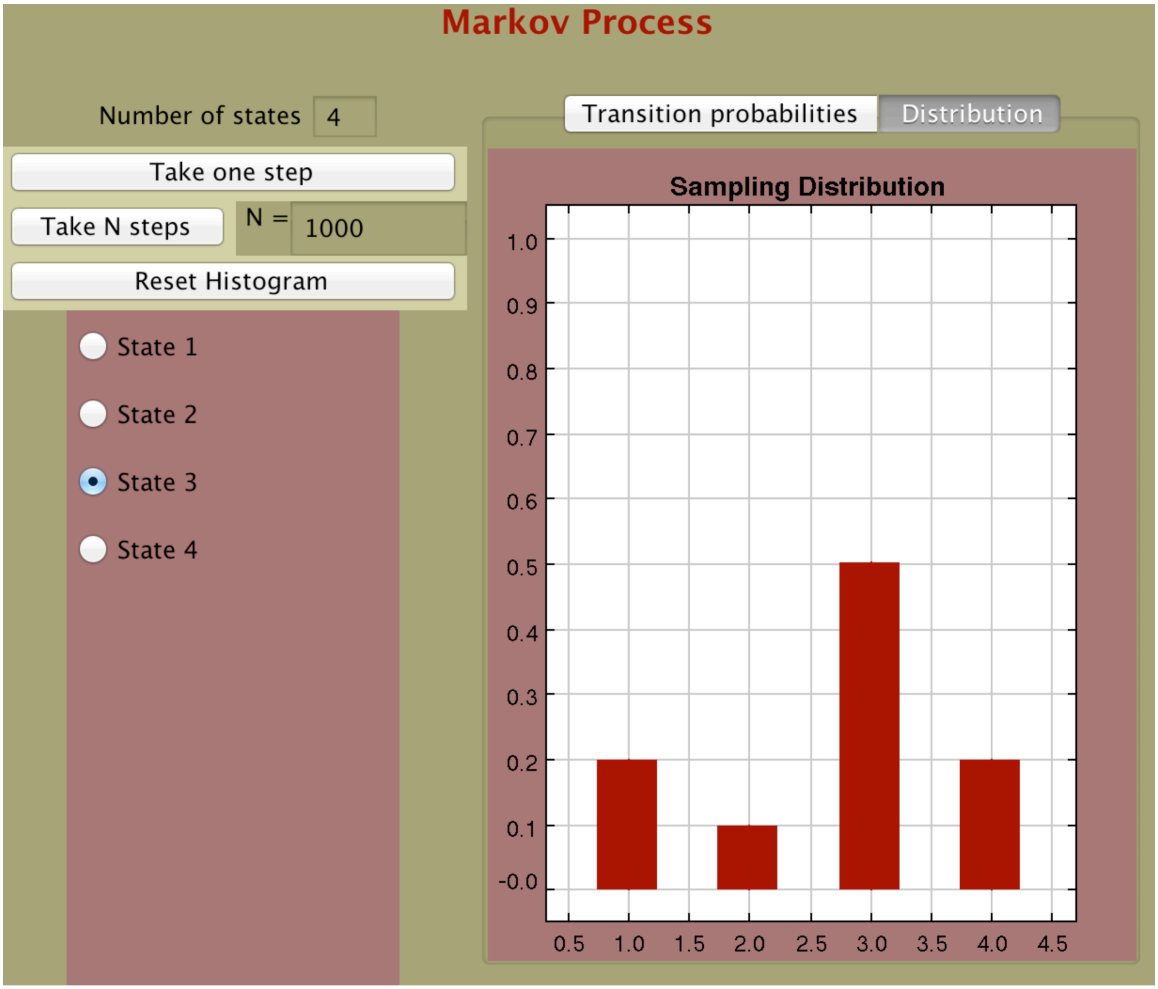
Out[148]//TableForm=

1	$\frac{1}{3}$	$-\frac{1}{6}$	$-\frac{1}{6}$
1	1		
$\frac{1}{2}$	$\frac{1}{2}$	-1	-1
$\frac{5}{2}$	$-\frac{5}{2}$	0	1
1	1	1	0
			0

Out[149]= $\left\{\frac{1}{5}, \frac{1}{10}, \frac{1}{2}, \frac{1}{5}\right\}$

(b) Use these transition probabilities in the Markov applet to show that they do indeed give the desired limiting distribution. Submit a screenshot of the distribution you get.




Markov Process
Number of states
4
Take one step
Take N steps
N =
1000
Reset Histogram
State 1
State 2
State 3
State 4
Transition probabilities
Distribution
Sampling Distribution
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
-0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5

5. It is common practice in Monte Carlo simulations to increment a running sum not after each and every elementary Monte Carlo step, but instead to do the increment only after some fixed number of elementary steps have been taken. This might be done, for example, because the calculation involved in getting a value for the current configuration is expensive, and it is not worth doing after every little change in the configuration.

   In effect, this procedure describes a "super-Markov" sequence, in which each subsequent configuration in the sequence is obtained by an intervening series of simpler Markov steps. Show that if the transitions probabilities for the simpler Markov steps obey detailed balance, then the transition probabilities for this super-Markov sequence also obey detailed balance for the same limiting distribution.

The transition probabilities for the multi-step move are given by Π^n, where Π is the transition-probability matrix (TPM) for the elementary (single-step) Markov process. We want to show that Π^n obeys detailed balance for the same limiting distribution as Π. Note that it is pretty easy to argue that they both correspond to the same limiting distribution:

$$\lim_{k\to\infty}\left[\Pi^n\right]^k = \lim_{k\to\infty}\Pi^k$$

and this is really all that matters practically in the question at hand. Detailed balance is a sufficient condition for a set of transition probabilities to give a particular limiting distribution, but it is not necessary. So the fact that both TPMs correspond to the same limiting distribution does not imply that they both obey detailed balance. Let us show that they do.

The detailed balance condition can be written compactly in matrix form. Let the matrix P be a diagonal matrix with diagonal terms given the probabilities in the limiting distribution

$$P = \begin{pmatrix} \pi_1 & 0 & 0 \\ 0 & \pi_2 & 0 \\ 0 & 0 & \pi_3 \end{pmatrix}$$

where we take a 3-state system for the illustration. The matrix $P\Pi$ is

$$P\Pi = \begin{pmatrix} \pi_1\pi_{11} & \pi_1\pi_{12} & \pi_1\pi_{13} \\ \pi_2\pi_{21} & \pi_2\pi_{22} & \pi_2\pi_{23} \\ \pi_3\pi_{31} & \pi_3\pi_{32} & \pi_3\pi_{33} \end{pmatrix}$$

Clearly, a simple statement of detailed balance is that this matrix is symmetric

$$P\Pi = \left(P\Pi\right)^T$$

We want to show that this condition implies the same for Π^n. Let us do this by showing that if Π^n obeys detailed balance then Π^{n+1} does also. Then by induction it will hold for any *n* since it holds for *n* = 1. We begin with the matrix identity

$$P\Pi^{n+1} = \left(P\Pi^n\right)\Pi$$
$$\left[\Pi^T\left(P\Pi^n\right)^T\right]^T$$

Given that Π^n obeys detailed balance this is

$$P\Pi^{n+1} = \left[\Pi^T\left(P\Pi^n\right)\right]^T$$

and since P is symmetric

$$P\Pi^{n+1} = \left[\Pi^T P^T \Pi^n\right]^T$$

with another matrix identity we have

$$P\Pi^{n+1} = \left[\left(P\Pi\right)^T \Pi^n\right]^T$$

and finally, invoking detailed balance for Π

$$P\Pi^{n+1} = \left[P\Pi\Pi^n\right]^T$$
$$= \left[P\Pi^{n+1}\right]^T$$

Proving that Π^{n+1} obeys detailed balance too.