Eleanor Sleightholm
201214895

Computational Practical 1 – Introduction to Markov Processes

Random Walks

Random walks are an integral part of mathematics and more specifically, statistics. They are so fundamentally important as they enable mathematicians to observe the behaviours of Markov and many other processes. We explore the computational aspects of creating Markov chains using R in this report whilst also analysing the effects varying sample sizes and probability values can have on random walks.

Before inputting any code into R, we must first consider the requirements needed for a random walk. We know that the walk begins at 0 and goes up by 1 with probability p and down by probability q = 1 – p. So, our first task is to create a random sample of 1's and -1's with probability p and q respectively. To do this, we input the following code into R:

```r
Z <- sample(c(1, -1), 10, replace = TRUE, prob = c(0.6, 0.4))
Z
```

Here, we used probability p = 0.6, q = 0.4 and n = 10. The code here is assigning the variable z to a vector of 10 (our n value) elements which are either 1 or -1 with the probability of 0.6 for a '1' and probability of 0.4 for a '-1'. To make the random walk code complete we need to add the values in this type of vector together with each step. To do so, we use the 'cumsum' command. We input the following command:

```r
X <- cumsum(Z)
X
```

This creates a vector for the random walk. This vector allows us to analyse where the random walk stops and the steps it took in between.

More generally, we could assign p, q and n values before assigning the variables z and x. Together, we have the code that produces a random walk for our desired p, q and n values:

```r
p <- 0.6 # Or your own desired p value
q <- 1 - p
N <- 10 # Or your own desired n value
# Add your code for increments:
Z <- sample(c(1, -1), N, replace = TRUE, prob = c(p, q))
# Should produce a string of 10 plus-or-minus 1s:
Z
# Turn increments into a random walk:
X <- cumsum(Z)
# Should produce something that looks like a random walk:
X
```

Now we have our code, we are able to plot random walks and observe any patterns that arise due to varied n, p and q values. First, we plot for N = 10 and p = 0.6. We use the code above and assign these values and then input the following:

```
plot(1:N, X,     # Takes our N value and X vector and plots them against each other
      type = "b",  # Creates a graph with circles
      col  = "red",    # Makes the graph red
      ylim = c(-10,10),    # Keeps the y axis between the limits -10 and 10.
      xlab = "Number of Steps Taken", # Labels the x-axis
      ylab = "Increments of the random walk",  # Labels the y-axis
      main = "Random Walk for N = 10")    # Labels the whole graph
```

We follow the same code but for N = 10,000. We assign N = 10000, remove the y axis limits, change the colour of the graph and alter the title for N = 10000. Doing so, we are able to produce the following two graphs:

Fig 1

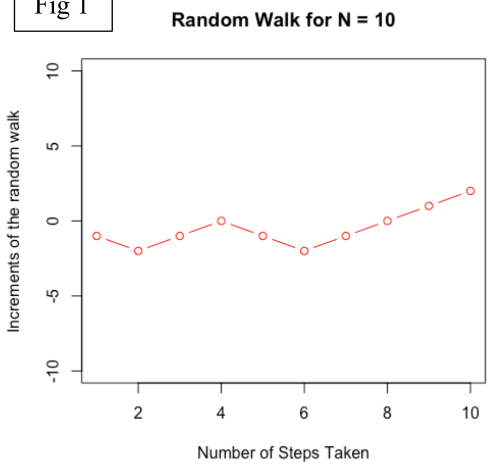
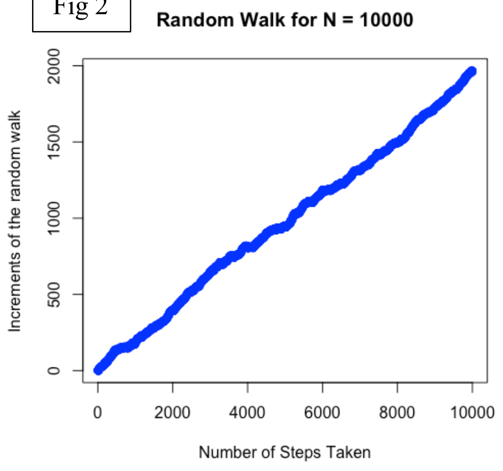


Fig 2

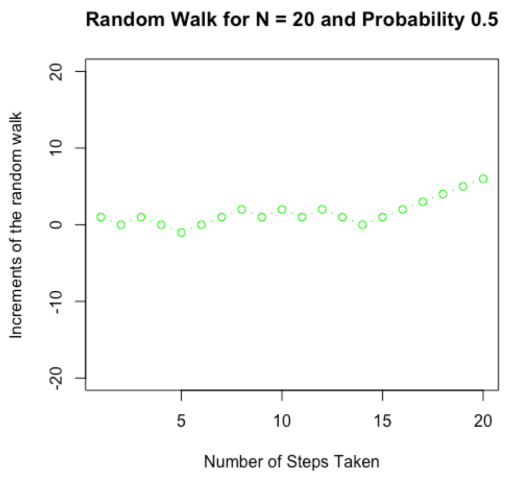


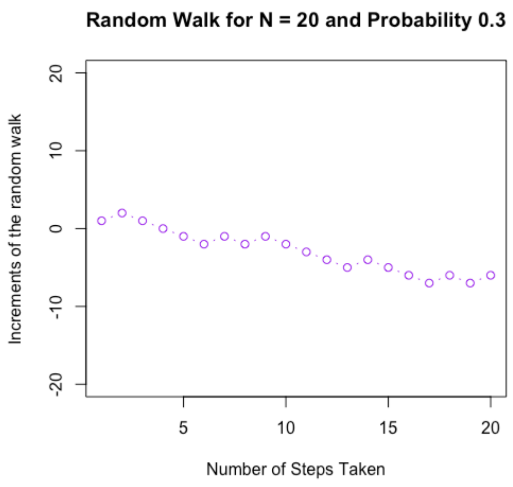
Through this, we observe the impact increasing the value of N has on our random walk. Since we have chosen probability p = 0.6 we expect the walk to increase more than it decreases. This is something that might appear ambiguous at first when plotting multiple graphs for N = 10 as there is no *obvious* pattern. However, when plotting for large N, as in our Figure 2, we see that there is a clear and consistent increase over time. Using probability p = 0.6, we would expect there to be a consistent increase over time as the probability of a 1 appearing is 0.2 percent higher than a -1 appearing. To further our studies, we can vary the value of p for large N. When doing so, we notice that choosing a p value below 0.5 produces a plot with a negative slope and when choosing a p value above 0.5 produces a plot with a positive slope.

In furtherance, to make the computational aspect of this study more concise, we can create a function that produces an X value (vector of the random walk) for specified p and N values:

```
RandomWalk <- function(N, p) {
  Z <- sample(c(1, -1), N, replace = TRUE, prob = c(p, 1-p))
  plot(cumsum(Z),
       type = "b",
       col  = "purple",
       ylim = c(-20,20),
       xlab = "Number of Steps Taken",
       ylab = "Increments of the random walk",
      main = "Random Walk")
}
```

Using this function, we are able to produce plots in a quicker way. We input N = 20 and p =0.5 and N=20 with p=0.3 into our function and produce the following plots:





Again, we see the expected behaviour of the random walk; decreasing with a probability lower than 0.5 and steadily hovering around the 0 mark for the standard random walk, p = 0.5.

Often when analysing computationally, mathematicians need to check the consistency between the computers estimated values and the expected mathematical values. For a random walk, we know that the expected value of a simple random walk is defined by:

$$\mathbb{E}X_n = \mathbb{E}\left(X_0 + \sum_{i=1}^{n} Z_i\right) = \mathbb{E}X_0 + \sum_{i=1}^{n} \mathbb{E}Z_i = \mathbb{E}X_0 + n\mathbb{E}Z_1,$$

Therefore, we can compute our estimated value of the random walks we produce in R and compare that with the definition of the expectation. This will allow us to see how accurate the computational methods are.

For our estimation, we must consider a large number of samples of the random walk. To do this, we use the ‘replicate’ function. Inputting the following code allows us to observe the estimated value for our specified random walk. We assign the variable ‘trials’ to be a large number to allow our estimation to be as accurate as possible but this value can be changed to a different appropriate large number. We use the ‘RandomWalk’ function to create multiple samples of this type of random walk and finally, take the expectation of these values.

```
N <- 100
p <- 0.5
trials <- 10000  # Or some other appropriate large number
```

```
samples <- replicate(trials, RandomWalk(N, p)[N])

expectation = sum(samples)/trials

expectation
```

First, we take a random walk with sample size N = 100 and p = 0.5 (the simple random walk). As per the definition above, we know the expected value would be 0 + 100*(0.5-0.5) since X_0 begins at 0. This gives us an expected value of 0 which is what we hope to achieve when inputting these specific values into our code above. Upon the first 5 trials we get the following estimated values: 0.048, -0.49, -0.006, 0.306 and 0.37 which are all fairly consistent with our expected value. We have observed no anomalies and all values begin with 0 so from this we see that our code appears sufficient and accurate.

Before making a final conclusion, we must test the code out with many other p values. We take N = 100 and p = 0.3, then the expected value is 0 + 100(0.3-0.7) = -40. Again, we use our code above and R produces the following estimated values: -40.094, -40.206, -40.118, -39.898 and -39.844 which, again, are consistent with our expected value and no anomalous data is present.

After continued testing for varying probabilities we can conclude that for N = 100, the estimated values for our generated random walks coincide with the actual expected value. This allows us to be confident with our generated random walks as they fulfil the properties needed/expected for them.

Overall, we can conclude that choosing a probability value larger than 0.5 and a sufficiently large N value, results in a random walk that continually increases over time despite occasionally decreasing. Similarly, for a probability less than 0.5, the random walk continually decreases over time despite occasionally increasing. We have observed that it will increase/decrease to the value (p-q)*n. Further, we can also conclude that the computational methods presented in this report are consistent with mathematical theory of the expectation of random walks.