PseudoCode

Dot Product:

- input: 2 vectors, u & v
- output: one scalar, dotProduct
- take transpose of one vector:
  $u = u'$
- need to initialize our dotProduct to be zero so we can sum in for loop
- for i from 1 to the length of the vectors
    multiply i^{th} element of each vector (u-across v-down)
    add it to dotProduct
  end

Matrix-Vector Multiplication

- input: 1 matrix A, 1 vector x
- output: 1 vector, prod
- use dot product code for each element of our vector prod
- for i from 1 to the # of rows of A (size(A,1))
    to get i^{th} element of prod, call perform_Dot_Product with i^{th} column of A & vector x
  end

Matrix-Matrix Multiplication

- input: 2 matrices, A & B
- output: 1 matrix, AB
- we will use dot product for each element of our matrix

Matrix-Matrix Mult Cont

- for i from 1 to # of rows in A
  (size(A,1)) → will be # of rows in AB)
  for j from 1 to # of cols in B
  (size(B,2)) → will be # of cols in AB
  element (i,j) of AB will call perform_Dot_Product of i^{th} row of A and j^{th} col of B.
  end
end

Chutes & ladders

- Split into 2 codes:
  1) play the game 1 time & output the # of turns it took
  2) calls 1st code in a for loop the # of times we want to play, save the counts in a vector, plot probability distribution

1) Play the Game

- no inputs
- output: # of turns it took, ct
- initialize count to be zero (before we start)
- need to keep track of what space we are on, call variable space
  ↳ initalize to be zero
- keep track of chutes & ladders in 2 matrices:

Play Game cont.

- both are 9×2 matrices, 1st col are where the chutes/ladders start, 2nd col are where they end
- while loop to play until space = 100
  - roll # between 1 & 6 → use rand function
  - add roll to space so we move forward
  - check if we landed on chute/ladder ⇒ if, elseif statement!, also must be in for loop b/c we have to check all 9 of each
    - for i from 1-9
      - if space is a chute, change space to corresponding 2nd col element & break loop
      - elseif space is a ladder, change space1 to corresponding 2nd col element & break loop
    - end
  - for each iteration, we are taking a turn, so add to the count!
  - check if we went past 100, then we have to go back to space before we rolled

end

2) Probability function

- no inputs
- no outputs (plot distribution later)
- set N to be the # of times we play the game
- create a vector to store the frequency of each count (# of turns it takes), call it ctVec & initialize it as a zero vector. Make it long (size should be greater than the max # of turns a game will take
- for i from 1 to N
  - play the game! call the function & store the # of turns it took as a scalar, playGame
  - add 1 to the ctVec location that matches the # of turns it took (ctVec(playGame, 1))

  end
- find the probability: $\frac{ctVec}{N}$ ← (# of games played)
- create a vector to store the # of turns: 1:1:length(ctVec)
- plot distribution:
  - x = # of turns vector
  - y = probability vector
- add titles, x & y labels, change linewidth & markersize

★ For No Chutes & No Ladders, codes are the same except for:

- in playing-the-game code, remove chute or ladder vector & corresponding if/else if statement
- in probability function, change code that is called to play the game & some parameters (N, size of ctVec)