correction

Question 1

1.a

The question requested to parametrize the model with the least possible amount of parameters. For both cases, you rightfully specified that 7 and 9 different update/jumps could be obtained respectively, given a current position on the graph.

However, the “least possible amount” was not covered as you did not parametrize each jump with its respective probability (e.g. $P_{a\to d}$, $1 - P_{a\to d}$, etc.) where it is possible to reduce the number of *actual* parameters from 7 to 4 as 3 parameters can be expressed in terms of the others for model 1 (and from 9 to 6 for model 2).

No mention of initialization state is made.

The graphs are correct.

The code was a nice addition.

1.b

Your code has several issues:

1. Print statements are obscuring the output when you run the function
2. You are pre-computing *multivariate* random *binomial* vectors from which to compute a markov chain X. This goes against the Markov Chain methodology as seen in class (the methodology requests that a uniform univariate function is drawn at each step in order to compute the next step). As such your implementation does not respect the Markovian definition
3. Your comment for when you are in the state drowsy is erroneous, from there summing Z and W can results in 3 values 0, 1, and 2 instead of 1 and 0 as you stated – the result is correct

The best practice would have to have declared your markov chain as a list of character A, D, or S and your markov chain an appendable vector that you would have incremented at each computed step – a much cleaner approach. However, as is the output still seems to be valid.

```
MarkovChain_1 = function(p_01, p_10, p_12, p_21, n, X_0){
  ############################
  ############################
  Y = rbinom(n,1, p_01)
  #print(Y) # <- Good practice: do not leave print statement like this in functions
  Z = rbinom(n,1, p_10)
  #print(Z) # <- Good practice: do not leave print statement like this in functions
  W = rbinom(n,1, p_12)
  #print(W) # <- Good practice: do not leave print statement like this in functions
  U = rbinom(n,1, p_21)
  #print(U) # <- Good practice: do not leave print statement like this in functions
  ############################
  ############################
  X <- c()
  for (i in 2:n){
  X[1] = X_0 #initializing state
      if (X[i-1] == 0){
        X[i] <- Y[i]
      }
      else if (X[i-1] == 1){ #we are in state drowsy
        #if we are in state drowsy, we can go back to sleep or awake, so we can go to
0 or to 1, and we can stay in 1
        X[i] <- Z[i] + W[i]
      }
      else if (X[i-1] == 2){
        X[i] <- U[i] + 1 #if U is 0, we stay in 2, if U is 1, we go back to 1
      }
  }
  return(X)
  }


############################
# I updated this part so that MarkovChain_1's output is stored in variable X
# not done in your code
############################
X = MarkovChain_1(0.5, 0.5, 0.5, 0.5, 10, 0)
############################
############################
```

Same observations for your model 2's simulation function.

```
MarkovChain_2 = function(p_01, p_10, p_12, p_21, p_02, p_20, n, X_0){
  ############################
  ############################
  Y = rbinom(n,1, p_01)
  print(Y) # <- ISSUE
  Z = rbinom(n,1, p_10)
  print(Z) # <- ISSUE
  W = rbinom(n,1, p_12)
  print(W) # <- ISSUE
  U = rbinom(n,1, p_21)
  print(U) # <- ISSUE
  P = rbinom(n,1, p_02)
  print(P)
  G = rbinom(n,1, p_20)
  print(G) # <- ISSUE
  ############################
  ############################
  X <- c()
  for (i in 2:n){
    X[1] = X_0 #initializing state
    if (X[i-1] == 0){
      X[i] <- Y[i] + P[i]
    }
    else if (X[i-1] == 1){ #we are in state drowsy
      #if we are in state drowsy, we can go back to sleep or awake, so we can go to 0
or to 1, and we can stay in 1
      X[i] <- W[i] + Z[i]
    }
    else if (X[i-1] == 2){
      X[i] <- U[i] + G[i] #if U is 0, we stay in 2, if U is 1, we go back to 1
    }
  }
  return(X)
}

MarkovChain_2(0.2, 0.9, 0.2, 0.1, 0.1, 0.8, 30, 0)
```

1.c

The computation is correct. However you don't seem to mention the outcome in the case of being in model 2 (you only solved for model 1).

Nice description of the recurrence condition, however.

1.d

Preliminatory note:

The first part of the code (the part you did not screen-capped in your pdf file) does not run as-is as you did not declare the variable n in the provided code.

Otherwise:

The functions you provide next (and reproduced in the PDF) do not compute the *convergence of the MLE* as requested but only a MLE at a given $n = 1000$ value. At this value, the convergence is not met (You could show convergence by repeating your process over a parameter range for n, e.g. from 10 to 100,000 to check whether your MLE parameter converges as n increases).

The code had some issues, corrected below:

```
#now let's make a function out of this and try to simulate it

amount_drowsy = function(X, n){
  ones <- c()
  for (k in 1:n){
    if (X[k] == 1){
      ones = c(ones, +1)
      #print(ones) # <- don't leave needless print statement
    }
  }
  return(n_ones = length(ones))
}


drowsy_to_sleep = function(X, n){
  one_to_zero <- c()
  for (j in 1:n){
    if (X[j-1] == 0 && X[j] == 1){
      one_to_zero = c(one_to_zero, +1)
      #print(one_to_zero) # <- don't leave needless print statement
    }
  }
  return(n_10 = length(one_to_zero))
}

p_10_hat = function(n_10, n_ones){
  return(n_10/n_ones)
}

#let's start the simulation with large n:

#First Case:

X = MarkovChain_1(0.5, 0.5, 0.5, 0.5, 1000, 0) # <- you need to record the output of
your function

test_len = length(X)
print(test_len)
```

```
## [1] 1000
```

```
n_ones = amount_drowsy(X, 1000) # <- you need to record the output of your function
n_10 = drowsy_to_sleep(X, 1000) # <- you need to record the output of your function

print(n_10)
```

```
## [1] 113
```

```
print(n_ones)
```

```
## [1] 496
```

```
p_10_hat = n_10/ n_ones
print(p_10_hat) # This is usually far from your true probability 0.5
```

```
## [1] 0.2278226
```

1.e

You mention figures but I didn't find them?

Question 2

2.a

The MLE is well stated. No implementation was provided however.

2.b

The MLE is well stated.

2.c

1. Good practice in R:

Your function:

```
Homo_Poisson = function(Tmax, lambda)
{
  t = 0 #t starts out with zero
  N = numeric()
  res = c()
  while(t < Tmax){ #we want the process to go on until the cumulative sum of t reache
s T, which is 1 here
    u = runif(1) #as long as the above is true, we create u, which is a uniform rando
m variable
    t = t - log(u)/lambda #then our new t will be this #it is the same thing as doing
it separately and then taking the cumulative sum of it.
    N = c(N, t)
  }
  res = c(N,"The a_hat is:",sum(N/Tmax), "the sum of N is:", sum(N))
  return(res)
}
```

On returning the output:

```
# you wrote:
res = c(N,"The a_hat is:",sum(N/Tmax), "the sum of N is:", sum(N))

#It is better to use a print statement to display those values and return a list as a
n output
cat("a_hat =",sum(N/Tmax), ", sum of n =", sum(N))
return(list(a_hat=sum(N/Tmax), sum_n=sum(N)))
```

2. On the simulation:

Stated by the teacher:

```
Your functions should:

1. Take in entry $T_{max}$ and $n$ and type 1 (for exp) and 2 (for sin)
2. The bound $M$ is n for type 1 and 2n for type 2
3. We make a step E(M ) and mark U([0, M ]) and :
  - if the mark is lower than the intensity, then the point is accepted
  -  otherwise, the point is rejected
```

One can simulate a poisson process by selecting a small timestep and drawing an exponential random variable with an adapted parameter λ on that timestep interval (here M). Alongside, we also draw a uniform random variable and check the result of this draw against the state of the underlying function (the rate function) to see whether we reject it or not.

As such your function lacked the check against the rate function itself and the poisson/exponential draw with parameter M.

1.d

Not completed

1.e

Not completed

Question 3

Not completed