

AMATH 410

Review:

Implementing basic probability concepts in MATLAB

First, we discuss **generating samples (or realizations / trials) of a random variable** x . This means using a random number generator to produce a number that occurs with the frequencies (or probabilities) that are pre-specified for x . These probabilities can be specified in one of two ways.

- First, if x is a discrete-valued random variable, with N possible values (making up the state space, or sample space) $\{s_1, s_2, \dots, s_N\}$, then there is a list of N values $P(x = s_j) = p_j$ that specify the probabilities. Examples of this from class are the binary and binomial random variables.
- Second, if x is a continuous-valued random variable, there is a whole range of possible values that it could take, not just a discrete list. We'll just talk about one of this type of random variable here, the uniform-distributed random variable.

What we mean by “frequencies of occurrence” above is, if I generate the random variable over and over again M times for a huge number of *trials* M , on what fraction of those trials will x take a certain value? Let us say we count up a total of m_j times that the state s_j occurs. The fact that $p(x = s_j) = p_j$ means that $m_j/M \approx p_j$. That's it: p_j is the frequency (or fraction) of trials where x is found in a certain state (i.e., where x takes the j^{th} value, $x_j = s_j$).

Counting up the total number of times that anything at all happened in our trials above,

$$\sum_j m_j = M .$$

This means, by our definition,

$$\sum_j p_j = 1$$

That's the frequency (or fraction) of trials when anything at all (any possible state) occurred. **That is equal to 1.** That basic fact is known as **normalization**, but it's just logic – whenever we talk about the probability of anything happening, we'd better be able to sum up the probabilities of all of the different possibilities and get 1. If this does NOT work, then I need to fix the situation!

One place where this comes up is in Markov chains. I find the eigenvector \mathbf{w} with (dominant) eigenvalue 1. Let's say there are three possible states in my markov chain, so that

$$\mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix}$$

Now, the different w_j here are supposed to give us the probabilities p_j of being in states s_j (at “equilibrium”). They'd better add to 1! In general, they will NOT add to 1 when they are produced as entries of an eigenvector, coming straight out of MATLAB. What do I

do? MAKE them add to one, by dividing each by the sum of the rest. The corresponding probabilities are then

$$\begin{aligned}p_1 &= w_1/(w_1 + w_2 + w_3) \\p_2 &= w_2/(w_1 + w_2 + w_3) \\p_3 &= w_3/(w_1 + w_2 + w_3)\end{aligned}$$

This certainly gives us $p_1 + p_2 + p_3 = 1$, so these now make sense as probabilities.

The rand command

The uniform-distributed random variable with range $[0, 1]$ is the basis for most of our codes involving probability models in the course. We START with it, and generate samples of binary or binomial random variables. Or, we START with it, and determine into which state a markov chain should transition at the next timestep k .

The uniform-distributed random variable with range $[0, 1]$ takes values between 0 and 1 – any value there, with equal probability. It is a continuous-valued random variable. Unlike for a discrete-valued random variable, it's not very useful to assign a probability to a specific value of the uniform-distributed random variable... since that requires x being equal to this value with perfect precision. E.g. the fraction of trials when, say, $x = .341741902437192347...$ is extremely small – actually zero, if we define what x is supposed to be with perfect precision (endless digits). So, instead we specify the probability that x lies anywhere in subrange $[a, b]$. If $0 \leq a \leq b \leq 1$, then we have

$$P(x \in [a, b]) = b - a \tag{1}$$

To generate a single realization of such a uniformly distributed random variable with range $[0, 1]$, we type `x=rand`. If we want a list of M samples, we type `x=rand(1,M)`. If we want a $M \times M$ matrix of samples, , we type `x=rand(M,M)`.

The rand command and binary-valued random variables

Say we want to use these samples to make a list of samples of a binary-valued random variable x , with $P(x = 1) = H$ and $P(x = 0) = 1 - H$. I start with a uniform random variable – this time, I'll call this r . Then, I will use equation (1). This gives us the fact $P(r \in [0, H]) = H$. So, if I set $x = 1$ every time that $r \in [0, H]$ and $x = 0$ otherwise, x will be the desired binary-valued random variable! Specifically:

```
r=rand
if r<H
x=1
else
x=0
end
```

does the trick.

Now, say I want to make not just one sample of a binary-valued random variable, but many, say M of them, all in a list `xlist`. Here's the code for that (see the HW solution code that solves Ex. 3.3):

```

rlist=rand(1,M)

xlist=zeros(1,M)
%we just declared xlist as a "dummy" or "blank" list --
%we're about to fill it in with real values

for j=1:M
    if rlist(j)<H
        xlist(j)=1
    else
        xlist(j)=0
    end
end

```

There is also faster, but absolutely equivalent, way of doing this, also explained in the comment-annotated HW solution codes:

```

rlist=rand(1,M)
xlist=zeros(1,M)
xlist(find(rlist<H))=1

```

The rand command and simulating Markov chains

Here we want to use `rand` to do something different but very closely related. This is also explained in detail in the comments in the HW solution code `EandG_exercise_3_6.m`.

The situation we'll discuss is that our Markov chain is currently in state 3 at timestep k . That is, $x(k) = 3$. I want to advance to the next timestep. That means setting $x(k+1)$. I am going to do this based on the transition probabilities that belong to the j^{th} column of the matrix A . Let us say that there are 3 states, so that A is a 3×3 matrix. Then the third column is

$$\begin{matrix} a_{13} \\ a_{23} \\ a_{33} \end{matrix}$$

Here, a_{13} is the probability that I will transition from state 3 to state 1, a_{23} is the probability that I will transition from state 3 to state 2, a_{33} is the probability that I will stay in state 3.

OK, so which of these three possibilities happens on a given trial? Clearly, I need to generate a random variable $x(k+1)$ – the state on the next timestep – such that

- $x(k+1) = 1$ with probability a_{13}
- $x(k+1) = 2$ with probability a_{23}
- $x(k+1) = 3$ with probability a_{33} .

I will use `rand` to do this. Just like for the binary valued random variable above, we first set `r=rand`. Next, I use the facts $P(r \in [0, a_{13}] = a_{13})$, $P(r \in [a_{13}, a_{13} + a_{23}] = a_{23})$,

$P(r \in [a_{23}, a_{23} + a_{33}] = a_{33})$, exactly as above. So, splitting up the interval $[0, 1]$ in that way gives me the probabilities that I was after!

Well, I'm now going to use this to define the value of the state on the next timestep, $x(k+1)$. Here's the setup, exactly as in `EandG_exercise_3_6.m`. I have a list called `states` that is we are gradually filling in with the values $x(k)$. Say I've already simulated up to timestep k . Then $x(k) = \text{states}(k)$ and we are going to set `states(k+1)=x(k + 1)`. Let's do it. Here we are ASSUMING that `states(k)=3`, like the above. The code `EandG_exercise_3_6.m` handles the more general case, where we don't prespecify the value of `states(k)` (that's a bit artificial, but simplest).

```
r=rand ;
if r < a13 %for transition FROM state 3 to state 1
    states(k+1)=1;
elseif r < a13+a23 %for transition FROM state 3 to state 2
    states(k+1)=2;
else
    states(k+1)=3; %for transition FROM state 3 to state 3
end
```

That's it! Let me repeat what we'd do in general (without the artificial "prespecifying" just discussed).

```
if r < A(1,states(k)) %for transition FROM states(k) to state 1
    states(k+1)=1;
elseif r < A(1,states(k))+A(2,states(k)) %for transition FROM states(k) to state 2
    states(k+1)=2;
else
    states(k+1)=3; %for transition FROM states(k) to state 3
end
```