Markov Chains

Suppose a machine at any one time can be in one of finitely many states, 1, 2, 3, ..., n. Over a discrete interval of time (say 1 minute), it may switch from one state to another. It is a Markov Chain if it for any two states \( i, j \), there is a fixed transition probability \( \text{trans}(i,j) \) that the machine will transition from \( i \) to \( j \). The Markov Chain graph has a node for each state and the edges are labeled with the transition probabilities.

For an example of a Markov Chain, see

http://setosa.io/blog/2014/07/26/markov-chains/
In the game “Snakes and Ladders” or “Chutes and Ladders” one moves forward one or more places depending on the roll of a die. In our short version, we’ll flip a coin. Landing on the base of a ladder takes one up to the top. Landing on the head of a snake (or chute) takes one down to the end of the snake. A coin is tossed. On tails you make one
move to the next higher-numbered square; on heads you make two moves (unless landing on a ladder base or snake head).

Draw the Markov chain graph.

Implicit none turns off implicit declaration. Use this to find spelling errors.

**CLASSWORK 29.1(6) snakes_ladders.f95** Calculate the \(9 \times 9\) transition matrix \(\text{trans}\) for the Snakes and Ladders game. Calculate the average number of steps to finish the game.

```fortran
!c29_1_6snakes_ladders.f95

integer function rand_prob(prob,n)
real::prob(n),r,p
r=rand(); p=0
do i=1,n
  if(r<p+prob(i))then;rand_prob=i;return;endif
  p=p+prob(i)
endo
endfunction

subroutine play_game(trans,n,history,nummoves)
!~ implicit none
real::trans(n-1,n)
integer,parameter::inf=10**5
integer::nruns,rand_prob,nummoves,history(200)
  history=0;nummoves=0;i=1;history(1)=1
do
```
... use rand_prob(trans(i,:),9) to get the next i
... update nummoves, history(nummoves+1)
... include the exit condition

enddo
endsubroutine

program snakes_ladders_individual
!~ implicit none
real::trans(8,9),average
integer,parameter::inf=10**5
integer::nruns,rnd_prob,nummoves,history(200)
! 1, 2, ..., 9 = board positions, trans = matrix of transition probs.
trans=0
trans(1,7)=.5; trans(1,5)=.5
trans(2,5)=.5; trans(2,4)=.5
trans(3,4)=.5; trans(3,5)=.5

... cases for 4,5,7

print'(9f6.2)',(trans(i,:),i=1,8)
do nruns=1,inf
   call play_game(trans,9,history,nummoves)
   average=(average*(nruns-1)+real(nummoves))/nruns
   if(mod(nruns,1000)!=1)cycle
   print 11,nummoves,average,history(1:nummoves+1)
enddo
11 format('#moves=',i2,3x,'average=',f5.2,200(i3))
endprogram
**Markov predictions**

Markov chains can be used to predict transitions from one state to the next. By keeping a record of the sequence of states, we can calculate the probabilities of the next state. We calculate the probabilities just as we calculate a running average. Suppose the sequence of five coin tosses is 0, 0, 1, 1, 1 where 0 is tails and 1 is heads. Then the probability of heads in this sequence is 3/5. This is also the average of the sequence: \( \frac{0 + 0 + 1 + 1 + 1}{5} \). Hence we can use the formula for the running average: \( \frac{A_{n-1}(n - 1) + a_n}{n} \). Instead of keeping an average of the whole sequence of length \( n \), it is often better to keep an average of say just the last 10. So the formula which updates on the last 10 will be \( \frac{A_{n-1} \times 9 + a_n}{10} \).

The optimal strategy for playing rock-paper-scissors against an intelligent player is to randomly choose the next move with all being equally likely, i.e., each having probability 1/3. But suppose we are playing against someone whose moves are not equally likely. By keeping a running average of his moves, we can build a Markov chain which
shows for each move (rock, paper, scissors), the probability of his next move. If the user tends to cycle through the three states, rock, paper, scissors, rock, paper, scissors, ..., then the Markov chain might be

\[
\begin{bmatrix}
0.1 & 0.8 & 0.1 \\
0.1 & 0.1 & 0.8 \\
0.1 & 0.1 & 0.1
\end{bmatrix}
\]

Then, once we find that when the player plays 1 (rock), his next play will probably be 2 (paper), we can take advantage of this and choose a play which will beat 2. Let \text{trans} be the transition matrix, then \text{trans}(1,:) = [0.1, 0.8, 0.1]. Use \text{trans_prob} to predict the next move based on these
transition probabilities: \textit{predicted} = \textit{trans} \_\textit{prob} (\textit{trans})(1,:),3). Since 2 (paper) has the highest probability (.8) it will be predicted most often. We want our move to beat the predicted move. Let \textit{beat}=[2,3,1]. Then \textit{beat}(i) = the move which beats i. For 1=rock, \textit{beat}(1)=2=paper which beats rock. \textit{beat}(2)=3 since scissors=3 beats paper=2. Hence our move should be \textit{mymove}=\textit{beat}(\textit{predicted}).

\begin{verbatim}
!rockpaperscissiors_advanced.f95
integer function rand_prob(prob,n)
real::prob(n),r,p
! picks numbers from 1, ..., n, i has probability prob(i)
r=rand(); p=0
do i=1,n
  if(p<r .and. r<p+prob(i) )then
    rand_prob=i;return
  endif
  p=p+prob(i)
enddo
endfunction
endfunction
program rockpaperscissors
integer::youmove,mymove,prevmove=1,predictedmove!
ymymove = computers move, youmove = user's move
integer::win,winner(3,3),beat(3)=(/2,3,1/)
\end{verbatim}
integer::moves(3,3),mywins=0,youwins=0,rand_prob
integer::rock=1,paper=2,scissors=3,draw=0
real::trans(3,3)=1./3
character(8)::move(3)
moves(1,:)=[1,0,0];
moves(2,:)=[0,1,0];
moves(3,:)=[0,0,1];
winner(rock,:)=[draw,paper,rock]
winner(paper,:)=[paper,draw,scissors]
winner(scissors,:)=[rock,scissors,draw]
Print*,'Enter 0=quit, 1=rock, 2=paper, 3=scissors'
6 format(a,i1,1x,a8,i3,a)
do
    predictedmove=rand_prob(trans(prevmove,:),3)
    mymove = beat(predictedmove)
    read*,youmove;
    if(youmove==0) exit;if(youmove>3)cycle
    win=winner(mymove,youmove)
    if(win==0)then; Print*,'Draw';endif
    if(win==youmove)then;
        youwins=youwins+1; print*,'You win.';
    endif
    if(win==mymove)then;
        mywins=mywins+1; print*,'I win.';
    endif
endif
print 6,'you:', youmove, move(youmove), youwins," wins for you"
print 6,' me:', mymove, move(mymove), mywins," wins for me"
trans(prevmove,:)= &
&(trans(prevmove,:)*9+moves(youmove,:))/10
prevmove=youmove
enddo
endprogram

EQUILIBRIUM DISTRIBUTIONS (STEADY STATE)
Initially, a company has 100 working machines
Over the course of a year,
75% of the working machines continue to work;
25% of the working machines break.
60% of the broken machine will be repaired;
40% of the broken machines remain broken.

Let 1 be the working state, let 2 be the broken state. Draw the Markov Chain graph. The initial distribution is dist=[100, 0]. The one-year-to-the-next transition matrix is
trans= [ .75 .25
       .60 .40 ]. After one year, the initial [100,0] distribution becomes [75,25]. Note that this is dist×trans. After 0, 1, 2, 3, ... years, the distributions are dist,
The eventual equilibrium or steady-state distribution is the infinite limit of these distributions. Populations tend to settle down to their equilibrium distributions rapidly and matrix multiplication is slow so we approximate infinity here with only 20 generations.

**Classwork 28.2(4) steady_state.f95**

Calculate equilibrium distribution for the number of working machines and broken machines.
program markov_equilibrium
real :: trans(2,2), dist(2), equilib(2)
integer, parameter :: inf = 20
! position 1 = working, 2 = broken
dist(:, :) = (/ 1 1 /) ! initial distribution
! working, broken
trans(1, :) = (/ 1 1 /)
trans(2, :) = (/ 1 1 /)
11 format(2f7.2)
print 11, (trans(i, :) , i = 1, 2); print*
equilib = 
! matmul is matrix multiplication
do i = 1, inf
   print 11, equilib(:)
   equilib = 
enddo
endprogram

CLASSWORK  29.1(6) snakes_ladders  28.2(4) steady_state
email: dale@math.hawaii.edu  subject line: 190  c29(10)