**ENSC 180 – Introduction to Engineering Analysis Tools**

**Assignment #3: Texas Hold'em**

## Background

Texas hold ‘em is one of the most popular variations of poker. At the start, each player is dealt two cards and, as the game progresses, an additional five community cards are dealt face-up. A player’s best 5-card hand is formed from the cards he is dealt along with the community cards. Due to the large number of different 5-card hands and the increasing complexity as more cards are dealt, we will only focus on the pre-flop portion of the game, i.e., before the community cards are dealt. This way, each player has two cards and we only need to be familiar with the following 2-card hands: *one pair* and *high card*. Hence, given a standard 52-card deck with 13 ranks,

2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King, Ace (listed from lowest rank to highest),

and 4 suits,

Spades (♠), Clubs (♣), Hearts (♥), Diamonds (♦),

a pair is any 2-card hand where both cards are of equal rank. For our 2-card case, a high card is any non-pair hand. A pair is higher in rank than a high card. The strength of a pair is determined by the rank of the cards that compose that pair, for example Q♠Q♥ is better than 8♠8♥. Similarly, the strength of a high card hand is determined by the highest ranked card in that hand, for example J♦2♣ is better than 10♦9♣. It then follows that an ace high card (e.g., A♥7♠) is the strongest high card hand and can only be beaten by a pair.

## Objectives

In this assignment, we will use combinatorics to count and analyze poker hands in the pre-flop stage of a Texas hold ‘em game of poker. To simplify our analysis, we will only have two players in the game. Player 1’s cards will be specified by the user. Then, the objective is to compute the probabilities of different hands for Player 2 based on the cards dealt to P1. These probabilities will be outputted in the form of a bar plot.

## Procedure

You are given a file “Texas\_HoldEm.m” with the following function inside of it:

function Texas\_HoldEm(card1, card2)

The inputs card1 and card2 are strings that specify P1’s starting (pre-flop) hand with the following format: '*<card rank><suit>*'. For example, to specify a starting hand of J♦2♣, call the function as:

>> Texas\_HoldEm('JD', '2C')

or

>> Texas\_HoldEm('2C', 'JD')

The function will parse these strings for you and format the chosen cards into a binary 4-by-13 matrix called CardMatrix:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 'A' | 'K' | 'Q' | 'J' | '10' | '9' | '8' | '7' | '6' | '5' | '4' | '3' | '2' |
| 'S' (♠) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 'D' (♦) | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 'H' (♥) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 'C' (♣) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Note that there are 52 entries corresponding to the 52 different cards in the deck. If an entry contains a 1, then P1 has the card corresponding to that entry; otherwise, it contains a 0. In the example above, P1 has been dealt J♦2♣.

If we don’t care about suits, we can collapse this matrix into an array by summing along the rows:

sum(CardMatrix)

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 'A' | 'K' | 'Q' | 'J' | '10' | '9' | '8' | '7' | '6' | '5' | '4' | '3' | '2' |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

If we don’t care about card ranks, we can sum along the columns:

sum(CardMatrix, 2)

|  |  |
| --- | --- |
| 'S' (♠) | 0 |
| 'D' (♦) | 1 |
| 'H' (♥) | 0 |
| 'C' (♣) | 1 |

This sort of formatting will allow us to count the various different hands that P2 can make based on P1’s cards in an efficient manner, without the use of excessive conditional statements. (In fact, the implementation in the following tasks can easily be extended to more players when this sort of organization is used). Also note that since the order of the cards does not matter, only combinations along with the addition and multiplication principles are needed throughout this assignment (no permutations necessary).

**Task 1:** Compute the probability that P2 has a pair. As a warm-up, consider the following specific situation: P1 is dealt J♦2♣. Then, the total number of pairs that P2 can make in this situation is:

If P2 has a pair of Jacks, it must be formed from the 3 Jacks left in the deck since P1 already has a J♦. The same logic applies for a pair of 2♣. For the remaining card ranks, there are 4 available suits in each.

To generalize this calculation for any hand that P1 might have:

1. Create a variable numPairs to store the number of one pair hands that P2 can make. Initialize numPairs to 0.
2. Collapse CardMatrix as previously shown and iterate through each card rank using a for-loop.
3. For each card rank, count the number of possible pairs that P2 can make from a particular card rank using the nchoosek function and add the result to numPairs.
4. After counting the number of all possible pairs, divide this amount by the total amount of 2-card hands that P2 can make to find the probability that P2 has a pair, i.e.:

Note that at this point, 2 cards have been removed from the deck. Store the probability in a variable probPair.

**Task 2:** Compute the probability that P2 has an Ace high hand (i.e., a hand that contains an Ace and a non-Ace). Just like in Task 1, utilize CardMatrix along with the nchoosek function to count the number of different Ace high hands that P2 can form in a variable numAceHigh. Divide numAceHigh by the total number of hands P2 can make to find the desired probability and store it in a variable probAceHigh.

**Task 3:** Compute the probability that P2 has two cards of the same suit. This is very similar to Task 1, however this time we don’t care about card ranks and so CardMatrix needs to be collapsed differently. Store this probability in a variable probSuited.

**Task 4:** Compute the probability that P2’s hand is better than P1’s hand. This task can be decomposed into two different problems using an if-statement as follows:

1. If P1 has a pair, then P2 must have a higher-ranked pair.
   1. Collapse CardMatrix using the sum function as before in a manner relevant to this analysis.
   2. The find function can be used on the collapsed CardMatrix to determine the rank of the pair.
   3. Proceed as in Task 1, but this time only count the pairs that P2 can make which are of higher rank than P1’s.
   4. Divide the result by the total number of hands P2 can make and store the result in a variable probBetter.
2. If P1 does not have a pair, then P2 must have either a pair or a higher-ranked high card.
   1. In this case the number of possible pairs is already determined in Task 1.
   2. In addition to this, count the number of higher-ranked high card hands that P2 can have. Remember, we’ve already counted pairs in step 2.a.; be very careful not to over-count.
   3. Divide the sum of all possible pairs and higher-ranked high card hands by the total number of hands P2 can make and store the result in a variable probBetter.

**Task 5:** Display the probabilities calculated in Tasks 1 – 4 using a horizontal bar plot as illustrated below.

D:\SFU\Work & Co-op\ENSC180 - Course development\Assignments\Combinatorics\Texas_HoldEm\untitled.emf

Proceed as follows:

1. Define a vector   
   prob = [probBetter, probSuited, probAceHigh, probPair].
2. Use the function barh to plot the vector prob.
3. Use the following code to add the percentages onto the plot. Try to understand what it does. See the MATLAB documentation for the function text.

|  |
| --- |
| for ii = 1:length(prob)  text(prob(ii) + 0.005, ii, sprintf('%.2f%%', prob(ii)\*100), ...  'FontSize', 12)  end |

1. Add a title and x-axis label as shown above with a font size of 15 for both.
2. Add labels for the y-axis ticks by modifying the YTickLabel property of the axis object using the set function.
3. Set the font size of the axis object to 15 using the set function.

Test your code with a variety of inputs for card1 and card2. Ensure that all probabilities are within the range [0,1]. Also make sure that they are sensible.