Monte Carlo Simulations Dissertation

Daniel Watson 213886

May 26, 2020

R version: 3.6.1 (2019-07-05) – "Action of the Toes"

1 Question One: Generation of (pseudo-)random numbers

Our random variable X is discrete and is Poisson distributed with the probability mass function $f(k;\lambda) = \frac{\lambda^k e^{-\lambda}}{k!}$ where $\mathbb{E}(X) = \mathbb{V}(X) = \lambda$.

The algorithms we use in R to generate X is taken from pages 501-505 Devroye's 1986 book "Non-Uniform Random Variate Generation". The first algorithm (Lemma 3.2 in Devroye's book) uses exponentially distributed random variables to generate the Poisson distributed X and is described below:

If $E_1, E_2, \ldots$ are i.i.d exponential random variables, and X is the smallest integer such that:

$\sum_{i=1}^{X+1} E_i > \lambda$,

then X is Poisson(λ).

Below is the implementation of this algorithm in R with some modifications, where the generate function allows us to generate n samples using a for loop with a specified rate λ.

It is important to note that the algorithm does not specify the rate of the exponential variables to use and so we choose it to be equal to one. This is to ensure that our Poisson variables are generated with the correct rate that we want. Otherwise, λ we obtain will be equal to that of the rate of the exponential random variables multiplied by the λ that we specify.

```
generate <- function(lambda, n) {
  x <- c()
  for (i in 1:n) {
  X <- 0
  sum <- 0
  while (TRUE) {
    exponential <- rexp(1, rate = 1)
    sum <- sum + exponential
```

```
      if (sum < lambda) {
        X <- X + 1
      }
      else {
        x[i] <- X
        break
      }
    }
  }
  return (x)
  }
```

I have chosen the values of λ to be equal to 10 and 50 each with sample sizes of 1000, 10,000, 100,000 and 1,000,000. The code below assigns the Poisson random variables to a new variable named result, where i and lambda are the looped values of sample sizes and λ values. We produce the values of k that are to be used to generate the theoretical probability mass function and assign the result to the variable y. We can then plot our generated variables as a histogram with density on the y-axis instead of frequencies so that we can super-impose our theoretical curve alongside it. This allows a direct visual evaluation on how good of a fit our generated variables are.

```
result <- generate(lambda = lambda, i)
k <- seq(0,max(result), length = max(result)+1)
y <- exp(-lambda)*((lambda**k)/factorial(k))
hist(result, breaks = (max(result)-min(result)), probability = TRUE,
main = paste("Histogram with sample size", i,"and rate", lambda, sep = " "))
lines(k,y)
```

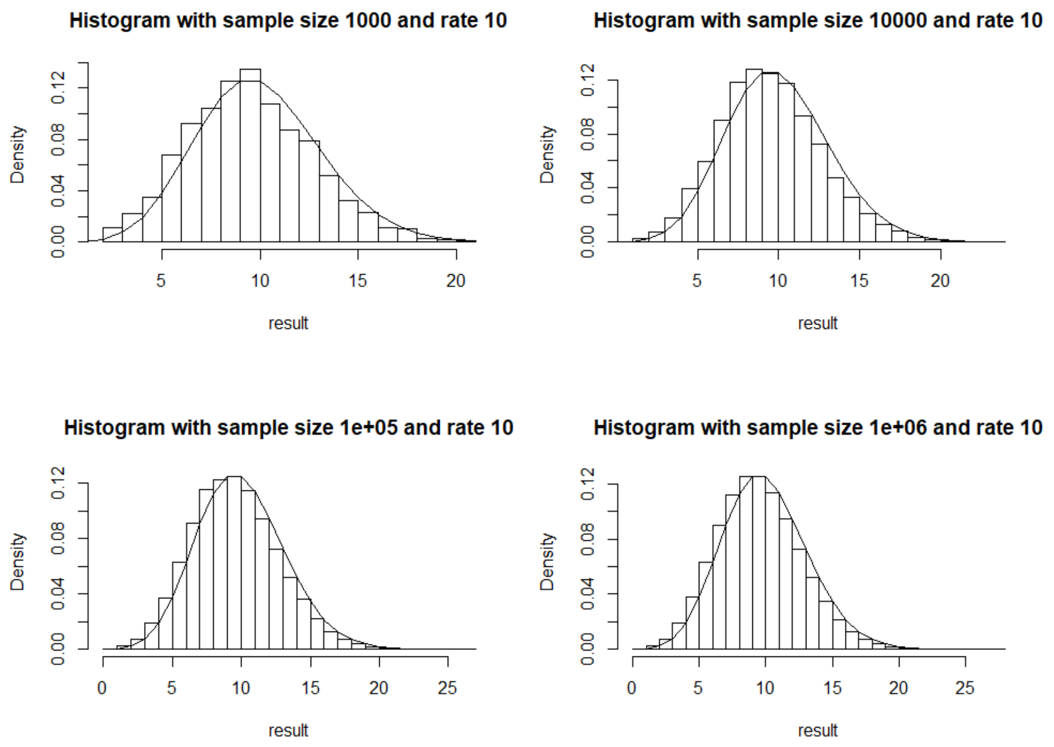


Figure 1: Plot of density histogram of Poisson variables and its theoretical curve for increasing sample size with rate = 10.

It is also true in this case that as sample size increases, the generated variables become a better fit for a Poisson distribution. Of course it is helpful to assess this visually, we now look

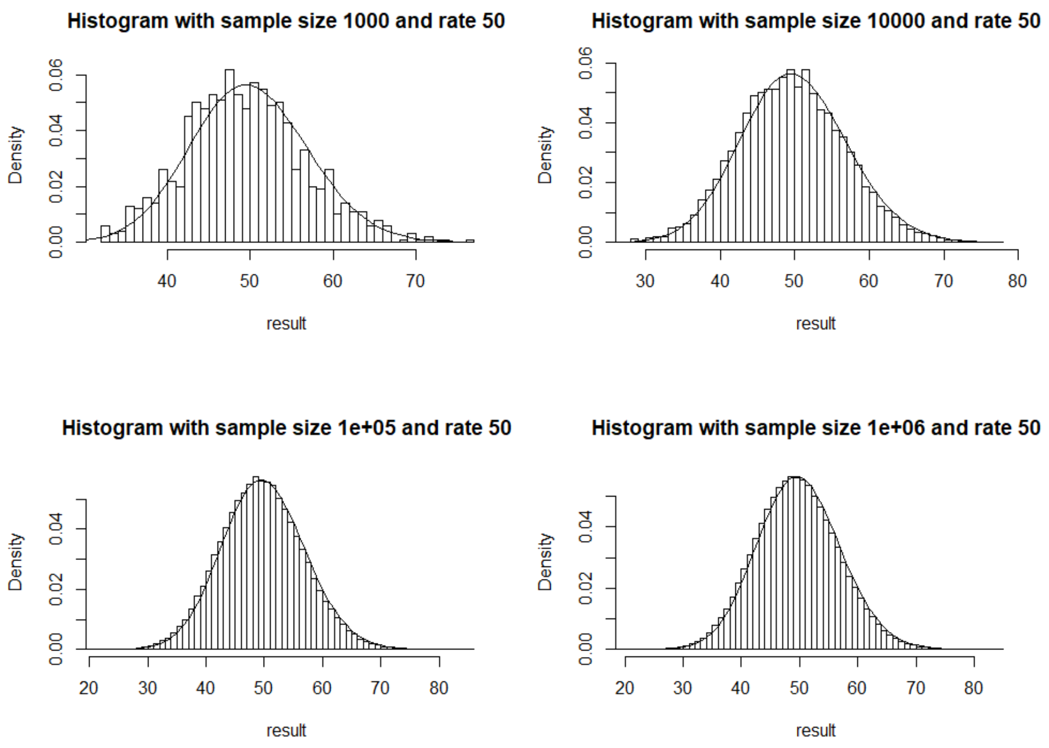


Figure 2: Plot of density histogram of Poisson variables and its theoretical curve for increasing sample size with rate = 50.

to a numerical evaluation of goodness-of-fit. As our distribution is discrete, the chi-squared test will be used. The null hypothesis will therefore be that there is no significant difference between the theoretical and generated samples which can be evaluated in R. The observed values are the frequencies of each bin and y is recalculated as the vector of theoretical frequencies. We use the formula:

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

where n is the number of bins, E_i is equal to the theoretical count for bin i and O_i is the actual count for bin i which correspond to our y and result vectors. We also need the number of degrees of freedom. This is equal to the number of bins - 2 for the Poisson distribution as there are 2 constraints. The table chi-squared value that corresponds to the calculated degrees of freedom with chosen significance is produced. The table and observed chi-squared values are then compared, with the null hypothesis being rejected if the observed chi-squared value is higher than the table value.

```
observed_vals <- c()
for (j in 0:max(result)) {
  observed_vals[j+1] <- length(which(result==j))
}
y <- y*i #Theoretical frequencies
chi_sq <- sum((((observed_vals - y)**2))/y)

df <- length(observed_vals) - 2

chisq_table <- qchisq(p = 0.10, df = df)
Accept_H0 <- chi_sq <= chisq_table
print(Accept_H0)
```

In this implementation, a criterion of $\alpha = 0.1$ was used. In both examples with rate 10 and 50, the null hypothesis that the samples come from the same distribution tends to be rejected, although sometimes is accepted. This is due to the fact that the values obtained are distributed and so answers change upon reruns of the code. It does seem that in this case, the algorithm is not perfect which be seen visually, with the results below the mean tending to be over-represented with the curve being below the histogram values.

2 Question Two: Markov Chains and Markov Chain Monte Carlo

In this section we explore Markov Chains. We use a variation of the Ehrenfest urn whereby we have two marbles and two possible sections where the marbles can lie. At each step, a random marble is selected with uniform distribution and with a certain probability it is moved into another section of the urn at random also. We now present defintions that help explain the properties of our Markov chain:

Definition (Discrete Homogeneous Markov chain). *We let $X_1, \ldots, X_n, \ldots$ be a sequence of random variables with each X_i taking values in a finite or countable state space $\mathcal{S}$. For $k = 0, 1, 2, \ldots$, define the k-step transition probabilities as:*
$p_{i,j}^{(k)} = Pr(X_{t+k} = j | X_t = i) \ldots (i, j \in \mathcal{S})$
The Markov chain is homogeneous due to the fact that the transition matrix P does not depend on on t.

We can interpret this by recognising that in the Markov chain, if given the present state then the past state contains no additional information on the future state and so predictions on future outcomes can be made solely based on its present state.

Definition (Irreducibility). *A Markov chain is irreducible if any state can be reached from any other state which can be formalised as being for any pair $i, j \in \mathcal{S}$, there exists a k such that $p_{i,j}^{(k)} > 0$. This is also referred to as ergodicity.*

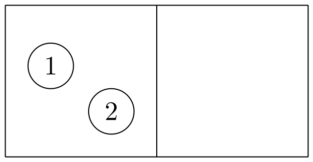
This can also be defined as all of the states being able to communicate with eachother - they are in the same communication class.

Definition (Periodicity). *An irreducible chain has period D which is a positive integer if D is the greatest common divisor of $\{k \geq 1 : p_{i,i}^{(k)} > 0\}$ for some $i \in \mathcal{S}$. If D is equal to one then the chain is aperiodic. It is also true that if $p_{i,i} > 0$ for some i then it is aperiodic.*

Our initial state is that both of the marbles are present in the left section which we define as being $(X_1 = 1, X_2 = 1)$. There are g^n individual descriptions of the states where g is the number of categories (2 in our case) and n is the number of marbles (2 in our case) meaning that there are 4 individual descriptions. There are also $\binom{(n+g-1)}{(g-1)}$ stastical descriptions which is equal to three. This is the number of possible states which means that our state space is:

$$\mathcal{S} = \{(2, 0), (1, 1), (0, 2)\}$$

Below is a graphical representation of the urn next to the transition matrix corresponding to the Markov chain.



$$P = (p_{i,j}) = \begin{pmatrix} 2/3 & 1/3 & 0 \\ 1/6 & 2/3 & 1/6 \\ 0 & 1/3 & 2/3 \end{pmatrix}$$

When a marble is selected we roll a standard 6-sided die. If the dice shows one or a two then the marble is moved, otherwise it remains in the same place. This corresponds to a 1/3 chance of the marble moving after selection. The probabilities above are obtained through the following calculations:

- $P_{0,0} = 2(1/2 \times 2/3) = 2/3$
- $P_{0,1} = 2(1/2 \times 1/3) = 1/3$
- $P_{0,2} = 0$
- $P_{1,0} = (1/2 \times 1/3) = 1/6$
- $P_{1,1} = (1/2 \times 2/3) + (1/2 \times 2/3) = 2/3$
- $P_{1,2} = (1/2 \times 1/3) = 1/6$
- $P_{2,0} = 0$
- $P_{2,1} = 2(1/2 \times 1/3) = 1/3$
- $P_{2,2} = 2(1/2 \times 2/3) = 2/3$

Of course only one marble is selected at a time and so $p_{2,0}$ and $p_{0,2}$ are equal to zero. Our Markov chain is irreducible as any state can be reached from any other state for a certain value of k. We can demonstrate this in R by using the expm package which allows one to take powers of a matrix. Below is a construction of our transition matrix, where at the end we take the second power of the matrix. This produces a matrix where all entries are greater than zero meaning that our Markov chain is indeed irreducible.

```
probs <- matrix(rep(0,9),nrow=3,ncol=3)
probs[1,1] <- 2/3
probs[1,2] <- 1/3
probs[2,1] <- 1/6
probs[2,2] <- 2/3
probs[2,3] <- 1/6
probs[3,2] <- 1/3
probs[3,3] <- 2/3
probs %^% 2
```

It is even easier to validate that the Markov chain is aperiodic due to the fact that it is irreducible and we have $p_{i,i} > 0$ for all i - all diagonal entries are non-zero hence it is aperiodic.

We now describe how the Markov chain behaves in the long run by first defining the invariant, equilibrium or stationary distribution.

Definition (Invariant distribution). *If we have an aperiodic and irreducible Markov chain with state space $\mathcal{S}$ then for every $i, j \in \mathcal{S}$ then the limit:*

$$\pi_j = \lim_{k \to \infty} p_{i,j}^k$$

exists and is independent of i. Additionally, if $\mathcal{S}$ is finite then $\sum_{j \in \mathcal{S}} \pi_j = 1$.

This means that the columns of the transition matrix become the individual stationary distribution values.

Our invariant distribution can be calculated using the formula:

$$\pi^T P = \pi$$

where we expect that all of the individual values in the distribution sum to one considering our state space has a cardinality of three, which is of course finite. Our distribution becomes stochastic. For our transition matrix:

$$(\pi_1, \pi_2, \pi_3) \begin{pmatrix} 2/3 & 1/3 & 0 \\ 1/6 & 2/3 & 1/6 \\ 0 & 1/3 & 2/3 \end{pmatrix} = \begin{pmatrix} \pi_1 \\ \pi_2 \\ \pi_3 \end{pmatrix}$$

$$2/3\pi_1 + 1/6\pi_2 = \pi_1$$
$$1/3\pi_1 + 2/3\pi_2 + 1/3\pi_3 = \pi_2$$
$$1/6\pi_2 + 2/3\pi_3 = \pi_3$$

So that $\pi = (1/4, 1/2, 1/4)$ after some manipulation of the simultaneous equations.
We now produce a Monte Carlo simulation of this situation in R. If we produce our matrix P in R and take the 20th power, we can see that each row in every column has the same value, corresponding to the theoretical stationary distribution of that state.

```
X <- rep(1,2)
 Niter <- 100000
K <- rep(0,Niter)
Z <- 0
 for (i in 1:Niter) {
   k <- sample(2, size = 1)
   if (runif(1)<1/3){X[k] <- -X[k] + 1} else {X[k] <- X[k]}
   Z = sum(X)
   K[i] <- Z
   Z <- 0
   }
K
```

We make use of the above code to produce our Monte Carlo simulation where X is the chosen individual description which also serves as the initial state of the simulation. In our case this is when both marbles are on the left size of the urn. Niter of course specifies the number of iterations, while K is the vector of states that will be analysed further later on. k is a variable that randomly selects one of the marbles and the next line is a representation of the dice. Of course if the roll is at or below 2 (probability of 1/3 out of a fair 6 sided die) then the marble selected is moved. Z then takes the sum of the marbles on the left side (0, 1 or 2) of the urn and this is recorded as the current state of the chain.

We can begin to simulate our empirical distribution which corresponds to the stationary distribution using the following code, assigned to the freq variable. This is calculated by storing the number of states of a certain value which is divided by the total number of iterations to calculate proportions.

```
freq <- rep(0,3)
for (j in 1:3) {
  freq[j] <- length(which(K==j-1))/Niter
}
freq
```

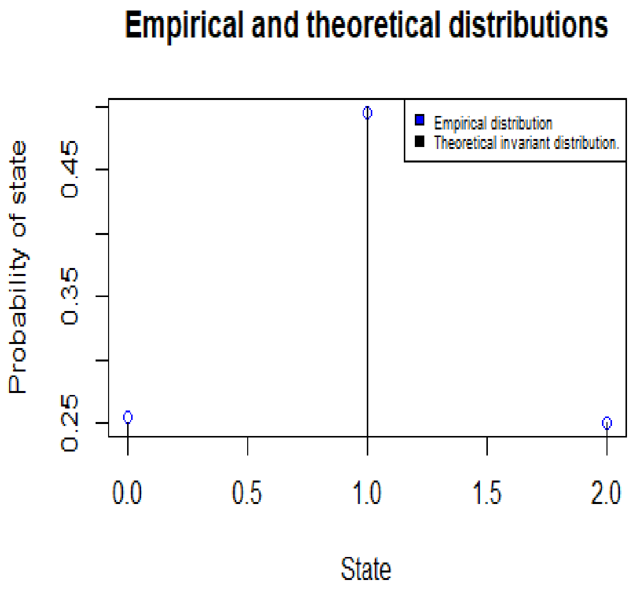


Figure 3: Empricial vs theoretical stationary distributions of the Markov chain

The figure above shows both the theoretical invariant distribution while also the empirical distribution generated. As you can see, the two vectors are very similar with the empirical distribution being $(0.254, 0.496, 0.25)$. Although the probability of state zero is slightly overestimated in the simulation and state one is underestimated, this generated distribution is very close to the theoretical version.

We can now assess how the empirical running mean of states changes over time and we can compare this to the theoretical mean using the code below. We choose n to be equal to 10, meaning every 10th value is chosen out of the states. The running mean is calculated by slowly increasing the number states available where m is the theoretical expected state that is calculated easily due to the fact that the distribution is discrete. If we define X as our random variable with its outcomes as $x_1, x_2, \ldots, x_k$ with probabilities $p_1, p_2, \ldots, p_k$ then the expectation of X is:

$$\mathbb{E}[X] = \sum_{i=1}^{k} x_i p_i$$

In our case, $x_1, x_2, \ldots$ are the possible number of marbles on the left side, being 0, 1 or 2. Our probabilities are our invariant distribution values 0.25, 0.5 and 0.25 again, giving an expected value of $0.25 * 0 + 0.5 * 1 + 0.25 * 2 = 1$ which we denote m in R.

```
n <- 10
Mean = seq(0,Niter/n-1)
for (i in 1:Niter/n) {Mean[i]=mean(K[1:i])}
m <- 1
TMean = rep(m, Niter/n -1)
```

Figure four demonstrates a plot of this.

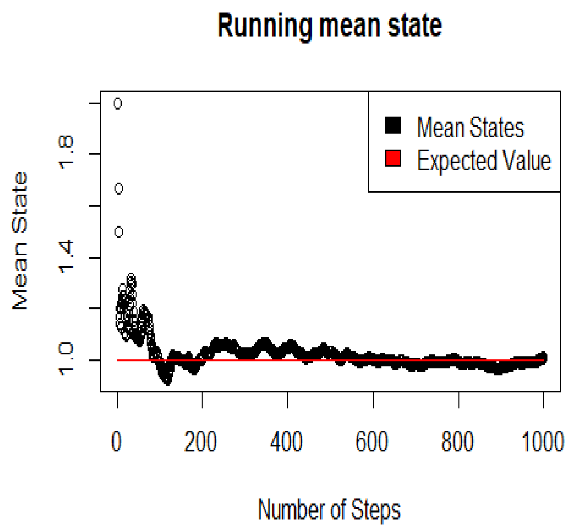


Figure 4: Empirical vs theoretical mean

The following code follows the same working but for the running variance where the variance is:

$$\mathbb{V}[X] = \sum_{i=1}^{k}(x_i - \mu)p_i$$

This gives $\mathbb{V}[X] = 0.5$ in our case. Figure five shows the plot of running variance against the theoretical variance.

```
Var = seq(0,Niter/n-1)
for (i in 1:Niter/n) {Var[i]=var(K[1:i])}
v <- 0.5
TVar = rep(v,Niter/n-1)
plot(Var)
lines(TVar)
```

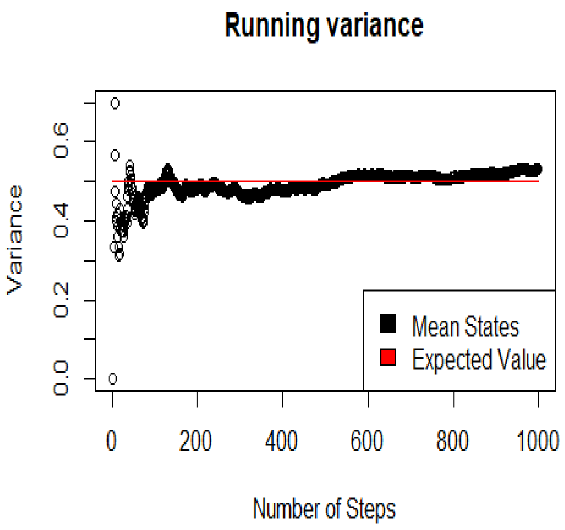


Figure 5: Empirical vs theoretical variance

3 Question Three: Ising model

In this question we consider an Ising model with 5 spins. Let $G = (V, E)$ be our graph with vertex set V and edge set E with notation $< i, j >$ to be the edge with endpoints at the vertices i and j. At each vertex, we have a spin variable s_i that has the possible states $S = \{-1, +1\}^V$, so $s_i = 1$ is when the spin is up and $s_i = -1$ is when the spin is down. The figure below shows a representation of what our graph looks like:

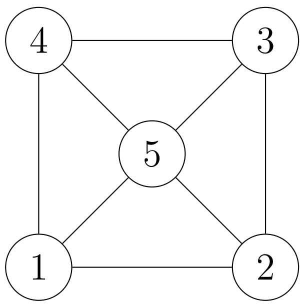


Figure 6: Representation of the graph

The Hamiltonian for a certain state s is defined as:

$$H(s) = -J \sum_{<i,j>\in E} s_i s_j$$

where J is a coupling constant which is positive in the ferromagnetic case and negative in the anti-ferromagnetic case and random for spin glasses. In our case, the Hamiltonian is:

$$H(s) = -J(s_1 s_2 + s_2 s_3 + s_3 s_4 + s_4 s_1 + s_1 s_5 + s_2 s_5 + s_3 s_5 + s_4 s_5)$$

Which are characterised by which vertices are connected by which edges. Hence for each different state, the hamiltonian has a different value - aswell as the magnetisation. We now produce a table that shows every possible combination of states and their Hamiltonian's and magnetisations as well as the magnetisation value divided by the number of vertices (5):

State	H(s)	M(s)	$\frac{M(S)}{\|V\|}$
+++++	-8J	5	1
++++-	0	3	3/5
+++-+	-2J	3	3/5
++-++	-2J	3	3/5
+-+++	-2J	3	3/5
-++++	-2J	3	3/5
+++–	2J	1	1/5
++-+-	2J	1	1/5
++–+	0	1	1/5
+-++-	2J	1	1/5
+-+-+	4J	1	1/5
+–++	0	1	1/5
-+++-	2J	1	1/5
-++-+	0	1	1/5
-+-++	4J	1	1/5
–+++	0	1	1/5
++—	0	-1	-1/5
+-+–	4J	-1	-1/5
+–+-	0	-1	-1/5
+—+	2J	-1	-1/5
-++–	0	-1	-1/5
-+-+-	4J	-1	-1/5
-+–+	2J	-1	-1/5
–++-	0	-1	-1/5
–+-+	2J	-1	-1/5
—–	-8J	-5	-1
—++	2J	-1	-1/5
+—-	-2J	-3	-3/5
-+—	-2J	-3	-3/5
–+–	-2J	-3	-3/5
—+-	-2J	-3	-3/5
—-+	0	-3	-3/5

The partition function is given by: $Z(\beta) = \sum_{s \in S} exp[-\beta H(s)]$ which in our case is:

$$Z(\beta) = 8e^{-2\beta J} + 8e^{2\beta J} + 2e^{8\beta J} + 4e^{-4\beta J} + 10 = 16cosh(2\beta J) + 10 + 2e^{8\beta J} + 4e^{-4\beta J}.$$

The thermodynamic potential, F is also given by: $F = -kTlog(Z)$ which is equal to:

$$F = -kTln[16cosh(2\beta J) + 10 + 2e^{8\beta J} + 4e^{-4\beta J}].$$

The entropy is:

$$S = -\frac{\partial F}{\partial T} = kln[16cosh(2J\beta) + 10 + 2e^{8J\beta} + 4e^{-4J\beta}] + \frac{8Jk\beta(e^{-4J\beta} - e^{8J\beta} - 2sinh(2J\beta))}{2e^{-4J\beta} + e^{8J\beta} + 8cosh(2J\beta) + 5}$$

where $\beta = \dfrac{1}{kT}$ with T being absolute temperature and k being Boltzmann's constant.

The energy is:

$$-\frac{1}{Z}\frac{\partial Z}{\partial \beta} = \frac{32Jsinh(2\beta J) + 16Je^{8\beta J} - 16Je^{-4\beta J}}{16cosh(2\beta J) + 10 + 2e^{8\beta J} + 4e^{-4\beta J}}$$

This can also be calculated as $F + TS = F + \frac{1}{k\beta}S$

The expected magnetisation per spin is $m = \mathbb{E}[m(s)]$ where $m(s) = \frac{M(S)}{|V|}$ and is specified in the previous table of states. This expectation can be calculated with the following formula:

$$\mathbb{E}[m(s)] = \sum_{s \in \mathcal{S}} m(s)\frac{e^{-\beta H(s)}}{Z(\beta)}.$$

The absolute average magnetisation $|m|$ is therefore given by:

$$|m| = \frac{2e^{8\beta J} + 6/5 + 8/5e^{-2\beta J} + 8/5 + 24/5e^{2\beta J} + 4/5e^{-4\beta J}}{16cosh(2\beta J) + 10 + 2e^{8\beta J} + 4e^{-4\beta J}}$$

We can also see that the absolute magnetisation, $|M| = 60$.

The following code sets up our monte-carlo simulation where we define the temperature to be 1.9 and the number of iterations to be 100000. We construct our vector of states, s with an initial state of all spins being up. We can then define the Hamiltonian that we calculated previously, as well as setting up our magnetisation and energy vectors and taking the initial value of energy to be equal to the Hamiltonian defined earlier in the code, while the initial magnetisation is the sum of the initial state of the spins. All spins are up and so this is equal to 5.

```
kTJ = 1.9
betaJ = 1/kTJ
Niter = 100000

s <- rep(0,5)

for (i in 1:5) {s[i]= 1}
```

```
H = -betaJ*(s[1]*s[2]+s[2]*s[3]+s[3]*s[4]+s[4]*s[1] + s[1]*s[5] + s[2]*s[5] + s[3]*s[5] + s[4]*s[5]

E <-rep(0,Niter+1)
M <-rep(0,Niter+1)
E[1] <- H
M[1] <- sum(s)
```

We are now ready to perform the simulation, described by the code given below:

```
for (i in 1:Niter) {
  k <- sample(5, size = 1) # selects a spin
  Hs = -(s[1]*s[2]+s[2]*s[3]+s[3]*s[4]+s[4]*s[1] + s[1]*s[5] + s[2]*s[5] + s[3]*s[5] + s[4]*s[5])
  s[k]<- -s[k] # the spin is flipped
  Hy = -(s[1]*s[2]+s[2]*s[3]+s[3]*s[4]+s[4]*s[1] + s[1]*s[5] + s[2]*s[5] + s[3]*s[5] + s[4]*s[5])
  DeltaH <- -betaJ*(Hy - Hs)
  alpha <- min(1,exp(DeltaH))
  U <- runif(1)
  if (U<=alpha)
  {E[i+1]<-E[i]+DeltaH
  M[i+1]<-sum(s)}
  else
  {s[k] <- -s[k]
  E[i+1]<- E[i]
  M[i+1]<- M[i]}
}
```

The metropolis-hastings algorithm must first be defined before this piece of code can be explained:

Definition (Metropolis-Hastings algorithm). *Let $\mathcal{S}$ be a finite state space and Q be a symmetric transition probability matrix. The target probability distribution is π on $\mathcal{S}$ with π_i for every $i \in \mathcal{S}$. A Markov chain $X_0 \dots X_N, \dots$ is built, with $X_t = i$ being the current state. The following algorithm determines the next state in the chain:*

- *Choose $Y = j$ where $\mathbb{P}(Y = j|X = i) = q_{i,j}$. $Y = j$ is known as the trial state.*
- *Compute $\alpha = min\{1, \dfrac{\pi_Y}{\pi_i}\}$*
- *Accept $Y = j$ with probability α meaning that a random uniformly distributed number $U \sim U[0,1]$ is generated. If $U \leq \alpha$ then accept $Y = j$ meaning that $X_{t+1} = Y$. If this is not true then set the new state to be equal to the old state, namely $X_{t+1} = X_t$.*

We actually have two Markov chains, one being the vector of energy values and the other being the vector of magnetisation of the states. Hence if $U \leq \alpha$ then the X_{t+1} that corresponds to the new energy state is equal to the old state added to the change in Hamiltonian values. If rejected, the new energy value is equal to that of the previous value. For the magnetisation, the new magnetisation value after acceptance would be equal to the sum of the new state after a flipped spin. If rejected then of course the new energy value is the same as the last like the energy.

$$\text{Additionally, } \frac{\pi_Y}{\pi_i} = e^{DeltaH}$$

In the code, our k variable selects a random spin, the current hamiltonian is calculated and then one spin is randomly flipped. The new Hamiltonian is then calculated, and the change can be given by the difference of these two multiplied by the inverse temperature. The metropolis-hastings algorithm is then used where we calculate α, the acceptance probability. $U \sim U[0,1]$ is then generated whereby if it is smaller than the acceptance probability then the move is accepted. If we pass in the formula we calculated for the absolute average magnetisation $|m|$ into R using the defined values of β and J then this gives a value of 0.889. This is our expected value. The following code allows us to produce the figure below. This figure shows the running average of the simulation against our calculated expected value and we can clearly see that our running mean converges to this expected value which is a good sign that the simulation is working well.

```
memp <- mean(abs(M[5000:10000]))/5

# Running average (sampled every n Monte Carlo steps)
n <- 10
Mean = seq(0,Niter/n-1)
for (i in 1:Niter/n) {Mean[i]=mean(abs(M[1:i]))/5}
m <- 0.8889
TMean = rep(m,Niter/n -1)
plot (Mean,xlab="MC steps per spin",ylab="|m| (kT/J = 1.9)")
lines(TMean, col = "red")
```

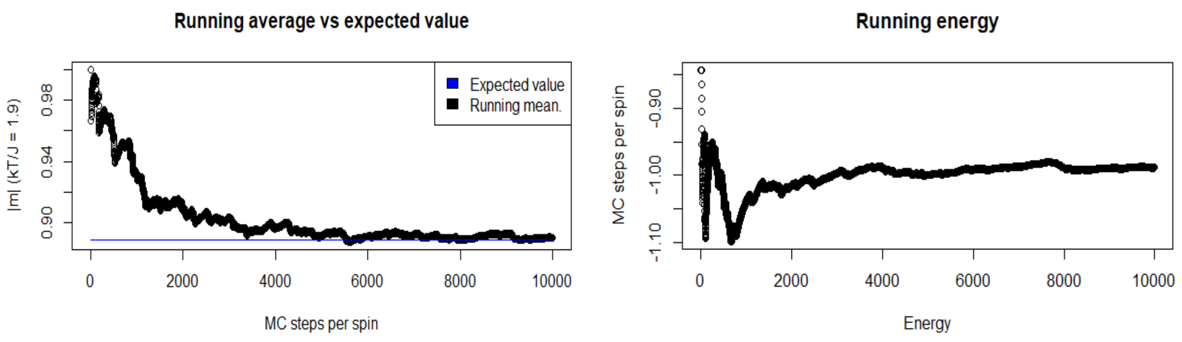


Figure 7: Expected magnetisation against running average as well as the running energy

We can also produce a figure of how the magnetisation changes with an increase in temperature which is shown below. It is clear to see that as temperature increases, the magnetisation decreases and that the empirical magnetisation also follows the theoretical magnetisation well. Next to it we include a plot of how the theoretical energy changes with temperature also:

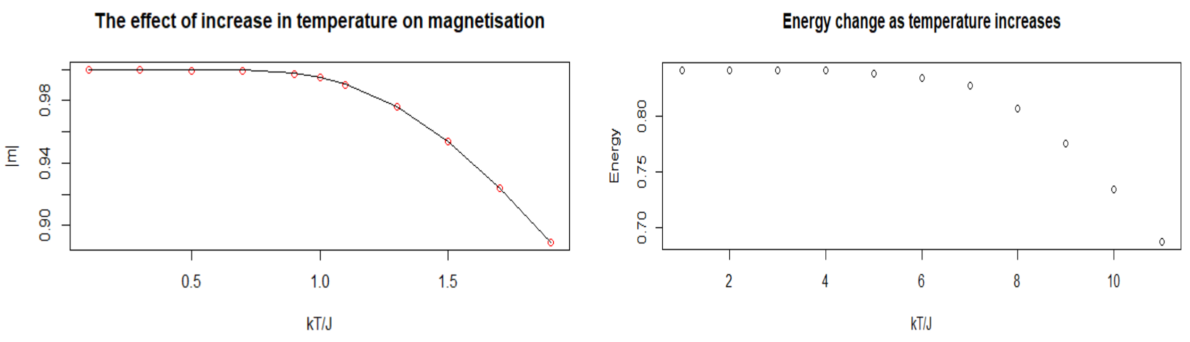


Figure 8: Plot of temperature against magnetisation and temperature against energy

4 Question Four: Random walk

In this question we use the random variable defined in question one which was generated from exponentially distributed random variables. Hence we have a sequence of i.i.d random variables $\{X_i\}_{i=1}^N$ where $X_i \sim$ Pois (λ). The cumulative distribution function of our random variables is:

$$F_X(k) = \mathbb{P}(X \leq k) = e^{-\lambda} \sum_{i=0}^{\lfloor k \rfloor} \frac{\lambda^i}{i!}$$

We now define:

$$Z_N = \textstyle\sum_{i=1}^N X_i$$

We have that if $X_i \sim \text{Pois}(\lambda_i)$ for $i = 1, \ldots, n$ are i.i.d then $\sum_{i=1}^N X_i \sim \text{Pois}(\sum_{i=1}^N \lambda_i)$.

For now we choose various N values - 10, 100, 1000 and 5000 with a value of $\lambda = 1$. This means that $Z_N \sim \text{Pois}(N\lambda) = \text{Pois(N)}$ and its cumulative distribution function is:

$$F_{Z_N}(k) = \mathbb{P}(Z_N \leq k) = e^{-N\lambda} \sum_{i=0}^{\lfloor k \rfloor} \frac{N\lambda^i}{i!}$$

We can now use the code:

```
n <- 10000
Ns <- c(10, 100, 1000, 5000)
lambda = 1
for (N in Ns) {

ZN <- rep(NA,n)

for (i in 1:n) {
  ZN[i] <- sum(generate(lambda = lambda, N))
}
```

where n is the number of samples which we keep at 10000 throughout this question, where the vector Ns contains the number of Poisson random variables to be generated as defined before. We make use of our previous generate function defined in question one meaning that in total, 10000 Z_N variables are generated. We can construct a theoretical distribution in R which can be used to compare our empirical distribution to the theoretical. This can be done using the following code:

```
x <- seq(0, max(ZN), length = max(ZN) + 1)
FZN <- dpois(x, lambda = lambda*N)
hist(ZN, breaks = max(ZN) - min(ZN) , probability = TRUE,
main = paste("Histogram with N value: ", N, sep = " "))
lines(FZN)
```

We can use the following code to produce the CDF of the theoretical distribution we generated, as well as to find the empirical CDF. We provide plots of comparisons between these two CDFs:

```
CDF <- rep(NA,n)
for (i in 1:(n)) {
  CDF[i] <- sum(FZN[1:i])
```

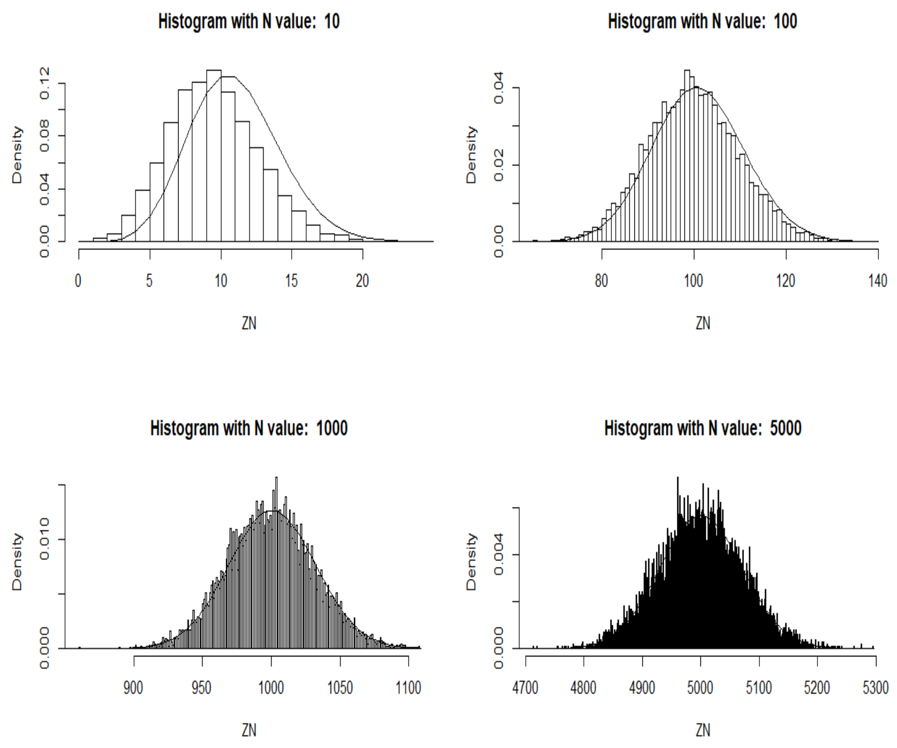


Figure 9: Plot of density histogram of the sum of increasingly larger poisson variables and their theoretical curves. It is easy to see that the ZN vector is a good fit for the theoretical distribution, although it is clearly not perfect.

```
}
ZNcdf = ecdf(ZN)
plot(ZNcdf, col = "red", lty = 1, main = paste("Theoretical vs Empirical CDFs
with N value: ", N, sep = ""))
lines(CDF, col = "blue", lty = 2)
legend(x = "right", legend = c("Fit", "Observed"), col = c("red", "blue"), lty = 1:2)
```

We can see plots of the theoretical versus the empirical distribution for increasing N values below:

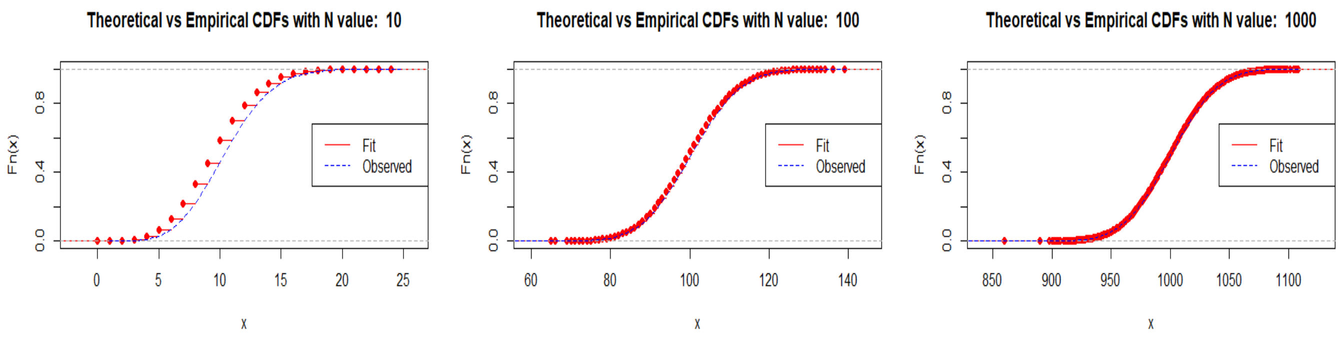


Figure 10: Plot of CDFs of empirical and theoretical distribution for increasing N values. It is clear that the CDFs are very similar, indicating a good fit.

We now move onto to the topic of verifying the central limit theorem. If $\mathbb{E}(X) < \infty$ and $\mathbb{V}(X) < \infty$ then the variable:

$$U_N = \frac{Z_N - N\mathbb{E}(X)}{\sqrt{N\mathbb{V}(X)}}$$

converges to a standard normal variable as N approaches ∞. That is $U_N \sim \mathcal{N}(0,1)$ as $N \to \infty$.
In our case, conveniently $\mathbb{E}(X) = \mathbb{V}(X) = \lambda = 1$ due to the fact that it is Poisson distributed. This means that:

$$U_N = \frac{Z_N - N}{\sqrt{N}}$$

Using the code:

```
expectation <- lambda
variance <- lambda

UN <- (ZN - N*expectation)/sqrt((N*variance))
normSeq <- seq(-10, 10, 0.1)
y = dnorm(normSeq)
hist(UN, breaks = 'FD', probability = TRUE, main = paste("Histogram of UN with N value:"
, N, sep = " "))
lines(normSeq, y)
```

we can calculate the U_N variable and plot it against a theoretical standard normal curve. This gives:

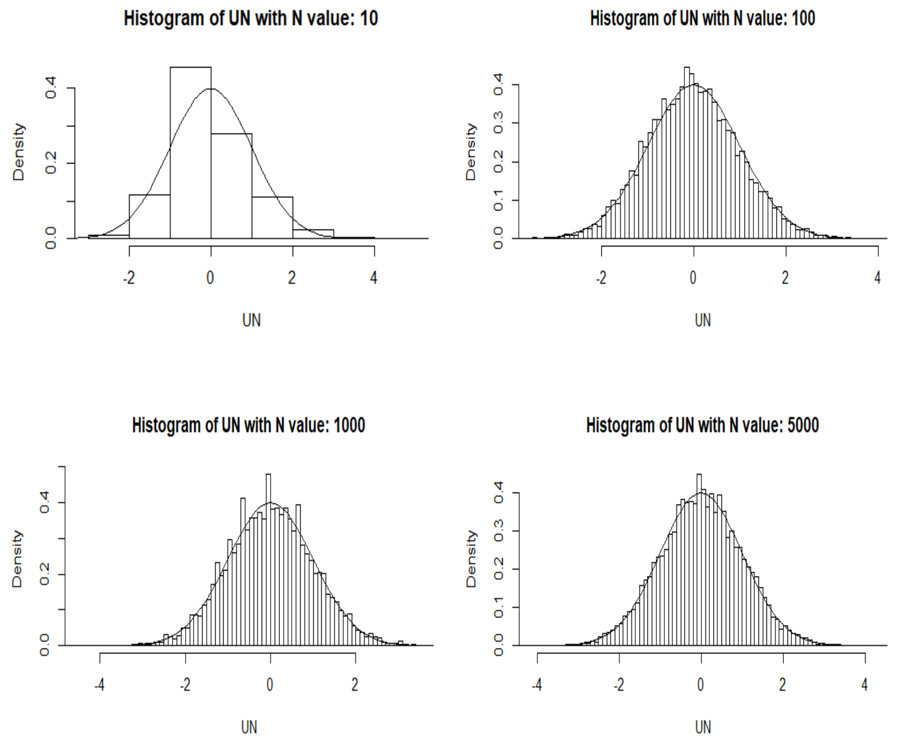


Figure 11: Histogram of UN for increasing sample sizes

We can also then quantify how good of an approximation each U_N is for a standard normal distribution using the Jarque-Bera test which tests for normality. The null hypothesis is that the data is from a normal distribution, whereas the alternative hypothesis is that it's not. The statistic can be calculated as follows whereby a value close to zero indicates a better fit:

$$JB = \frac{n}{6}\left(S^2 + \frac{1}{4}(K-3)^2\right)$$

where S is the empirical skewness and K is the empirical kurtosis.

From the figures above it is clear that common results for the JB statistic are close to zero, meaning that we can conclude that U_N follows a standard normal distribution. Rather than calculating manually, we can also perform the JB test with a function from the tseries R package where we choose a criterion of $\alpha = 0.05$ with the null hypothesis being that U_N is distributed according to the standard normal distribution and of course the alternative being that it's not. The p-values that are produced are 2.2e-16, 0.0006, 0.14 and 0.48 respective to our N values of 10, 100, 1000 and 5000. The statistics that are produced are also 158, 14, 4 and 1.5 so we can clearly see that as N increases, the better the fit. The first two p-values are below the criterion hence we must reject the null hypothesis in favour of the alternative that U_{10} and U_{100} do not follow a standard normal distribution. However, U_{1000} and U_{5000} can be considered to be standard normal variables.

The code below details the implementation:

```
print(jarque.bera.test(UN))
lr <- diff(UN)
mlr <- mean(lr)
stdlr <- sd(lr)
JBstat <- rep(NA, length(UN))
```

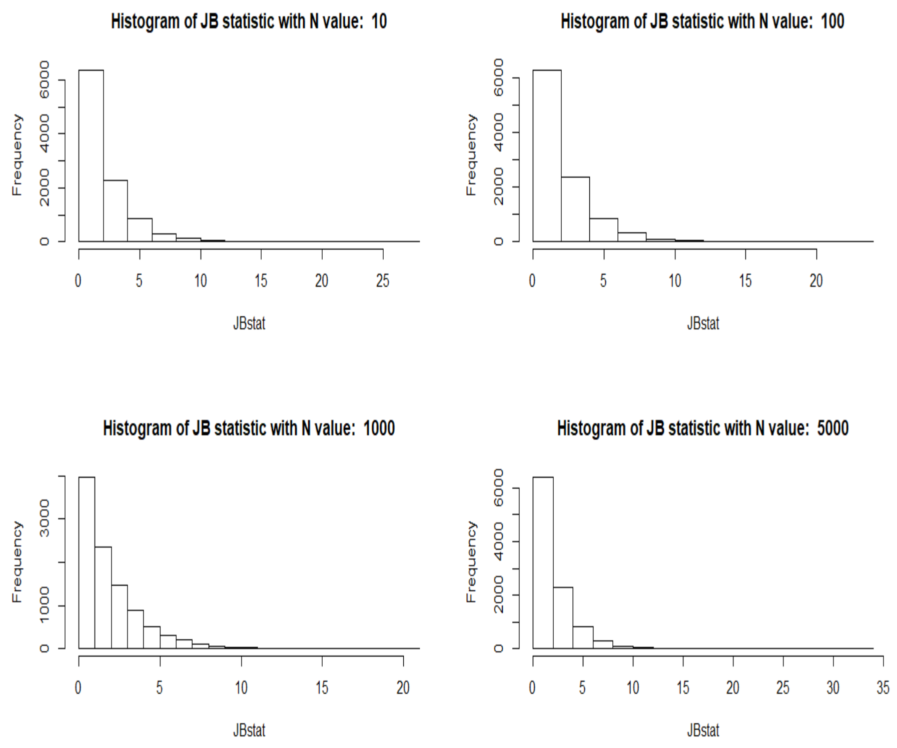


Figure 12: Jarque-Bera test for normality of U_N as N increases.

```
number <- length(lr)
for (i in 1:n) {
  lrth <- rnorm(number,mean=mlr,sd=stdlr)
  JBstat[i] <- number*(skewness(lrth)^2+(kurtosis(lrth)-3)^2/4)/6
}
hist(JBstat, main = paste("Histogram of JB statistic with N value: ", N, sep = " "))
}
```