Introduction to C++: Assessment 1 - Week 5, 2019

Instructions for using each file can be found in README.md which is found in the same directory as this. The specific data used in generating this report can be found in the *raw_data* subdirectory.

Question 1: Using Random Numbers

This program (random.cpp) takes in an integer input (or asks for one in the absence of an input), generates that many random floating point numbers between 0 and 1, stores them in a vector and then uses them to calculate the mean and variance of the generated numbers.

To calculate the mean, the vector storing them is iterated through and summed into a variable, which is then divided by the original input to obtain the mean.

For the variance, the vector is again iterated through, this time calculating the squared difference from the mean and storing in a variable, which again is divided by the original input to obtain the variance.

The program then outputs the original input, the variance and the mean to stdout as comma separated values.

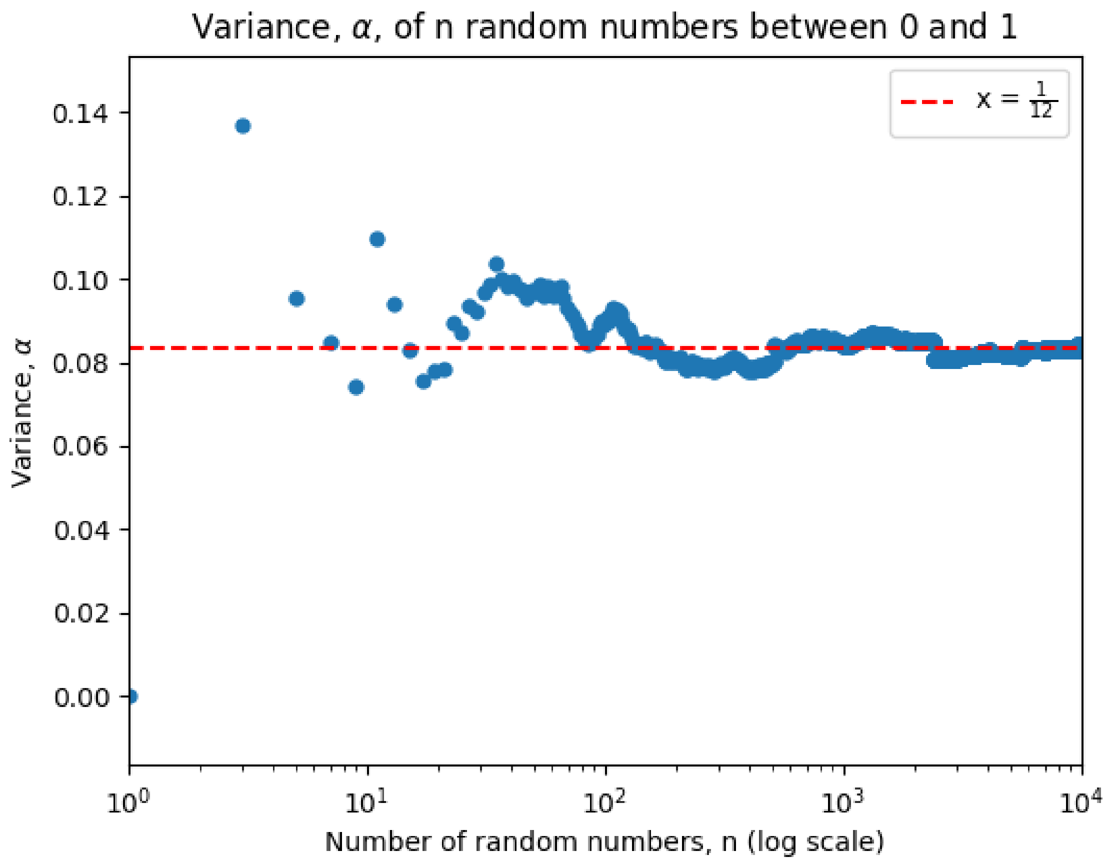

Figure 1. Empirical calculation of the variance

Raw data used in this graph can be found in random_used.csv

To empirically calculate the limit of the variance as the number of simulated numbers grows, this program was run with inputs of every odd number below 10000, piped into a .csv file and plotted using python, which resulted in Figure 1 above; the data in this graph clearly points to a convergence value of $\frac{1}{12}$ as the input value grows.

Question 2: Simulating Dice Throws

This program (dice.cpp) takes an integer input and returns the aggregated results of that many throws of a pair of simulated six-sided dice.

To accomplish this it first checks to ensure the input is an integer and initialises a vector big enough to count all possible results. After this a random floating point number between 0 and 1 is generated, and then mapped this to an integer between 1 and 6 inclusive, before repeating this process and summing the two values; the corresponding counter in the aforementioned vector is then incremented. This process is repeated N times, where N is the initial integer input.

A string for the filename is then created according to the specification supplied by the problem sheet, and stdout is then redirected to this file. Finally the vector is iterated through and the results (each sum and the number of times that sum was generated) are then outputted to stdout in the format specified.

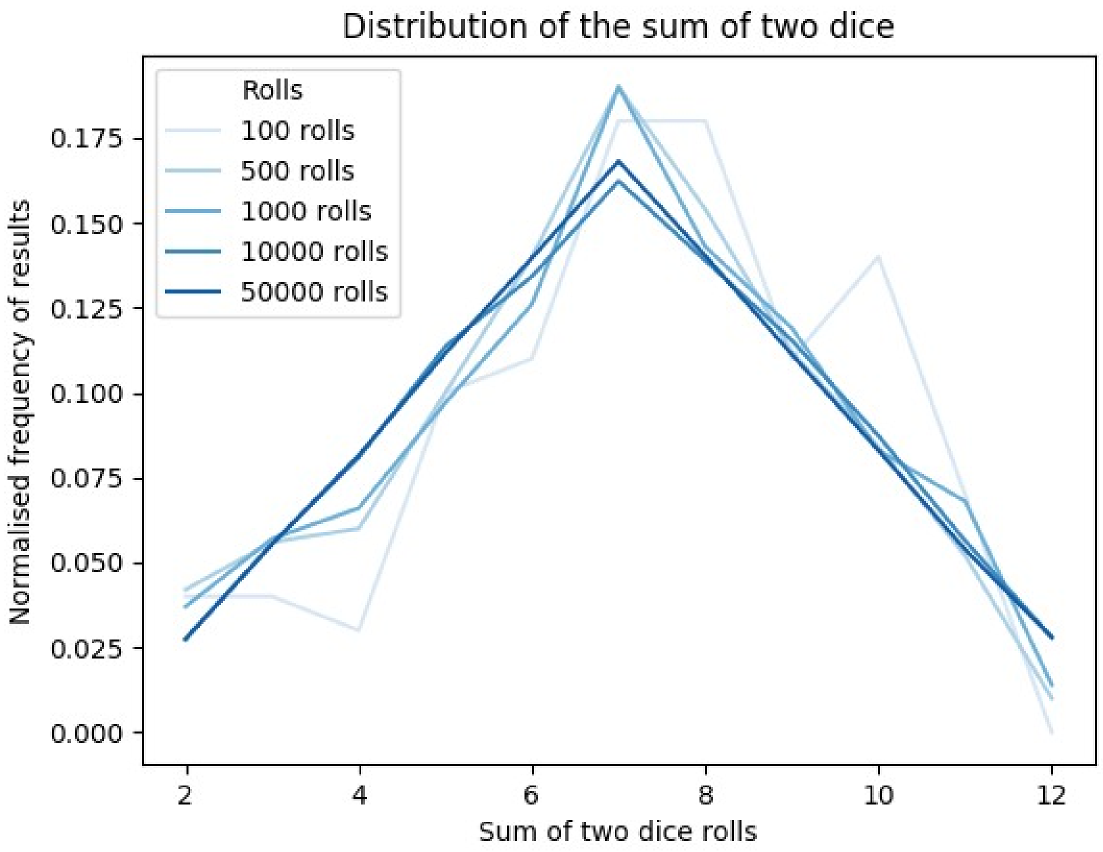

Figure 2. Normalised distribution of the sum of a pair of dice rolls with varying numbers of trials.

Raw data used to create this graph can be found in dice_N_used.dat

Results were then obtained for the specified number of trials, and plotted as seen in Figure 2 above. This figure shows that as the number of trials increases towards infinity, the distribution of results converges towards a symmetric triangular distribution centred around the most common result, 7. This raw data is now below in Figure 3.

Sum	2	3	4	5	6	7	8	9	10	11	12
No. Trials											
100	4	4	3	10	11	18	18	11	14	7	0
500	21	28	30	50	70	95	77	56	42	26	5
1000	37	57	66	97	126	190	143	119	83	68	14
10000	275	555	809	1136	1342	1623	1389	1154	873	564	280
50000	1370	2769	4077	5580	6982	8407	7016	5561	4149	2691	1398

Figure 3. Table of results for dice.cpp, the sum total of two simulated dice and the frequency of that score given a certain number of trials

Raw data used to create this table can be found in dice_N_used.dat

Question 3: The Game of Craps

This program (craps.cpp) takes an integer input and simulates that many games of craps, storing the result (win/loss) as well as how many pairs of dice throws it took to achieve that result. This is outputted to stdout as comma separated variables with: count of games reaching this result with that many rolls, number of rolls of the game, result of the game (Win/Loss) on each line.

To do that, the input is checked to be a positive integer and two vectors of counters are initialised: one to store losses and the other to store wins. The program then enters a for-loop over the number of games requested by the input; first initialising/resetting variables used to simulate the game, before entering a while-loop to simulate the game. Within this while-loop, the rolling of two dice is simulated as above, and the value is assigned to a variable while the roll counter is incremented. This value is then checked using the first roll specific conditions to see if it wins or loses craps; if it does, the while-loop is exited; if it doesn't, it saves the value to another variable initialised earlier, then the while-loop iterates and dice are rolled again, roll counter is incremented and the new value is checked against the non-first roll conditions, and this repeats until a victory or loss is achieved and the while-loop exited.

When a result has been found, the vector corresponding to that result is checked to have enough elements (if not, the vector has new elements added until it is), and the counter at the element corresponding to the number of rolls the game took is incremented (i.e. if the game took 7 rolls, the counter at the 7th element in the vector is incremented by 1).

If there are more games due to be simulated, the for-loop is iterated and variables are then re-initialised and another game is "played"; if not, the results are outputted to stdout by iterating through each vector.

This program was then run for 10,000 games of craps as specified, and the results were piped and saved to a .csv file. This was then analysed and plotted using python, resulting in the figures below.

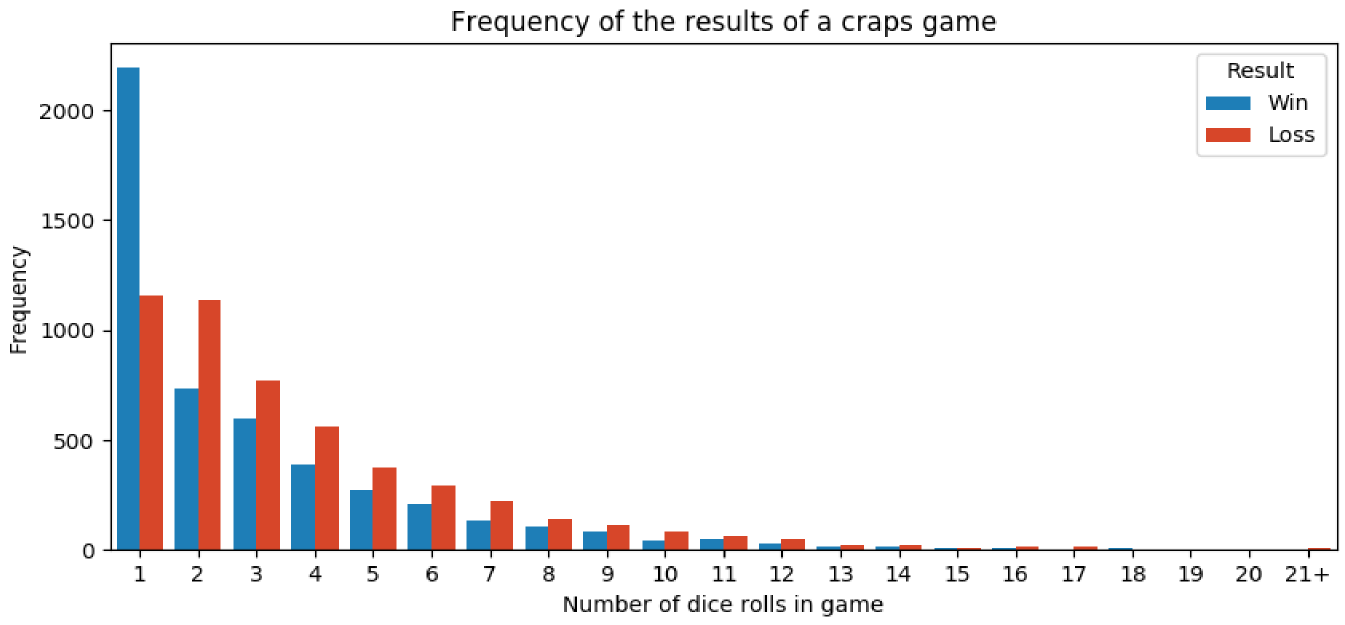

Figure 4. Graphical representation of the spread of the length and result of 10,000 simulated craps games

Raw data used to create this graph can be found in craps_used.csv

Both Figures 4, above, and 5, below, show the relationship between length of the simulated game of craps and it's result. Games after 20 rolls are grouped due to their relative scarcity.

Rolls	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21+
Wins	2196	735	599	387	273	211	135	108	83	46	50	30	17	13	8	7	5	8	4	1	3
Losses	1159	1134	767	564	376	295	221	143	113	88	64	48	26	20	9	17	16	6	3	4	8

Figure 5. Table of results for 10,000 games of craps

Raw data used to create this graph can be found in craps_used.csv

From this data, the mean length of a craps game was calculated to be 3.382 +/- 0.030 rolls, quoted here with the standard error, while the median and mode lengths of a craps game in this sample were 2 and 1 respectively.

The overall probability of winning a game was calculated to be 0.492 +/- 0.005, again quoted with the standard error.

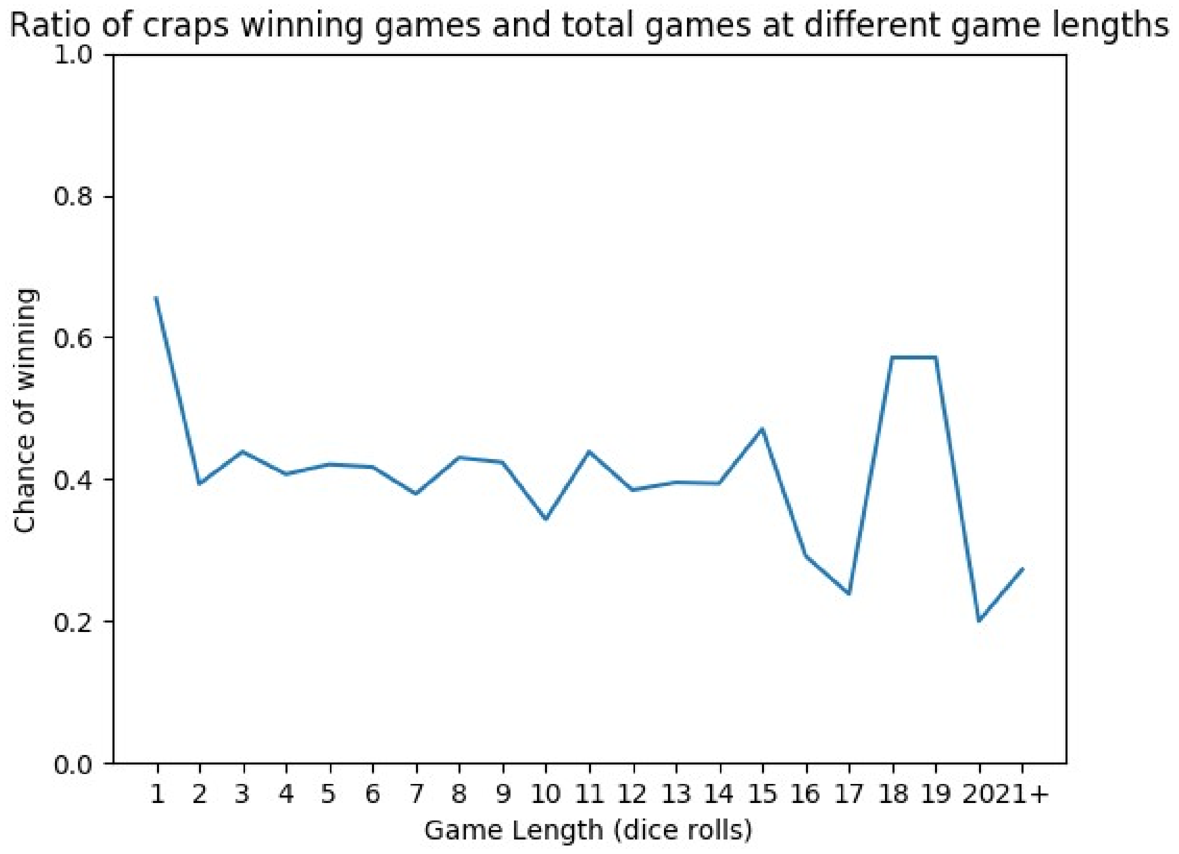

Figure 6. Graph depicting the chance of winning a game of craps when getting a result on the x^{th} pair of dice rolls

Raw data used to create this graph can be found in craps_used.csv

From Figure 6 there is clearly a high chance of winning on the first roll, after which there is a sharp decrease in winning percentage for each following roll, which is fairly stable somewhere around 0.4; however after about 12-15 rolls, it becomes somewhat noisier due to the low frequency with which it gets to that point, and thus lack of data. This is calculated by taking the total wins at each game length, and dividing by the total number of results (wins and losses) at each game length.