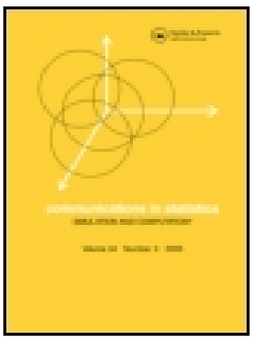

Taylor & Francis
Taylor & Francis Group

Communications in Statistics - Simulation and Computation

ISSN: 0361-0918 (Print) 1532-4141 (Online) Journal homepage: http://www.tandfonline.com/loi/lssp20

MCMC and the Fibonacci Distribution

Myron Hlynka

To cite this article: Myron Hlynka (2015): MCMC and the Fibonacci Distribution, Communications in Statistics - Simulation and Computation, DOI: 10.1080/03610918.2015.1085558

To link to this article: http://dx.doi.org/10.1080/03610918.2015.1085558

MCMC and the Fibonacci distribution

Myron Hlynka
Department of Mathematics and Statistics
University of Windsor
Windsor, Ontario, Canada N9B 3P4
hlynka@uwindsor.ca

Keywords: MCMC, Markov chain Monte Carlo, Fibonacci

Abstract

A method to create a Markov transition matrix for Markov Chain Monte Carlo studies is presented and applied to the Fibonacci probability distribution.

1 Introduction

Markov Chain Monte Carlo (MCMC) is a simulation method that has become enormously popular in the past 40 years. See [9], [4], [11], [7], [12] as examples of recent work. Although excellent descriptions of the MCMC technique exist (see [1] or [8], for example), it is still difficult for a novice to grasp the topic. This paper performs MCMC by setting up an appropriate transition matrix in a new manner, and applies it to the Fibonacci distribution ([10]). An R program is included in the appendix.

In section 2, we present the Fibonacci distribution on which we perform our MCMC simulation. This section includes our new development of an appropriate Markov chain transition matrix.

In section 3, we illustrate the use of the R program and comment on the results. The actual program appears in the appendix.

2 Matrix Development

In [10], Shane [essentially] defines the Fibonacci distribution to have probability mass function

$$f(x) = \frac{F_{x-1}}{2^x} \tag{1}$$

for $x = 2, 3, \ldots$ where $F_1 = 1, F_2 = 1$ and $F_x = F_{x-1} + F_{x-2}$ for $x = 3, 4, \ldots$ are the Fibonacci numbers. The Fibonacci distribution is quite interesting. The probability mass $f(x) = P(X = x)$ represents the probability that x trials are needed to observe two consecutive heads when flipping a fair coin. This might be used as a model to test the learning of a skill through two consecutive correct answers on a T/F test or it might be used to quickly search for the most promising medical treatments from a huge list when assuming equipoise ("a state of genuine uncertainty on the part of the clinical investigator", Freedman [3]), by having two consecutive successes. The Fibonacci distribution is also discussed in Medhi ([6]).

Suppose we wish to simulate observations from the Fibonacci distribution using MCMC. First we compute the ratio of consecutive Fibonacci probabilities.

$$\frac{f(x+1)}{f(x)} = \frac{F_x}{2^{x+1}} \Big/ \frac{F_{x-1}}{2^x} = \frac{F_x}{2F_{x-1}}. \tag{2}$$

Rather than using the Rosenbluth-Hastings (a.k.a. Metropolis-Hastings; see [8]) method, with its candidate states, to generate

the transition probabilities for given limiting ratios, we use our own method. Our transition matrix could result from a special choice in the Rosenbluth-Hastings method, but our development is completely different. We intend to create a Markov chain on the state space $\{2, 3, 4, \dots\}$ with state varying probabilities. We will use a birth and death type of system to create a Markov chain which will give us the desired limiting probability distribution $\{f(x)\}$. Because of the ratio in (2), we have

$$2F_{x-1}f(x+1) = F_x f(x).$$

We take this equation to be the balance equation for a yet-to-be-described continuous time Markov process. The LHS is the rate from state $x+1$ to state x and is the product of the limiting probability $f(x+1)$ of being in state $x+1$ times the rate $2F_{x-1}$ of moving to state x given that the system is in state $x+1$. The RHS is the rate from state x to state $x+1$ which is the product of the limiting probability $f(x)$ of being in state x times the rate F_x of moving to state $x+1$ given that the system is in state x.

Our birth and death transition diagram looks like

$$\begin{array}{ccccccccc} & F_2 & & F_3 & & F_4 & & F_5 & \\ & \rightarrow & & \rightarrow & & \rightarrow & & \rightarrow & \\ \textcircled{2} & & \textcircled{3} & & \textcircled{4} & & \textcircled{5} & & \textcircled{6}\cdots \\ & \leftarrow & & \leftarrow & & \leftarrow & & \leftarrow & \\ & 2F_1 & & 2F_2 & & 2F_3 & & 2F_4 & \end{array}$$

In the usual setting for birth and death processes, all of the limiting probabilities are found in terms of one base probability (usually π_0, but in our setting, there is no π_0 and our base probability is π_2).

This base probability is then determined by the condition that all of the probabilities sum to 1 (see [2],p. 255). In our case, if we label the limiting probabilities as $\pi_2 = f(2), \pi_3 = f(3), \pi_4 = f(4), \ldots$, we find

$\pi_3 = \frac{F_2}{2F_1}\pi_2, \pi_4 = \frac{F_2F_3}{2F_1 2F_2}\pi_2, \pi_5 = \frac{F_2F_3F_4}{2F_1 2F_2 2F_3}\pi_2, \ldots,$

so $\pi_3 = \frac{F_2}{2F_1}\pi_2, \pi_4 = \frac{F_3}{2^2F_1}\pi_2, \pi_5 = \frac{F_4}{2^3F_1}\pi_2, \ldots,$

and $1 = \pi_2 + \pi_3 + \cdots = \frac{\pi_2}{F_1}(1 + \sum_{i=2}^{\infty}\frac{F_i}{2^i})$

We get the same limiting probabilities as long as the ratios of paired rates (upper over lower, e.g. F_2 over $2F_1$) are maintained. To prevent the rates from becoming arbitrarily large, we divide each pair by the sum of the two components (leaving the same ratio) to get a new state transition diagram, with the same limiting probabilities.

$$\underset{\rightarrow}{\frac{F_2}{F_2+2F_1}} \quad \underset{\rightarrow}{\frac{F_3}{F_3+2F_2}} \quad \underset{\rightarrow}{\frac{F_4}{F_4+2F_3}} \quad \underset{\rightarrow}{\frac{F_5}{F_5+2F_4}}$$

$$\textcircled{2} \qquad \textcircled{3} \qquad \textcircled{4} \qquad \textcircled{5} \qquad \textcircled{6} \cdots$$

$$\underset{\leftarrow}{\frac{2F_1}{F_2+2F_1}} \quad \underset{\leftarrow}{\frac{2F_2}{F_3+2F_2}} \quad \underset{\leftarrow}{\frac{2F_3}{F_4+2F_3}} \quad \underset{\leftarrow}{\frac{2F_4}{F_5+2F_4}}$$

The corresponding infinitesimal generator matrix (see [2], p. 243, for example) for states $2, 3, \ldots$ (with the pairs of rates appearing in the off-diagonal positions) is

$$Q = \begin{bmatrix} a & \frac{F_2}{F_2+2F_1} & 0 & 0 & 0 & \dots \\ \frac{2F_1}{F_2+2F_1} & b & \frac{F_3}{F_3+2F_2} & 0 & 0 & \dots \\ 0 & \frac{2F_2}{F_3+2F_2} & c & \frac{F_4}{F_4+2F_3} & 0 & \dots \\ 0 & 0 & \frac{2F_3}{F_4+2F_3} & d & \frac{F_5}{F_5+2F_4} & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix},$$

where $a, b, c, d, \dots$ are negative values chosen so that each row sums to 0.

If we let $\vec{\pi}$ represent the limiting row vector (of Fibonacci probabilites), and let $\vec{0}$ be the row vector of zeros, then standard results for Markov processes (see [2], p. 254) give $\vec{0} = \vec{\pi}Q$. We intend to convert our setting to a discrete time Markov chain so we want to adjust the entries in Q. Since the row sums of the non-negative entries could be greater than 1, we choose to work with Q^* instead, where $Q^* = Q/2$, giving the same limiting probabilities. Then $\vec{0} = \vec{\pi}Q^*$. All entries of Q^*, excluding the diagonal entries, are less than .5 in absolute value. This is true not only in this construction for Fibonacci probabilities, but in general, due to our construction.

Next we add $\vec{\pi}$ to both sides to get $\vec{\pi} = \vec{\pi}(I + Q^*)$. Define $P = I + Q^*$ so P satisfies $P = \vec{\pi}P$ and P has row sums of 1 with non-negative entries. So P is a discrete time probability transition matrix. This is the uniformization technique; see [5], p. 183.) Now that we have converted our setting to a discrete time Markov chain, we can specify the precise Markov transition matrix that we will use.

Here $P = I + Q^*$ so

$$P = \begin{bmatrix} 1+.5a & \frac{.5F_2}{F_2+2F_1} & 0 & 0 & 0 & \dots \\ \frac{F_1}{F_2+2F_1} & 1+.5b & \frac{.5F_3}{F_3+2F_2} & 0 & 0 & \dots \\ 0 & \frac{F_2}{F_3+2F_2} & 1+.5c & \frac{.5F_4}{F_4+2F_3} & 0 & \dots \\ 0 & 0 & \frac{F_3}{F_4+2F_3} & 1+.5d & \frac{.5F_5}{F_5+2F_4} & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

$$\approx \begin{bmatrix} .833 & .167 & 0 & 0 & 0 & \dots \\ .333 & .417 & .25 & 0 & 0 & \dots \\ 0 & .25 & .536 & .214 & 0 & \dots \\ 0 & 0 & .286 & .48 & .227 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}.$$

The states are $2, 3, 4, \dots$. This is a probability transition matrix that has its limiting probability vector of the Fibonacci probabilities and will be used to generate random values of a Fibonacci distribution.

3 R program and Results

First we use R to generate 1000000 uniform(0,1) values. If we are in state x at step i, then we move to state $x-1$, x, or $x+1$ at step $i+1$ with probabilities $\frac{F_{x-2}}{F_{x-1}+2F_{x-2}}$, $1 - \frac{F_{x-2}}{F_{x-1}+2F_{x-2}} - \frac{F_x}{F_x+2F_{x-1}}$, $\frac{.5F_x}{F_x+2F_{x-1}}$ respectively, for $x = 2, 3, \dots$ where we define $F_0 = 0$.

This is particularly easy to do in R, as we simply obtain $x[i+1]$ from $x[i]$ by subtracting 1 from $x[i]$ if our uniform(0,1) value is less than $\frac{F_{x[i]-2}}{F_{x[i]-1}+2F_{x[i]-2}}$, by adding 1 if our uniform(0,1) value lies in

$(1 - \frac{.5F_{x[i]}}{F_{x[i]} + 2F_{x[i]-1}}, 1)$ and doing nothing otherwise. Note that if the state is $x = 2$, then there is no probability of decreasing. The R commands to implement the Markov chain, based on the transition matrix P, appear in the appendix.

We print out the first 100 (of the 1000000) dependent values of a single run of the commands to illustrate typical output.

```
x[1:100]
  [1] 2 2 2 2 2 2 2 3 4 3 3 3 2 2 2 2 2 3 3 2 2 2 2 2 2 2 2
 [28] 2 2 2 2 3 2 3 3 4 4 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3
 [55] 4 4 4 5 5 4 5 5 6 5 6 6 5 5 4 4 5 5 5 4 4 5 5 5 5 5 5
 [82] 4 4 4 3 2 2 2 2 3 4 4 4 4 3 3 2 2 2 2
```

Note the clear dependence of the state at each step on the state at the previous step. Although these values are not independent, the ergodic theorem (see [5], p.95) states that the long run proportion of steps at each state matches the limiting distribution. So we can use the MCMC values that we have created to estimate the pdf (if we did not already know it), and if we scramble the values, we can treat our values as if they were randomly chosen from the pdf.

To check that our method has the desired limiting probabilities, we present three plots together. The first plot gives the exact Fibonacci probabilities. The second plot gives the relative frequencies of random values generated with our MCMC method. The third plot gives random values generated from the exact probabilities using the inverse fubnction method, and taking advantage of R's "sample" command. The plots are almost indistinguishable. Thus MCMC

did a very good job. The commands to generate the three plots appear in the appendix.

[Insert Diagram here]

The estimated probabilites from the MCMC simulation and the true probabilities of $\{2, 3, 4, 5, 6, 7, 8\}$ are:

x	2	3	4	5	6	7	8
Est.Prob.	0.2486	0.1244	0.1253	0.0945	0.0795	0.0634	0.0513
TrueProb.	0.2500	0.1250	0.1250	0.0938	0.0781	0.0625	0.0508

The estimated probabilites from the inverse function simulation and the true probabilities of $\{2, 3, 4, 5, 6, 7, 8\}$ are:

x	2	3	4	5	6	7	8
Est.Prob.	0.2506	0.1253	0.1249	0.0932	0.0778	0.0623	0.0510
TrueProb.	0.2500	0.1250	0.1250	0.0938	0.0781	0.0625	0.0508

Again we see how close the simulated probabilites are to the true probabilites.

Also of interest is the right tail of the two simulations. Of the 1000000 values generated by the two methods, we get counts of the highest 4 values of x as follows. In this case we have

MCMC (x,count)	(49,9)	(50,7)	(51,6)	(52,2)
InvFunctionCount	(58,1)	(61,3)	(65,1)	(70,1)

Both simulation methods are somewhat unsatisfactory here. The MCMC method fails to pick up any values over 52, when one expect that some should be present. (And if MCMC did find a large values,

every smaller value must also be present). The inverse function method, which is an exact method, has large gaps in the upper values, which suggest the particular pattern at the tail would be highly unlikely to appear again for some time and could not be reliably used to represent the upper tail.

4 Comparison of Methods and Comments

Our MCMC method is actually a special case of the Rosenbluth-Hasting MCMC method. However, the development is different. Our MCMC method of generating random values requires us to know only the ratio of the probabilities of interest (just as the usual Rosenbluth-Hastings method does). However, we do not explicitly refer to proposal distributions and acceptance probabilities (although they are present implicitly). The programming required for our method is very simple and requires only a uniform random number generator, and a very simple computation.

The inverse function method is the standard method to simulate many random variables. However, it requires one to know or to be able to compute the exact form of the mass function or density function. That is known for the Fibonacci distribution but is unknown in many other cases. In most languages other than R, the programming becomes more difficult than our MCMC method.

Other Comments:

1. The "sample" command in R picks a random subset, which means that for a large population and a small sample size, the values selected behave essentially as being independent. In our R com-

mands, if the vector x represents the 1000000 simulated MCMC values from the Fibonacci distribution, so the command sample(x)[1:20] will give 20 nearly independent values from the Fibonacci distribution. The output from sample(x)[1:20] is

```
4  4  3  5  4  2  4  2  7  2  2 12  2  6 16  2  5  9  5  5  3
```

These values indeed look like independent values from the Fibonacci distribution.

2. In Bayesian analyses, we often encounter distributions for which the ratio of the probabilities is easily obtained, so MCMC works well for such distributions.

3. Although our results are for the Fibonacci distribution, the same method would work for many other discrete distributions, as long as the ratios of probabilities are easy to obtain.

4. The matrix P that was created in this study is in fact a special case of one that could be produced by the Rosenbluth-Hastings algorithm, although it certainly would not be a natural choice.

5. If we know the probability mass function, then we can easily find moments and other useful measures, so we might wonder why simulation of the distribution is needed. One answer is that if we are testing a new strategy (e.g. scheduling, cost based queueing) then we need to have actual values from the distribution, not just the probabilities, so we can perform a simulation to see the effect of the strategy.

7. On advantage of MCMC over other simulation methods is that the estimated probabilities based on MCMC (of the type presented

here) will not have an estimated probability of zero with non zero neighboring estimates on both sides. For small probabilities far from the median, this makes the MCMC results appear more reasonable (especially to nonstatisticians).

References

[1] Albert,J. and Rizzo, M. (2012). R by Example. Chapter 13. Springer.

[2] Kao, E. (1997) An Introduction to Stochastic Processes. Duxbury.

[3] Freedman, B. (1987). Equipoise and the ethics of clinical research. New England Journal of Medicine, 17:3, 141-145.

[4] Herbei, R. and Berliner, L.M., (2014) Estimating Ocean Circulation: An MCMC Approach With Approximated Likelihoods via the Bernoulli Factory. J. Amer. Statist. Assoc. 109:507, 944–954.

[5] Medhi, J. (2010) Stochastic Processes. New Age Science.

[6] Medhi, J. (2013) Success Runs in Symmetric Bernoulli Process. Missouri J. Math. Sci. 25:2, 215–219.

[7] Papamarkou, T., Mira, A., and Girolam, M. (2014) Zero Variance Differential Geometric Markov Chain Monte Carlo Algorithms. Bayesian Analysis, 9:1, 97–128

[8] Press, B. (viewed 2014) Opinionated Lessons in Statistics. #39 MCMC and Gibbs Sampling. Video lecture. (history at 15:45 in video) http://www.youtube.com/playlist?list=PLUAHeOPjkJseXJKbuk9-hlOfZU9Wd6pS0

[9] Sadegh, M., and J.A. Vrugt (2014) Approximate Bayesian Computation using Markov Chain Monte Carlo simulation: DREAM(ABC), Water Resour. Res., 50, 6767-6787, doi:10.1002/2014WR015386.

[10] Shane, H.D. (1973) A Fibonacci Probability Function. Fibonacci Quarterly. 11:5, 517–522.

[11] Xiang, F. and Neal, P. (2014) Efficient MCMC for temporal epidemics via parameter reduction. Comput. Statist. Data Anal. 80. 240–250.

[12] Zuev, K. and Beck, J. (2014) Asymptotically Independent Markov Sampling: A New MCMC Scheme for Bayesian Inference. Vulnerability, Uncertainty, and Risk. 2022–2031. doi: 10.1061/9780784413609.203

A Appendix: R commands

The following commands in R generate dependent random values from the Fibonacci distribution, following the method of section 3. Commands are also given to generate Fibonacci numbers, Fibonacci probabilities, to plot true Fibonacci probabilities, to generate independent Fibonacci random variables. Commands are given for 3 plots (true, MCMC simulation approximation, inverse function simulation approximation).

```
par(mfrow=c(1,3))
#This sets up the frame for the combined 3 plots.

u=runif(1000001);
#This generates 1000001 random uniform (0,1) values.

F=rep(1,150)
for (i in 4:150) {F[i]=F[i-1]+F[i-2]}
#This generates the first 150 Fibonacci numbers

v=2:70; w=2:70
for (i in 1:69) {w[i]=F[i+1]/2^{i+1}}
# Vector w consists of first 70 Fibonacci probabilities.
plot(log(v),w,xlab="ln(x), x=2,...,70",ylab="True Fibonacci Probabilities",typ
#This plots the true probabilities.

x=rep(2,1000000);
```

```
#This is a vector with all values 2.
for (i in 1:1000000)
{a=(x[i]>2)*(u[i+1]<F[x[i]-1]/(F[x[i]]+2*F[x[i]-1]))
b=+1*(u[i+1]>(1-(.5*F[x[i]+1]/(F[x[i]+1]+2*F[x[i]]))))
x[i+1]=x[i]-a+b}
#This generates 1000000 values from the Fibonacci distribution.

y1=table(x); x1=as.numeric(rownames(y1))
#This table gives counts of the number of times each point is
generated using MCMC.
plot(log(x1),y1/1000000,xlab="ln(x), x=2,...,70",ylab="MCMC
Estimated Probabilities",type="h")
#This plots the relative frequency in log scale.

c=sample(v,1000000, replace=TRUE, prob=w)
y2=table(c);x2=as.numeric(rownames(y2))
#This table gives counts of the number of times each point is
generated using MCMC.
plot(log(x2),y2/1000000,xlab="ln(x), x=2,...,70",ylab="Inverse Function
Estimated Probabilities", type="h")
#This plots the relative frequency in log scale.
```