Markov Chains

A Markov Chain is a machine/game/process which can be in one of \( n \) states, 1, 2, 3, ..., \( n \). It sequentially switches from one state to another. For any two states \( i, j \), there is a fixed transition probability \( \text{trans}(i,j) \) of transitioning from \( i \) to \( j \). The Markov Graph has a node for each state and edges labeled with the transition probabilities; \( \text{trans} \) is the Markov matrix.

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 toss a coin. On tails you make one move to the next higher-numbered square; on heads you make two moves. But if you land at the base of a ladder, you climb up to its top; if you land on the head of a snake, you slide down to its tail.

The Markov graph and matrix. Since you can’t stay at 2, 3, 6, 8 you only need nodes 1, 4, 5, 7, 9.

\[
\begin{align*}
\text{trans}(1,7) &= .5, \\
\text{trans}(1,5) &= .5, \\
\text{trans}(4,5) &= .5, \\
\text{trans}(4,1) &= .5, \\
\text{trans}(5,1) &= .5, \\
\text{trans}(5,7) &= .5, \\
\text{trans}(7,4) &= .5, \\
\text{trans}(7,9) &= .5
\end{align*}
\]

Classwork 29.1(5) snakes_ladders.f95 Write a function \( \text{next_state} \) which models randomly moving a state \( j \) to a next state according to the transition probabilities \( \text{trans} \). Use it to find the average number of steps to finish the game.
The only unbeatable rock-paper-scissors strategy is to randomly choose the next move, with equal probabilities. But suppose your moves are not equally likely. By keeping a running average of your moves, the computer (it will list itself as me) can build a Markov chain trans which find for each previous move (rock, paper, scissors), the probability of your next move. If your tend to simply cycle through the three states, rock, paper, scissors, the Markov graph (trans) might be:

```
1r .1 .1 .1
  |     |     |
  |     |     |
  .8 .8 .8
```

In this case, after playing 3 (scissors) the you usually play 1 (rock), the computer (self-listed as me) can take advantage of this and choose to play 2 (paper) to beat you. The payoff_matrix lists the computer’s payoffs for each pair of choices you and it makes. 1 if the computer wins, -1 it is loses, 0 for a draw. For your previous choice yourprevmove, the corresponding row trans(yourprevmove,:) of the transition matrix...
trans give the probabilities of your next choice. This row multiplied by payoff_matrix gives the predicted payoffs predicted_payoffs for each of the computer’s possible choices. The computer picks the max_loc choice mymove with the highest payoff.

C\textsc{lasswork} 29.2(5) rockpaperscissors.f95
!c29_2_5rockpaperscissors.f95
program rockpaperscissors
implicit none
integer:: yourmove, mymove, yourprevmove=1
!yourmove = user move mymove = computer move,
integer:: payoff_matrix(3,3) !computer's payoff
integer:: rock=1, paper=2, scissors=3, draw=0
character(8):: move(3)=&
  (/'rock    ','paper   ','scissors'/)
integer:: mywins=0, yourwins=0, payoff !computer
real:: trans (3,3)=1./3, predicted_payoffs (3)
!your choice events [rock, paper, scissors]
event (1,:)=(/1,0,0/)
event (2,:)=(/0,1,0/)
event (3,:)=(/0,0,1/)
!my payoff_matrix(your move, my move)

payoff_matrix (rock, rock) = 0
payoff_matrix (rock, paper) = 1
payoff_matrix (rock, scissors) = -1
payoff_matrix (paper, rock) = -1
payoff_matrix (paper, paper) = 0
payoff_matrix (paper, scissors) = 1
payoff_matrix (scissors, rock) = 1
payoff_matrix (scissors, paper) = -1

print*, 'Enter 0=quit, 1=rock, 2=paper, 3=scissors'
do
  ! predicted_payoffs, mymove
  ! your move, 0, >3
  ! payoff, payoff cases
  print 6, 'you:', yourmove, move(yourmove), &
  yourwins, ' wins for you'
  print 6, ' me:', mymove, move(mymove), &
  mywins, ' wins for Markov'
  ! update trans
  ! yourprevmove
enddo
6 format (a,i1,1x,a8,i3,a)
endprogram

\textbf{Three points extra credit.} Find a sequence of 30 moves which beats this machine learning program two-to-one, i.e., wins at least 20 times against the computer’s 10 or less wins. Write moves using 1-2-3 rather than rock-paper-scissors, e.g., 332112... 

\textbf{Optimal stopping time}

The optimal strategy for buying and selling stocks and bonds is to buy low and sell high. What is the minimum height at which you should sell? Call this height the minimum acceptable amount min_accept. Here is a gambling version of this problem where the player chooses a number between 1 and 5 as the minimum acceptable amount.
A die is rolled. If a 6 comes up, the game stops and he gets $0. If the amount is \( \geq \min\_{\text{accept}} \), he collects in dollars the amount on the die and the game stops. Otherwise the game continues for another round. Here are the games for the possible \( \min\_{\text{accept}} \) amounts:

- Stop when he gets \( \geq 1 \) (hence he stops after the first roll).
- Stop when he gets \( \geq 2 \) (\( \min\_{\text{accept}} \) is 2)
- Stop when he gets \( \geq 3 \) (\( \min\_{\text{accept}} \) is 3)
- Stop when he gets \( \geq 4 \)
- Stop when he gets \( \geq 5 \)

Waiting until he gets a 6 gets him nothing.

What \( \min\_{\text{accept}} \) amount gives the most money?

**CLASSWORK 29.3(5) optimal_stopping.f95**

One play consists of rolling the die until the player wins or loses. 

- \( \text{dollars\_won} \) = the amount the player wins in one play.
- \( \min\_{\text{accept}} \) = the minimum number the player will accept.
- \( \min\_{\text{accept}} = 3 \) means he will not stop until he gets \( \geq 3 \) or loses because of rolling a 6.

We play a large number of times and keep a running average of the dollars won in each play.

```fortran
! roll the die until the player wins or loses
! calculate the dollars\_won
average=(average*(n-1)+dollars\_won)/n
enddo
endfunction
program optimal_stopping_amount
real::average, integer::min\_accept
do min\_accept=1,5
  2 format(a,i2,a,f5.2)
print 2,'If min accepted =',min\_accept,' average winnings =',average(min\_accept)
endoendprogram
integer function randint(m,n)
call random_number(r)
randint = floor(m+(n-m+1)*r)
endfunction
```

**CLASSWORK 29.1(5) snakes_ladders 29.2(5)**

dollars\_won = the amount the player wins in one play.

- \( \min\_{\text{accept}} \) = the minimum number the player will accept.
- \( \min\_{\text{accept}}=3 \) means he will not stop until he gets \( \geq 3 \) or loses because of rolling a 6.

We play a large number of times and keep a running average of the dollars won in each play.

```fortran
!c29_3_5optimal_stopping.f95
real function average(min\_accept)
integer::min\_accept,dollars\_won,die,randint
integer,parameter::inf=10**4
average=0
do n=1,inf
! roll the die until the player wins or loses
! calculate the dollars\_won
average=(average*(n-1)+dollars\_won)/n
enddo
endfunction
```

**CLASSWORK 29.1(5) snakes_ladders 29.2(5)**

- \( \text{rockpaperscissors} \) 29.3(5) optimal_stopping
- email: dale@math.hawaii.edu  subject line: 190 c29(15)