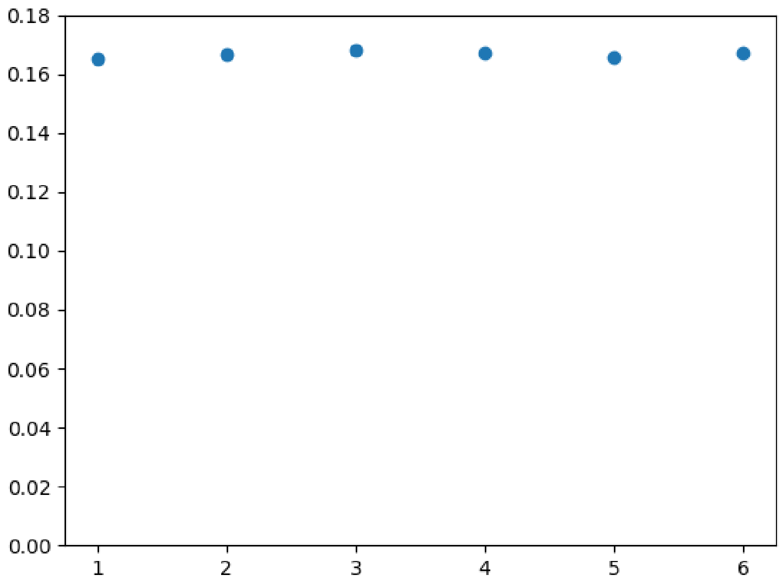
Assignment -2

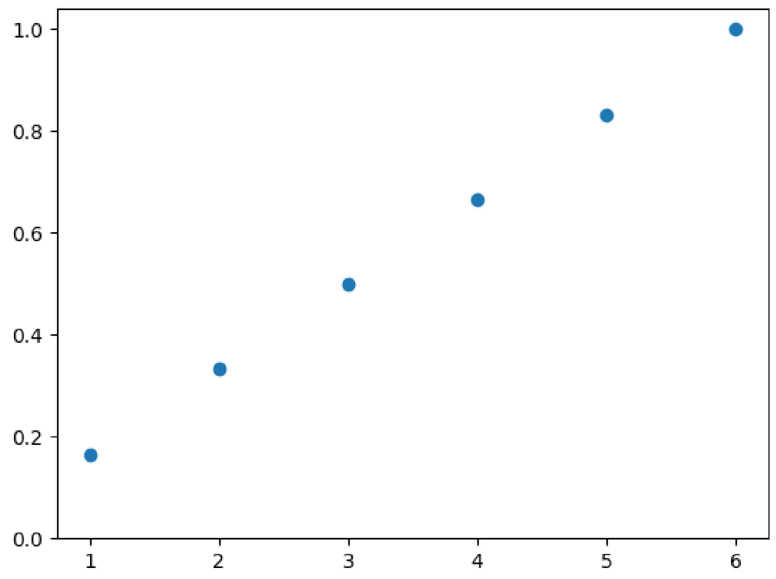
I did the assignment in Mathematica but the ASU Citrix receiver is not respondig and as a result I have lost my Mathematica notebook. So in quick response I did the whole assignment in Mathematica. Sorry for the trouble, it was beyond my control.

Q1- Fair dice roll.

Probability of getting any number in a fair dice roll is same which is also reflected in the graph. Also we expect the CDF to be 1 as well and we obtained the same result. Hence our simulation was successful.



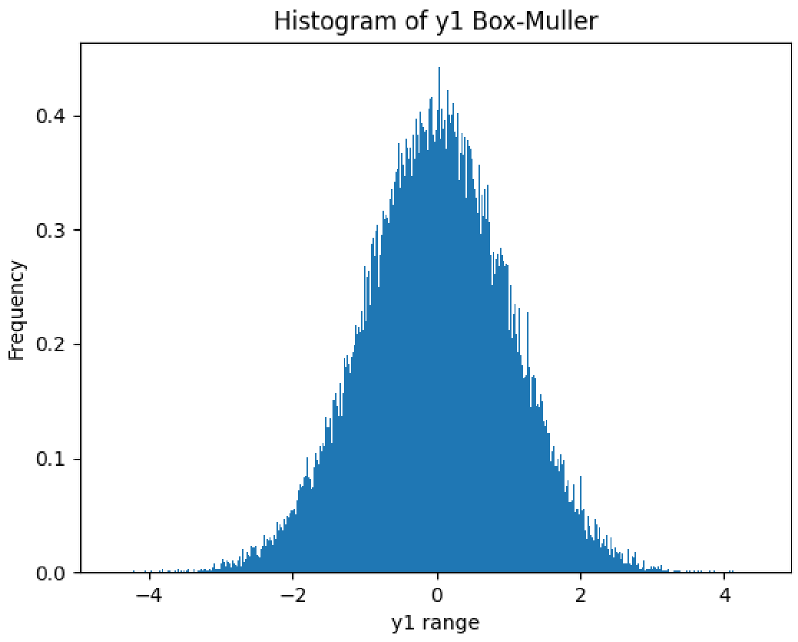
Probability of getting 1,2 . . . 6 in a dice roll can be found in di-



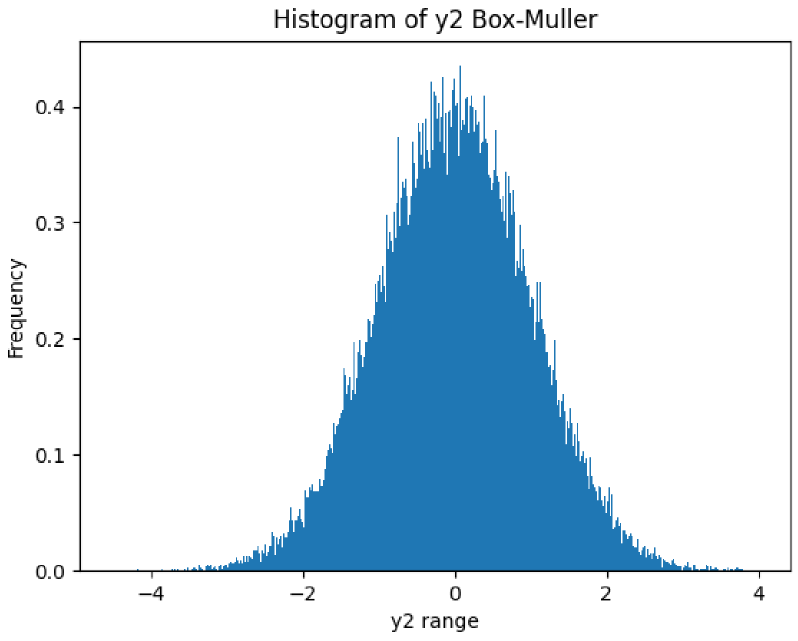
ceProb.png
CDF of the dice roll can be found in diceCDF.png

Q2 - Box-Muller Algorithm:

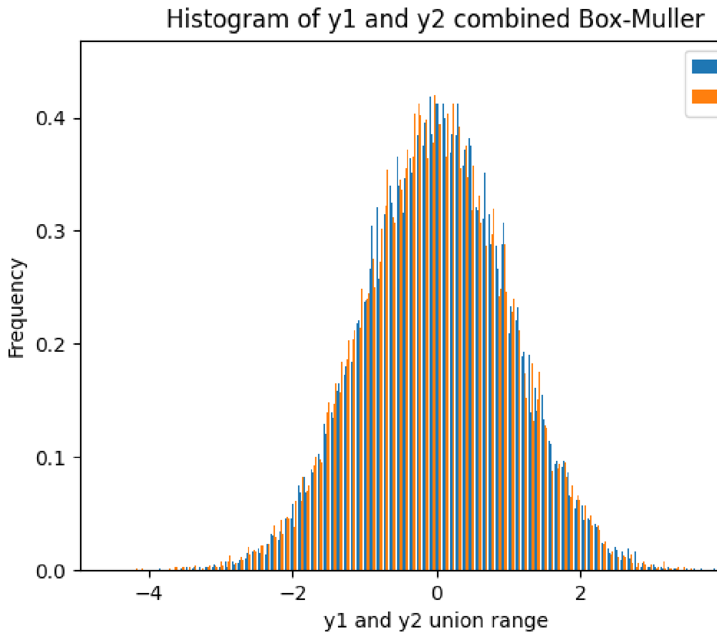
As can be seen from the histogram plot the mean is around 0 and almost 95.5% of the area is within 2 to -2 representing 2 sigma width. Within 3 to -3 equivalent to 3 sigma the whole graph is present. Hence our code is giving correct results.



The plot of the 1st sequence generated using Box-Muller is in y1.png



file.



2nd sequence plot is in y2.png file.
Both plotted together in y-combined.png file.

Q3 - Mass Action formula:

Solving the ODE in mathematica obtained the result

y(t) = b/a + c1*exp(-dt)

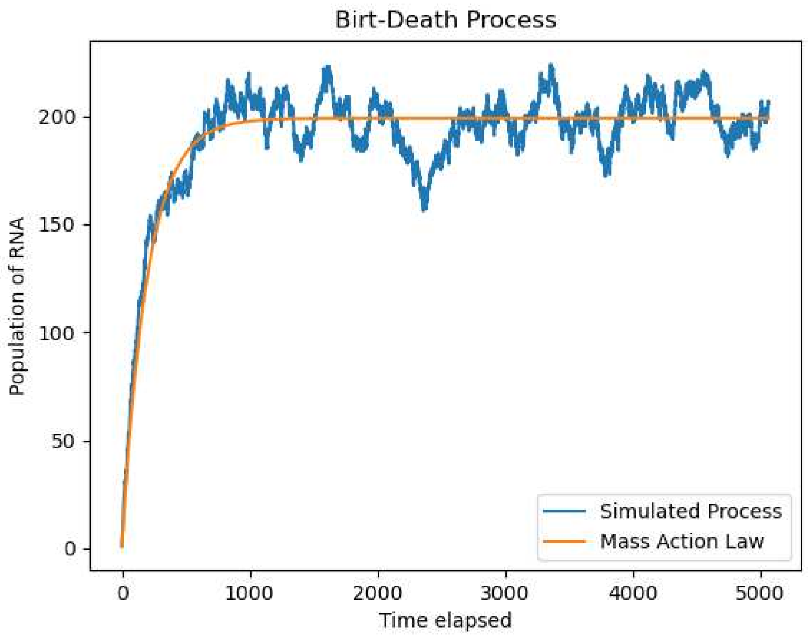
Using the initial condition y(0)=1, obtained

c1 = 1-b/a

Therefore the final equation is

y(t) = b/a + (1-b/a)*exp(-dt)

Here in the above equations b = birth rate and d = death rate.



The plot is in plot.png

```
import numpy as np; # Importing numpy for mathematical operations
from tqdm import tqdm; # Important tqdm for progress bar.
import matplotlib.pyplot as plt; #Plotting library

def fairDice(rolls): #Fair dice roll implementation
    rolls = int(rolls); # Number of rolls should be integer.
    p = np.zeros(6,dtype=float); #Initializing array to store probability
    cdf = np.zeros(6,dtype=float); # Storing the cdf of probability
    u = np.random.uniform(0,1,rolls); #Generating uniform random numbers from 0 to 1
    for i in tqdm(range(rolls)):#Looping over total number of rolls
        for j in range(6): #Looping over all the dice rolls
            if (u[i]<(j+1)/6 and u[i]>= j/6):#Boning the random uniform distribution
 into bins.
                p[j]+=1;#If within [j/6,(j+1)/6) then add 1 to bin

    p /=float(rolls); #Dividing all the bins with total roll gives the probability.
    cdf[0] = p[0]; #Initializing the first value of cdf with the first probability.
    for i in range(1,6):
        cdf[i] = cdf[i-1]+p[i];# creating the cdf

    return p,cdf #Returning the probability and the cdf array.

diceRoll = fairDice(100000) #Implement the fairDice function.
print("Probability of getting 1,2, .... 6 =",diceRoll[0]);#Printing the probability
print("CDF of dice roll getting 1,2 ... 6 =",diceRoll[1]);#Printing the cdf

x=np.linspace(1,6,6); #Generating x axis for the graph a linear array from 1 to 6
#Plotting the graph
plt.plot(x,diceRoll[0],'o');# Plotting probability.
plt.ylim(0,0.18);# Setting y axis ranging from 0 to 0.18
plt.savefig("diceProb.png",dpi=100); #Save the probability graph.
plt.clf(); #Clear plot canvas of the plot.
plt.plot(x,diceRoll[1],'o'); #Plotting cdf
plt.ylim(0,) #Starting y axis from 0
plt.savefig("diceCDF.png",dpi=100); #Save cdf plot.
```

```
import numpy as np; # Importing numpy module used for all the mathematical operation
s
import matplotlib.pyplot as plt; #Importing the plotting library.
from tqdm import tqdm; #Tqdm gives the progress bar

def box_muller(mean, sigma):#Box-Muller implementation.
    u1 = np.random.uniform(0,1); #Uniform sequence generator from 0 to 1.
    u2 = np.random.uniform(0,1); #Uniform sequence generator for the second box-mull
er series.
    z1 = np.sqrt(-2 * np.log(u1)) * np.cos(2 * np.pi * u2) #Box-muller final formula
 for series 1
    z2 = np.sqrt(-2 * np.log(u1)) * np.sin(2 * np.pi * u2) #2nd Box muller formula.
    return mean + sigma * z1, mean + sigma * z2 # Return both the box muller results
.

def multiple_box_muller(mean,sigma,number): #Multiple values generated from box mull
er algorithm.
    y1,y2 = np.zeros(number,dtype=float),np.zeros(number,dtype=float); #Initilizing
the y1 and y2 array for holding the Normal values generated from box-muller algorith
m.
    for i in tqdm(range(number)): #Looping to generating n numbers of values for nor
mal distribution from box-muller algorithm.
        y1[i],y2[i] = box_muller(mean,sigma); #Storing the box muller values.
    return y1,y2 # Return both the box-muller Normal sequences.
mean = 0 #mean is assumed to be 1.
sigma = 1 #Sigma is assumed to be 1.
data = multiple_box_muller(mean,sigma,100000); # Generate 1e5 Normal distribution va
lues

#Plotting the 1st distribution generated from Box-Muller algorithm.
plt.hist(data[0],density=True,bins=1000); #Plot a histogram from the 1st Box-Muller
distribution.
plt.xlabel("y1 range"); # X-axis label
plt.ylabel("Frequency");# Y-axis label
plt.title("Histogram of y1 Box-Muller"); # Title of the plot.
plt.savefig("y1.png",dpi=100); #Saving the plot.
plt.show(); #Display the plot

#Plotting the 2nd distribution from the Box-Muller algorithm.
plt.hist(data[1],density=True,bins=1000); # Comments same as above.
plt.xlabel("y2 range");
plt.ylabel("Frequency");
plt.title("Histogram of y2 Box-Muller");
plt.savefig("y2.png",dpi=100);
plt.show();

#Plotting both the values together.
plt.hist(data,density=True,bins=1000);
plt.xlabel("y1 and y2 union range");
plt.ylabel("Frequency");
plt.title("Histogram of y1 and y2 combined Box-Muller");
plt.legend(["y1","y2"]);
plt.savefig("y-combined.png",dpi=100);
plt.show();
```

```
import random #random number generator
import numpy as np #numpy library for mathematical operation
import matplotlib.pyplot as plt; #Plotting library
import time; #current system time
from tqdm import tqdm; #progress bar

time = int(time.time()); #calling the system time.
np.random.seed(time); # using the system time as random seed.

def bernoulli_draw(p):
    # Generate a random number between 0 and 1
    x = random.random()

    # Check if the random number is less than p
    if x < p:
        return 1
    else:
        return -1

def simulate_birth_death_process(birth_rate, death_rate, s0, size): # Function for b
irth-death process
    if (s0<0): # making sure initial population is greater than 1
        print("Initial population must be greater than 0"); #warning message
        quit(); #terminate the program
    if (birth_rate+death_rate !=1): #p+q should be 1
        print("Birth Rate + Death Rate should be 1");# warning message
        quit(); #terminate the program
    s0 = int(s0); #confirm initial population is integer
    time = np.zeros(size,dtype="float"); #initializing  time array to store the cdf
of waiting time from one state to another.
    s = np.zeros(size,dtype="float"); # initializing the s array that will contain t
he population of RNA at different time points.
    s[0] = s0; # The 0th position is the initial value.

    for i in tqdm(range(1,size)): # Looping from 1st position to 1-size steps. TQDM
is used to generate the progress bar.
        # events = np.random.choice([1, -1], size=1, p=[birth_rate / (birth_rate + s
[i-1]*death_rate), s[i-1]*death_rate / (birth_rate + s[i-1]*death_rate)]); # In-buil
t bernoulli draw.
        events = bernoulli_draw(birth_rate / (birth_rate + s[i-1]*death_rate)); # My
 bernoulli draw function.
        s[i] = s[i-1] + events; # Increasing or decreasing the population accourding
 to bernoulli draw.
        if s[i] <0: # Test case, population can't be less than 0.
            s[i]=0; # even if it goes to 0 due to some error set it to 0
            print("Crazy"); #Warning something wrong.
        time[i] = np.random.exponential(scale=1/(birth_rate + s[i-1]*death_rate),siz
e=1); #smapling the time from an exponential distribution.
        time[i] += time[i-1]; # CDF of the time.

    return time,s; # Return the time and population array.


x,y = simulate_birth_death_process(0.995,0.005,1,10000); #Call the function.
plot = (0.995/0.005)+(1-(0.995/0.005))*np.exp(-0.005*x); # Plot the mass action law,
 please look into the mathematica notebook for the final formula.
# print(x,y); # Prinint x and y for debugging.
plt.plot(x,y); # Plot the time and the population array.
plt.plot(x,plot); # Plot time and the mass action law to verify our results.
plt.xlabel("Time elapsed"); # X-axis label of the plot
plt.ylabel("Population of RNA");# Y-axis label of the plot.
plt.title("Birt-Death Process"); # Plot title.
plt.legend(["Simulated Process","Mass Action Law"]); # Plot legend.
plt.savefig("plot.png",dpi=100); # Saving the plot into a file plot.png with 100 dot
 per inch resolution.
# plt.show(); # Would show the plot in qt window but it is not necessary.
```