**MATH 310 - FINAL PROJECT REPORT**

**Group Members**:

* Zainab Ali Vyajkora
* Hassam Zaheer Butt

**Task 1 – discrete random variables**

**Approach:** For this task, keeping the starting position at 0, we assumed an equally likely probability of moving left, right and and staying in place by generating a random integer in the range {-1,0,1}. The input probabilities are given L=-1 to move left and R=1 is the probability to move right while 0 would result in no change in position. Note that L and R can be changed to increase or decrease the range of numbers in order to simulate with unequal probabilities. For example, a range (-6,2) would mean that the probability to move left is twice as much as that of moving right.

**Assumptions:**

1. The Python random function is an unbiased random number generator.

(Figure: task1.PNG)

**Task 2 – discrete random variables**

**Approach:** For this task, we fixed two starting positions for two people 10 units apart from each other. Using the same approach as in task 1, we used two variables taking different discrete values for each in the range {-1,0,1} at random and fixed a time variable at starting time x=0. Using a while loop that would run until person 1 and person 2 are at the same position, we increment the time variable x and return it once the loop breaks.

**Assumptions:**

1. The Python random function is an unbiased random number generator.
2. It takes 1 second to move 1 step backward or forward for each person and a discrete increment in variable x translates to 1 second passed.
3. Both persons move at the same time.

(Figure: task2.PNG)

**Task 3 – discrete random variables**

**Approach:** Since this model was 2 dimensional, we had two discrete random variables. A variable 'step' to select a discrete step size in the range {0,0.5,1} and a variable 'theta' to select a random angle between 0 and 2π. We used the formulas for polar coordinates to calculate the next position for the point with the obtained random 'step' and 'theta'.

***Logic for bounce back:*** We used the distance formula to calculate the distance from the centre of the circle to the current position of the point as well as the next position in each iteration. **if:** the next position would land outside the circle, we calculated the extra distance as well as the distance remaining to reach the 100-unit boundary. The point would be allowed to travel the remaining distance to reach the 100-unit bound and then bounce back inside the circle travelling the extra distance with the same angle but now with respect to the boundary point instead of the previous position.

**Assumptions:**

1. The Python random function is an unbiased random number generator.
2. Each iteration takes 1 second (Variable 't' for discrete time)

**Note:** A simulation video is attached for reference with **magnified ranges for step size** (Video: task 3 instance.AVI)

**Task 4 – Continuous random variables**

**Approach:** We take the same approach as in task 1 with the difference of how the step size is chosen which is now a continuous random variable obtained as a float value between 0 and 1. We do this by first selecting a random integer (0 or 1) to decide whether the point would move left or right or stay in place. Then the step size is either added(to move right) or subtracted(to move left) to the current position.

(Figure: task4.PNG)

**Task 5 – Continuous random variables**

**Approach:** The approach is the same as task 3 but now the step size will be a float value between 0-1 and the angle would also be a float value between 0-2π generated with Python random.uniform() function used with the specified ranges for each.

**Task 7 – Continuous random variables**

**Approach:** In this case, step size is a discrete RV while the angle would be a float value between 0-2π. The approach is the same as task 3 other than that.

**Task 8**

**Approach:** For this task, we generated two points at two random positions instead of the centre of the circle at (0,0). 2 different random variables theta and step size would be generated at random for each of the points and the formula for polar coordinates would give the starting position. The rest of the logic to keep the points inside the boundary was applied the same way as in previous tasks, keeping track of the distance of both points' current positions from the center. A while loop was used with the terminating condition that distance between the two points becomes <=1.

**Note:** A simulation video is attached for reference with **magnified ranges for step size and terminating condition at distance=30 units.** (Video: task 8 instance.AVI)