IS 609 Course Project

Shipra Ahuja, Ben Arancibia, Xingjia Wu

December 3, 2015

Team Members and Problems

Shipra Ahuja - Chapter 11 Section 5 Project 5

Ben Arancibia - Chapter 5 Section 3 Project 3

Xingjia Wu - Chapter 9 Section 3 Project 4

Chapter 5 Section 3 Project 3

Problem

Craps - Construct and perform a Monte Carlo simulation of the popular casino game of craps. The rules are as follows:

There are two basic bets in craps, pass and don't pass. In the pass bet, you wager that the shooter (the person throwing the dice) will win; in the don't pass bet, you wager that the shooter will lose. We will play by the rule that on an initial roll of 12 ("boxcars"), both pass and don't pass bets are losers. Both are even-money bets.

Conduct of the game: Roll a 7 or 11 on the first roll: Shooter wins (pass bets win and don't pass bets lose). Roll a 12 on the first roll: Shooter loses (boxcars; pass and don't pass bets lose). Roll a 2 or 3 on the first roll: Shooter loses (pass bets lose, don't pass bets win).

Roll 4, 5, 6, 8, 9, 10 on the first roll: This becomes the point. The object then becomes to roll the point again before rolling a 7. The shooter continues to roll the dice until the point or a 7 appears. Pass bettors win if the shooter rolls the point again before rolling a 7. Don't pass bettors win if the shooter rolls a 7 before rolling the point again.

Write an algorithm and code it in the computer language of your choice. Run the simulation to estimate the probability of winning a pass bet and the probability of winning a don't pass bet. Which is the better bet? As the number of trials increases, to what do the probabilities converge?

Solution

Craps is a game where two dice are rolled amd the dice rolls events are independent of each other. Possible totals from rolling two dice are seen in the table below.

	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

To roll a 7 or 11, which is defined as shooter wins, on the first roll is 8/36.

to roll a 2, 3, or 12, which is defined as shooter loses, on the first roll is 4/36.

There are two types of bets that we are dealing with in this problem - Pass and Don't Pass. In the Pass bet, the gambler wins only when Shooter wins and in Don't Pass bet the gambler wins only when Shooters loses except the Boxcars (Roll a 12 on the first roll).

The question asks for a simulation to estimate the probability of winning a pass bet and the probability of winning a don't pass bet. Bellow is the code for simulation of winning a pass bet.

```
#pass bet
N = 10000
data = {}
for (i in 1:N){
  stop = FALSE
  shooter = sample(1:6,1) + sample (1:6,1)
  if (shooter == 12){
    data[i] = 0
    stop = TRUE
  }
  else if ((shooter == 7)||(shooter==11)){
    data[i] = 1
    stop = TRUE
  }
  else if ((shooter == 2)||(shooter==3)){
    data[i]=0
    stop = TRUE
  }
  else{
    point = shooter
  }
  repeat{
    if (stop){
      break
    }
    shooter = sample(1:6,1) + sample (1:6,1)
    if (shooter == 7){
      data[i]=0
      stop = TRUE
    }
    else if (shooter == point){
      data[i]=1
      stop = TRUE
    }
  }
}

#Probability of winning when you are a pass bet
mean(data)
```

```
## [1] 0.4973
```

The probability of winning a pass bet is:

```
(mean(data))
```

```
## [1] 0.4973
```

Now a don't pass bet simulation is run and the probability of winning is examined.

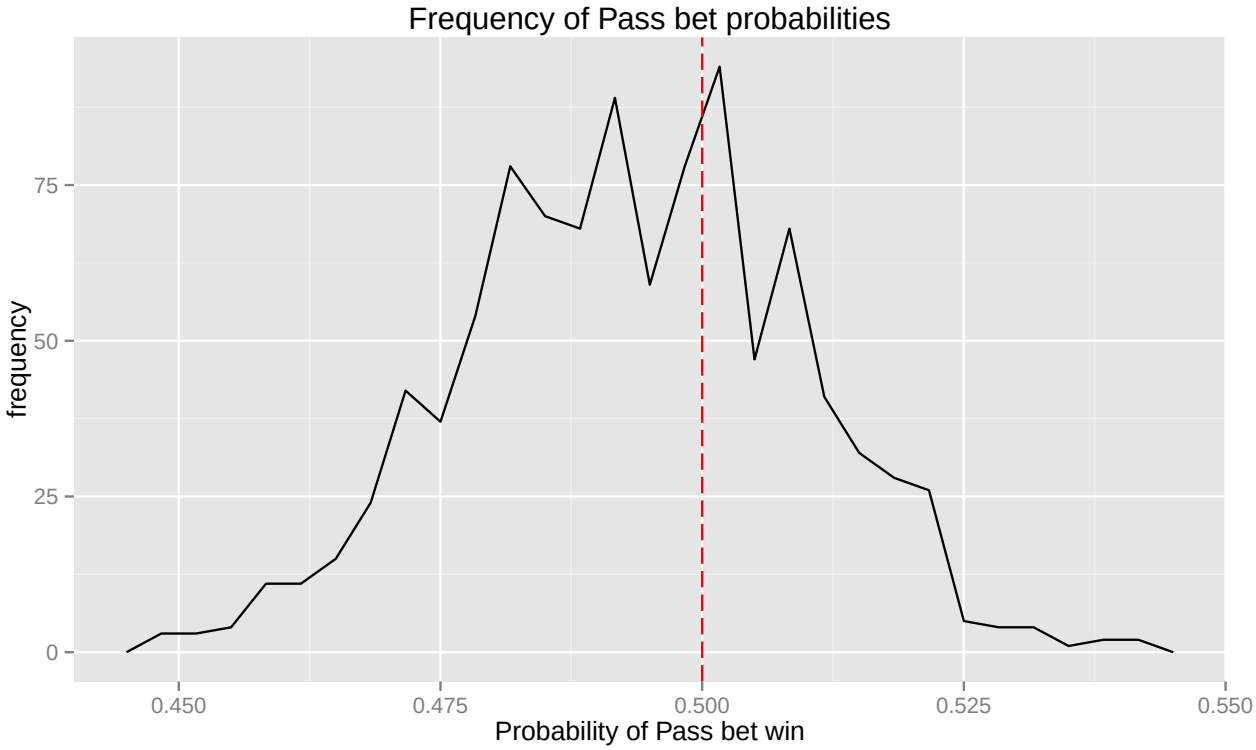
```
#no-pass bet
N = 10000
data = {}
for (i in 1:N){
  stop = FALSE
  shooter = sample(1:6,1) + sample (1:6,1)
  if (shooter == 12){
    data[i] = 0
    stop = TRUE
  }
  else if ((shooter == 7)||(shooter==11)){
    data[i] = 0
    stop = TRUE
  }
  else if ((shooter == 2)||(shooter==3)){
    data[i]=1
    stop = TRUE
  }
  else{
    point = shooter
  }
  repeat{
    if (stop){
      break
    }
    shooter = sample(1:6,1) + sample (1:6,1)
    if (shooter == 7){
      data[i]=1
      stop = TRUE
    }
    else if (shooter == point){
      data[i]=0
      stop = TRUE
    }
  }
}

#Probability of winning when you are a no-pass bet
mean(data)
```

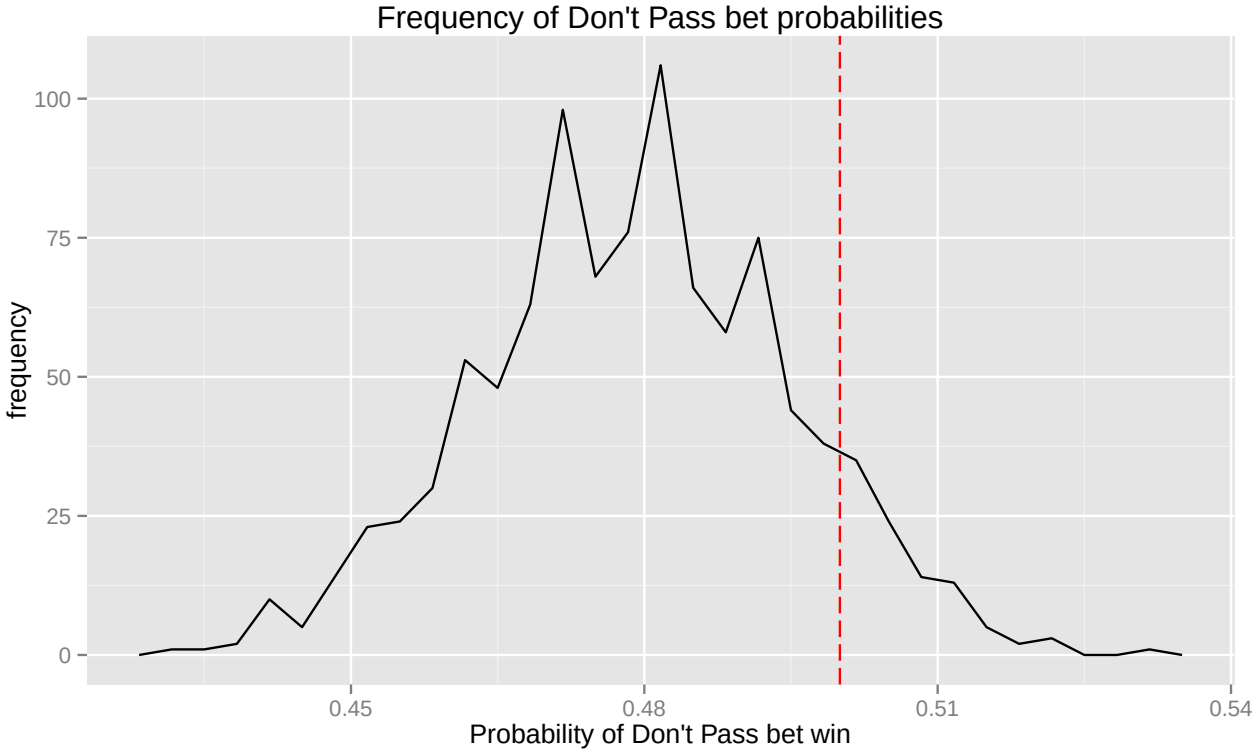
```
## [1] 0.4842
```

The probability of winning a don't pass bet is:

```
(mean(data))
```

```
## [1] 0.4842
```





Conclusion

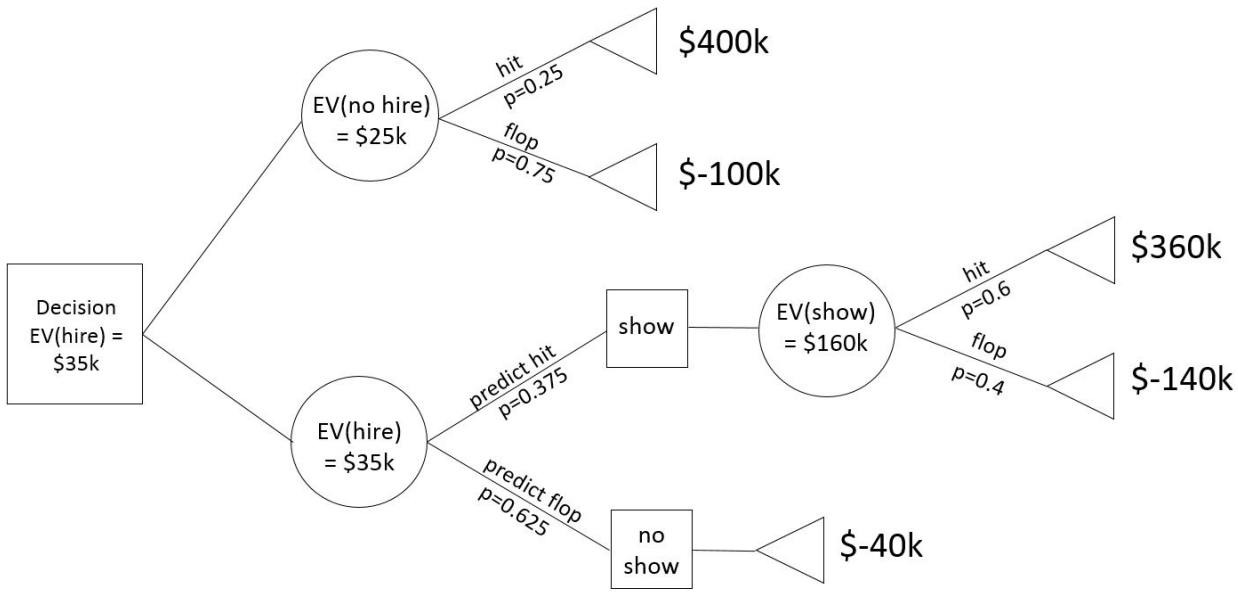
Craps is the best game for gamblers to play in a Casino, as long as the player does not play a large number of iterations. In the long run the casino will win because the probability of winning is little bit less than 50%. The better bet is the when the player makes a Pass Bet. The probabilities converage to around 49% as the number of trials increased.

Chapter 9 Section 3 Project 4

The NBC TV network earns an average of $400,000 from a hit show and loses an average of $100,000 on a flop (a show that cannot hold its rating and must be canceled). If the network airs a show without a market review, 25% turn out to be hits, and 75% are flops. For $40,000, a market research firm can be hired to help determine whether the show will be a hit or a flop. If the show is actually going to be a hit, there is a 90% chance that the market research firm will predict a hit. If the show is going to be a flop, there is an 80% chance that the market research will predict the show to be a flop. Determine how the network can maximize its profits over the long haul.

Decision tree diagram

A decision tree diagram is shown below.



The probabilities are queried from built network using gRain package.

Build network

```
source("http://bioconductor.org/biocLite.R")
```

```
## Bioconductor version 3.0 (BiocInstaller 1.16.5), ?biocLite for help
## A new version of Bioconductor is available after installing the most
##   recent version of R; see http://bioconductor.org/install
```

```
biocLite(c('Rgraphviz','gRbase','gRain','RBGL'))
```

```
## BioC_mirror: http://bioconductor.org
## Using Bioconductor version 3.0 (BiocInstaller 1.16.5), R version 3.1.2.
## Installing package(s) 'Rgraphviz' 'gRbase' 'gRain' 'RBGL'
```

```
##
## The downloaded binary packages are in
##  /var/folders/6l/wn3v67_128jckf4kljt7td8w0000gn/T//RtmpNGEBvN/downloaded_packages
```

```
suppressWarnings(suppressMessages(library(Rgraphviz)))
suppressWarnings(suppressMessages(library(gRain)))


hf <- c("hit", "flop")
phf <- c("p.hit", "p.flop")

# Specify the Conditional Probability Tables
show <- cptable(~show, values=c(25, 75), levels=hf)
predict <- cptable(~predict|show, values= c(0.9, 0.1, 0.2, 0.8), levels=phf)

# Compile plist
plist <- compileCPT(list(show, predict))
summary(plist)
```

```
## $show
## show
##  hit flop
## 0.25 0.75
##
## $predict
##         show
## predict  hit flop
##   p.hit  0.9  0.2
##   p.flop 0.1  0.8
```

```
# Build the network
net <- grain(plist)
```

Query the network to get probabilities

- The probability that the market review predicts a hit

```
q1 <- setFinding(net, nodes="predict", states=c("p.hit"))
(phit <- pFinding(q1))
```

```
## [1] 0.375
```

There is probability of 0.375 that the market review will predict a hit. In this condition, the show will be on.

- The probability the market review predicts a flop

```
q2 <- setFinding(net, nodes="predict", states=c("p.flop"))
(pflop <- pFinding(q2))
```

```
## [1] 0.625
```

There is probability of 0.625 that the market review will predict a flop. In this condition, the show will be off.

- For the show condition (which the market review predicts a hit), we need to find out the probability the show actually turns out to be a hit or a flop.

```
(querygrain(q1, nodes = "show", type="marginal"))
```

```
## $show
## show
##  hit flop
##  0.6  0.4
```

When market review predicts a hit, the probabilities that a show actually turns out to be hit or flop are 0.6 and 0.4, respectively.

Calculate the expected profits

- Calculate the expected profit of airing show

```
hit <- 400
flop <- -100
review <- -40
(show.profit <- 0.6*(hit+review)+0.4*(flop+review))
```

```
## [1] 160
```

The expected profit of airing show will be 160k.

- Calculate the expected profit of hiring a market review

```
noshow.loss <- review
(hire.profit <- phit*show.profit + pflop*noshow.loss)
```

```
## [1] 35
```

The expected profit of hiring a market review is 35k.

- Calculate the expected profit of not hiring market review

```
(no.hire.profit <- 0.25*hit +0.75*flop)
```

```
## [1] 25
```

The expected profit of not hiring market review is 25k.

Conclusion

Since the expected profit of hiring market review (35k) is higher than not hiring (25k), the network can hire a market review to maximize its profits over the long haul.

Chapter 11 Section 5 Project 5

Chapter 11 - Section 11.5 - Page 499 - Project # 5

The Spread of a Contagious Disease. Consider the following ordinary differential equation model for the spread of a communicable disease:

dN/dt = 0.25N(10 - N),N(0)=2 (a)

where N is measured in 100's. Analyze the behavior of this differential equation as follows.

Part A

Since this is an autonomous differential equation, perform a qualitative graphical analysis as discussed in Section 11.4.

Solution

$\frac{dN}{dt}$ = 0.25N(10-N)

We have equibrium points as N=0 and N=10

Using the phase line approach, we get following conditions -

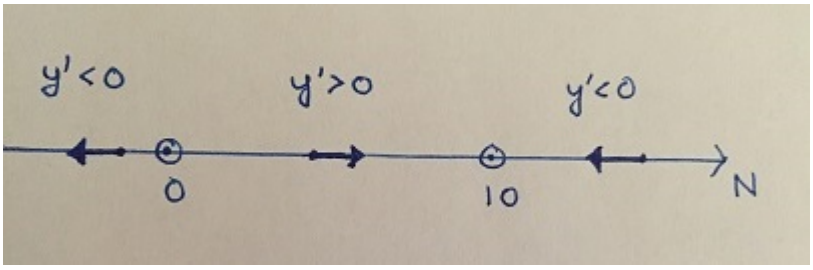
When N > 10,y' < 0

When N < 10,y' > 0

When N > 0,y' > 0

When N < 0,y' < 0

Phase Line Here



Taking the second derivative of the differential equation, we get

$\frac{dN}{dt} = 0.25N(10\text{-}N)$

$\frac{dN}{dt} = 2.5N - 0.25N^2$

$d^2N/dt^2 = d/dt(2.5N - 0.25N^2)$

$= 2.5dN/dt - 0.5NdN/dt$

$= 2.5y' - 0.5Ny'$

$= (2.5 - 0.5N)y'$

$= (2.5 - 0.5N)(0.25N)(10-N))$

We have equilibrium points as

N=0, N=10, N=5

Therefore,

When $N > 0$, $y'' > 0$

When $N < 0$, $y'' < 0$

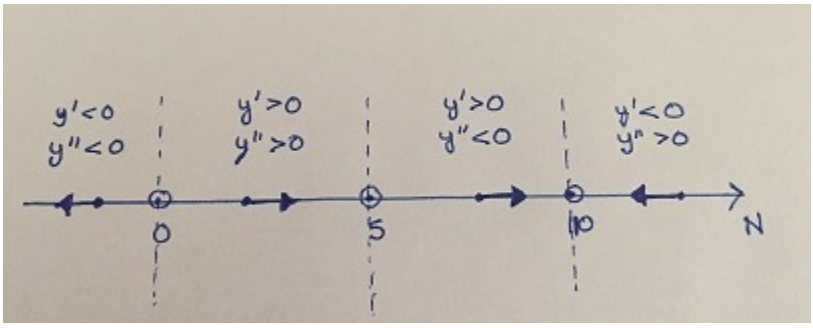
When $N > 5$, $y'' < 0$

When $N < 5$, $y'' > 0$

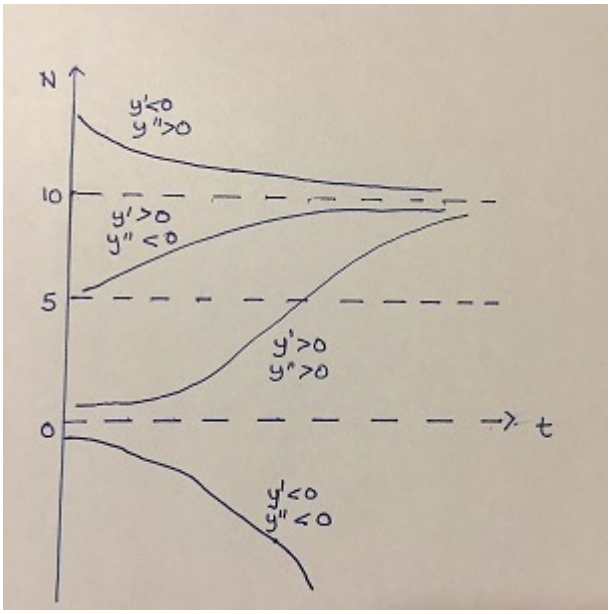
When $N > 10$, $y'' > 0$

When $N < 10$, $y'' < 0$

Phase Line Here



Solution Curve based on Phase Line



Part 1

Plot dN/dt versus N: Find and label all rest points (equilibrium points).

Solution

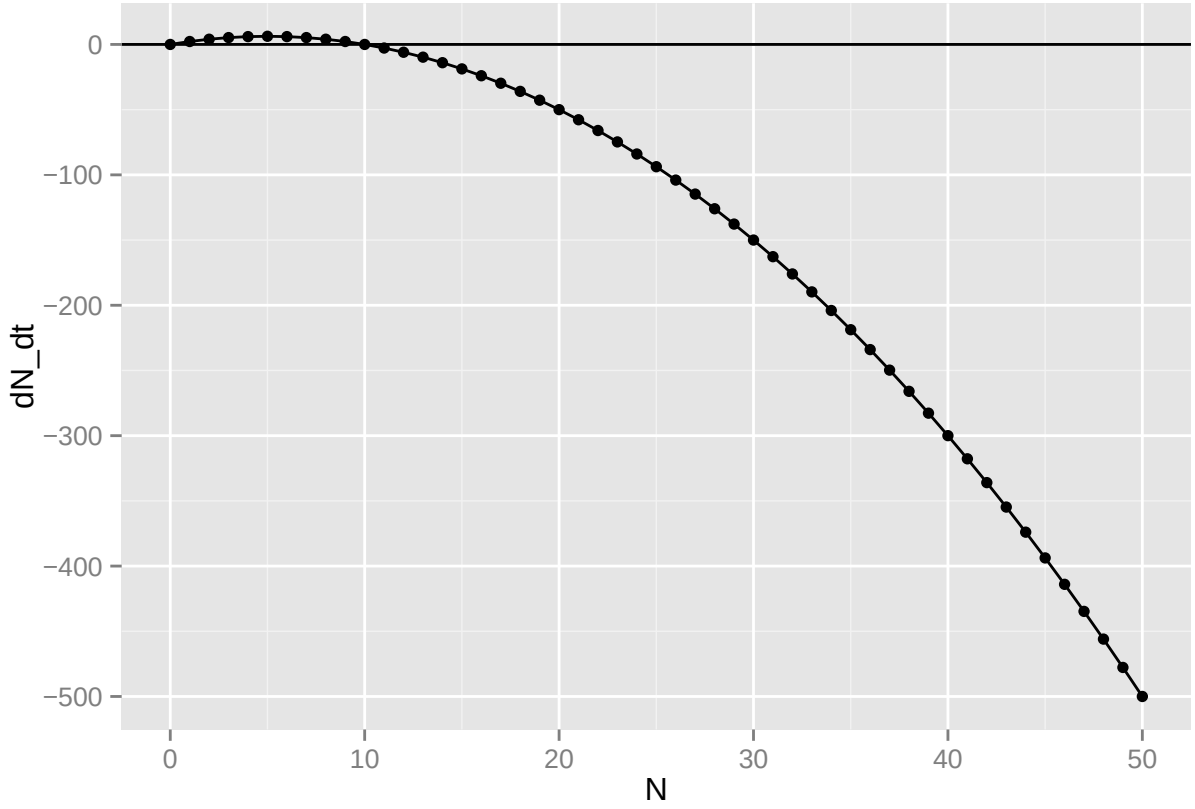
```
suppressWarnings(suppressMessages(library(ggplot2)))

df <- data.frame()

for(N in 0:50)

  {
   dN_dt <- ((0.25*N) * ((10 - N)))

   colbind <- cbind(N,dN_dt)

   df <- rbind(df,colbind)

  }

knitr::kable(df)
```

N	dN_dt
0	0.00
1	2.25
2	4.00
3	5.25

N	dN_dt
4	6.00
5	6.25
6	6.00
7	5.25
8	4.00
9	2.25
10	0.00
11	-2.75
12	-6.00
13	-9.75
14	-14.00
15	-18.75
16	-24.00
17	-29.75
18	-36.00
19	-42.75
20	-50.00
21	-57.75
22	-66.00
23	-74.75
24	-84.00
25	-93.75
26	-104.00
27	-114.75
28	-126.00
29	-137.75
30	-150.00
31	-162.75
32	-176.00
33	-189.75
34	-204.00
35	-218.75
36	-234.00
37	-249.75
38	-266.00
39	-282.75

N	dN_dt
40	-300.00
41	-317.75
42	-336.00
43	-354.75
44	-374.00
45	-393.75
46	-414.00
47	-434.75
48	-456.00
49	-477.75
50	-500.00

```
p <- ggplot(df,aes(x=N,y=dN_dt)) + geom_point() + geom_line() + geom_hline(yintercept=0)

print(p)
```



Equilibrium point is at N=0 and N=10 because $\frac{dN}{dt}$=0 at N=0 and N=10

Part 2

Estimate the value where the rate of change of the disease is the fastest. Justify your answer.

Solution

Rate of change of disease is the fastest at N=5 because dN/dt has the highest value at N=5 that is dN/dt = 6.25(from the data points above) at N=5.

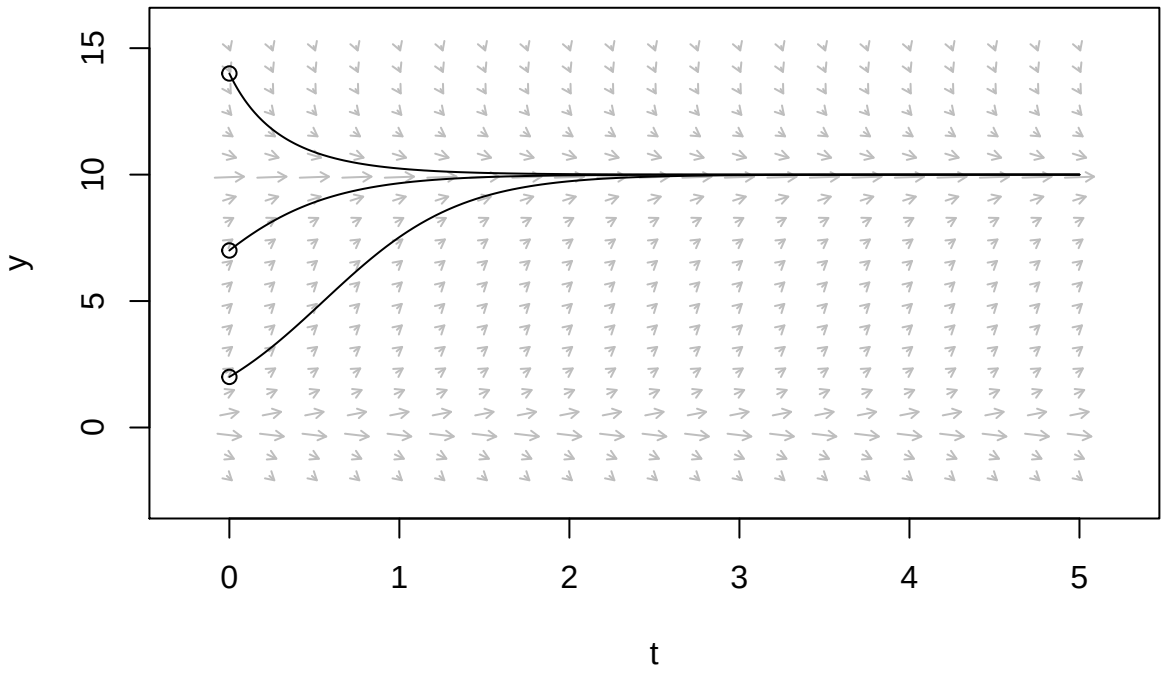
Part 3

Plot N versus t for each of the following initial conditions:

$N(0) = 2$, $N(0) = 7$, $N(0) = 14$

Solution

```
suppressWarnings(suppressMessages(library(phaseR)))

fx <- function(t,y,parameters)
{
  dy <- 0.25*y*(10-y)
  list(dy)

}


fx.flowField <- flowField(fx, x.lim = c(0, 5), y.lim = c(-2, 15),parameters=NULL,points=21,system = "on


fx.trajectory <-
  trajectory(fx, y0 = c(2,7,14), t.end = 5,
             parameters = NULL, system = "one.dim", colour = rep("black", 3))
```



Part 4

Describe the stability of each rest point (equilibrium point).

Solution

N=0 is unstable because the arrows in the phase line move away from the point N=0 and consequently solution curves also move away from the point N=0.

N=10 is stable because the arrows in the phase line move towards the point N=10. Solution curves also move towards N=10.

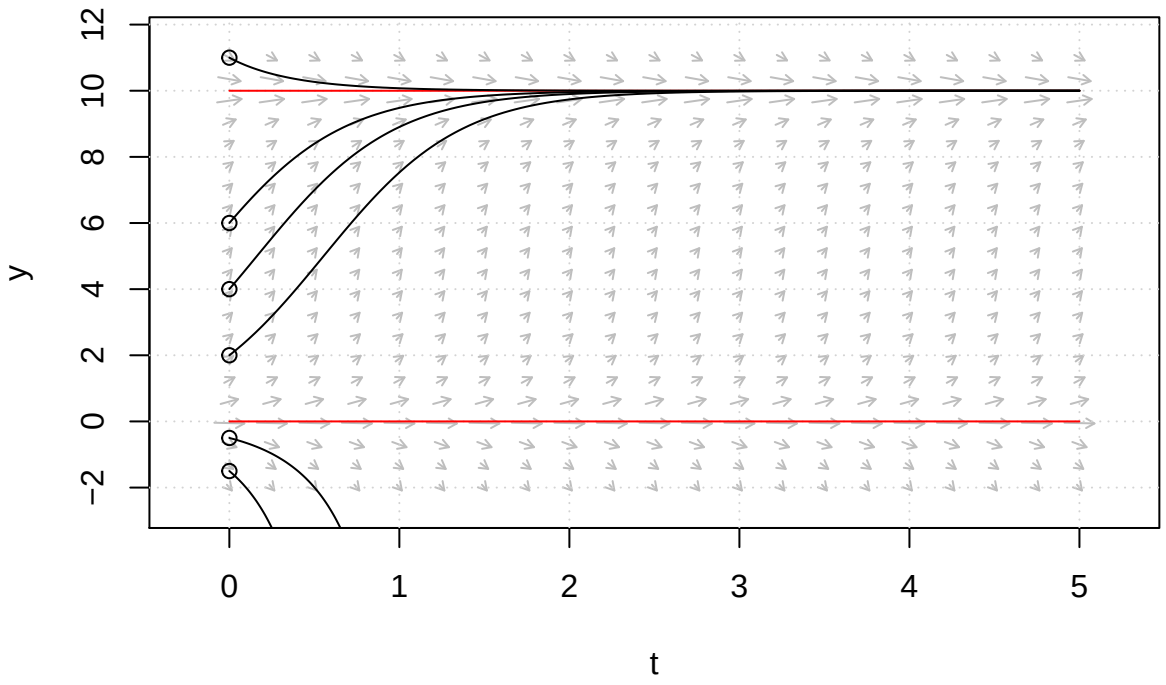
Part B

Obtain a slope field plot of this differential equation. Briefly analyze the slope field plot. Compare it to your qualitative plot in part (a) above.

Solution

```
suppressWarnings(suppressMessages(library(phaseR)))

fx <- function(t,y,parameters)
{
  dy <- 0.25*y*(10-y)
  list(dy)

}
```

```
fx.flowField <- flowField(fx, x.lim = c(0, 5), y.lim = c(-2, 11),parameters=NULL,points=21,system = "one

grid()

fx.nullclines <-
  nullclines(fx, x.lim = c(0, 5), y.lim = c(-2, 11),
             parameters = NULL, system = "one.dim")


fx.trajectory <-
  trajectory(fx, y0 = c(-0.5,-1.5,2,4,6,11), t.end = 5,
             parameters = NULL, system = "one.dim", colour = rep("black", 6))
```



From the short line segments slopes as well as trajectories, it shows that points N=10 is stable as all small line segments move towards N=10 and point N=0 is unstable as all line segments move away from N=0.

Stability of the points can also be demonstrated as -

```
# Determine stability for N=0

fx.stability.0 <- stability(fx, y.star = 0, parameters = NULL, system = "one.dim")
```

```
##
## Discriminant: 2.5   Classification: Unstable
```

```
# Determine stability for N=10

fx.stability.10 <- stability(fx, y.star = 10, parameters = NULL, system = "one.dim")
```

```
##
## Discriminant: -2.5   Classification: Stable
```

The above shows as well that point N=0 is unstable and point N=10 is stable.

The slope field matches the qualitative plot in Part(a) above. It matches the directions of the phase lines well as the solution curves created above.

Part C

Solve this differential equation using the separation of variables technique. Plot the solution over the interval of time from [0, 10]. How does your actual solution compare to the qualitative graphical solution?

Solution

Below is the actual solution

$\frac{dN}{dt}$ = 0.25N(10-N)

Using separation of variables -

$\frac{dN}{N(10-N)}$ = 0.25dt

Performing integration of LHS with respect to N and RHS with respect to t, we get

$\int \frac{dN}{N(10-N)} = \int 0.25dt$

$\frac{1}{10}$ ln(N) - $\frac{1}{10}$ ln(N-10) = 0.25t + C1

$\frac{1}{10}$ (ln(N) - ln(N-10)) = 0.25t + C1 .. Eqn(1)

Using Quotient Rule of natural logarithm -

ln(x/y) = ln(x) - ln(y)

Substitute ln($\frac{N}{N-10}$) in Eqn (1)

$\frac{1}{10}$ ln($\frac{N}{N-10}$) = 0.25t + C1

ln($\frac{N}{N-10}$) = 10(0.25t + C1)

ln($\frac{N}{N-10}$) = 2.5t + C .. Eqn (2)

Using properties of exponential functions $e^{ln(y)}$ = y

$e^{ln(\frac{N}{N-10})} = e^{2.5t+C}$

$e^{ln(\frac{N}{N-10})} = e^{2.5t}\ e^{C}$ (Assume e^{C} = K)

$\frac{N}{N-10} = Ke^{2.5t}$

$\frac{1}{Ke^{2.5t}} = \frac{N-10}{N}$

$\frac{1}{Ke^{2.5t}}$ = 1 - $\frac{10}{N}$

$\frac{10}{N}$ = 1 - $\frac{1}{Ke^{2.5t}}$

$\frac{10}{N} = \frac{Ke^{2.5t}-1}{Ke^{2.5t}}$

N = $\frac{10Ke^{2.5t}}{Ke^{2.5t}-1}$ ... Eqn(3)

When t=0, N = $\frac{10K}{K-1}$

```
# Compute K when t=0

# When t = 0, N = 10K/(K-1) so K= N/N-10

computeK <- function(N)
  {
    K <- N/(N-10)
    return(K)
  }

# Call function computeK when N(0)=2

k.N2 <- computeK(2)
print(paste0("Value of K for N(0)= 2 is ",round(k.N2,digits=3)))
```

```
## [1] "Value of K for N(0)= 2 is -0.25"
```

```
k.N7 <- computeK(7)
print(paste0("Value of K for N(0)= 7 is ",round(k.N7,digits=3)))
```

```
## [1] "Value of K for N(0)= 7 is -2.333"
```

```
k.N14 <- computeK(14)
print(paste0("Value of K for N(0)= 14 is ",round(k.N14,digits=3)))
```

```
## [1] "Value of K for N(0)= 14 is 3.5"
```

Substituting the value of K computed above in the Eqn(3) above we get

When K = -0.25, substituting value of K in Eqn(3) we get

$\mathbf{N} = \frac{-2.5e^{2.5t}}{-0.25e^{2.5t}-1}$ ... Eqn(4)

When K = -2.33, substituting value of K in Eqn(3) we get

$\mathbf{N} = \frac{-23.3e^{2.5t}}{-2.33e^{2.5t}-1}$ ... Eqn(5)

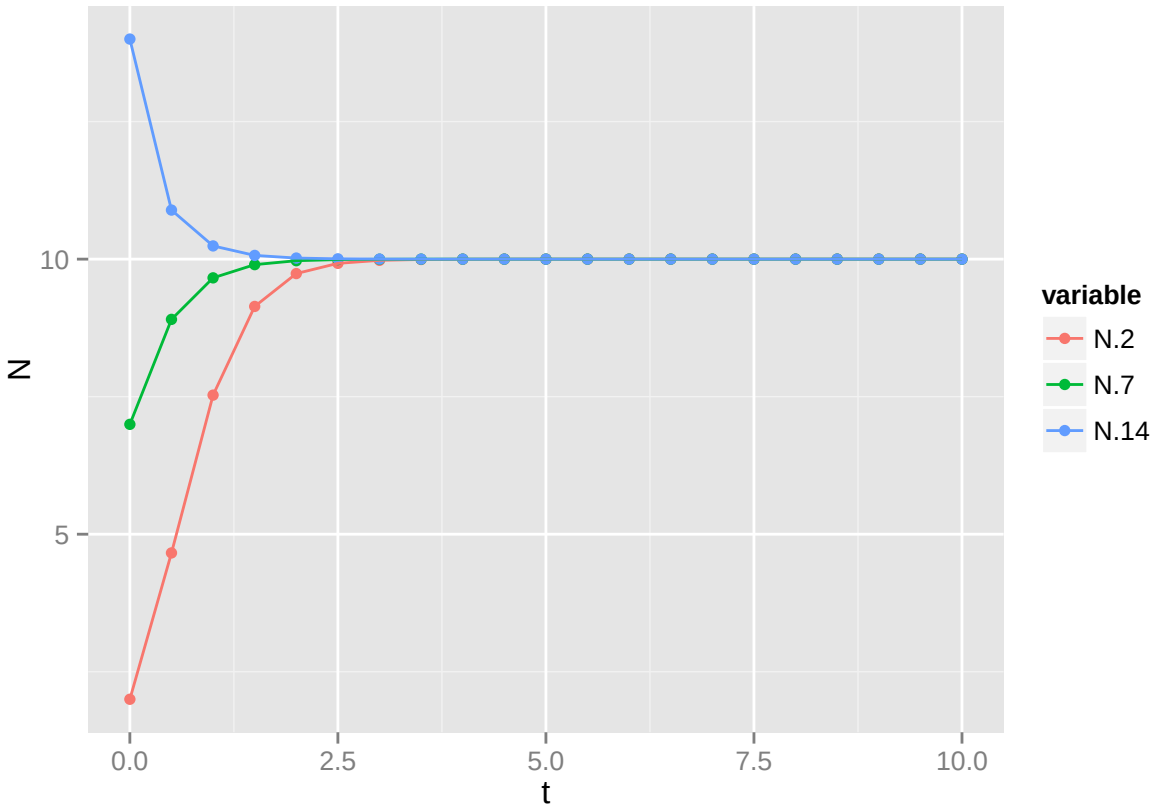
When K = 3.5, substituting value of K in Eqn(3) we get

$\mathbf{N} = \frac{35e^{2.5t}}{3.5e^{2.5t}-1}$ ... Eqn(6)

Find points to plot N vs t and plot N vs t for initial condition N(0)=-0.5,-1,2,7,14

```
suppressWarnings(suppressMessages(library(reshape2)))

# Compute data points for plotting N vs t plots for various initial values of N

df1 <- data.frame()   # Dataframe for points when N(0) = 2
df2 <- data.frame()   # Dataframe for points when N(0) = 7
df3 <- data.frame()   # Dataframe for points when N(0) = 14


# Loop to compute the points
```

```
for(t in seq(0,10,by=0.5))
  {
    # Data Points for N(0)=2
    N <- (-2.5 * exp(2.5*t))/((-0.25*exp(2.5*t)) - 1)
    colbind <- cbind(t,N)
    df1 <- rbind(df1,colbind)

    # Data Points for N(0)=7
    N <- (-23.3 * exp(2.5*t))/((-2.33*exp(2.5*t)) - 1)
    colbind2 <- cbind(t,N)
    df2 <- rbind(df2,colbind2)

    # Data Points for N(0)=14
    N <- (35 * exp(2.5*t))/((3.5*exp(2.5*t)) - 1)
    colbind3 <- cbind(t,N)
    df3 <- rbind(df3,colbind3)


  }

# Bind all three dataframes for plotting

df <- cbind(df1,df2$N,df3$N)

colnames(df) <- c("t","N.2","N.7","N.14")

# Melt the dataset for plotting the curves

melt <- melt(df,id="t")


# Plot the curves of actual solution
p <- ggplot(data=melt,aes(x=t,y=value,color=variable)) + geom_point() + geom_line() + ylab("N")

print(p)
```



Compare actual solution with qualitative graphical solution

Actual Solution matches the qualitative graphical solution because all the solution curves go towards N=10 and reach a stable equilibrium.

Part D

Compute the time t when N is changing the fastest using the initial condition N(0) = 2. Compare your answer to your qualitative estimate in part a(iii) above.

Solution

```
# Dataframe for points when N(0) = 2

df <- data.frame()

# Loop to compute data points to find dN_dt for N(0)=2

for(t in seq(0,10,by=0.5))
  {
    # Data Points for N(0)=2

    N <- (-2.5 * exp(2.5*t))/((-0.25*exp(2.5*t)) - 1)  # Compute N for N(0)=2

    dN_dt <- (100*exp(2.5*t))/(exp(2.5*t) + 4)^2       # Compute dN/dt for N(0)=2
```

```
    colbind <- cbind(t,N,dN_dt)

    df <- rbind(df,colbind)
  }

knitr::kable(df)
```

t	N	dN_dt
0.0	2.000000	4.0000000
0.5	4.659791	6.2210644
1.0	7.528193	4.6520599
1.5	9.140175	1.9647382
2.0	9.737555	0.6388920
2.5	9.923374	0.1900983
3.0	9.977925	0.0550645
3.5	9.993666	0.0158261
4.0	9.998184	0.0045383
4.5	9.999480	0.0013006
5.0	9.999851	0.0003727
5.5	9.999957	0.0001068
6.0	9.999988	0.0000306
6.5	9.999996	0.0000088
7.0	9.999999	0.0000025
7.5	10.000000	0.0000007
8.0	10.000000	0.0000002
8.5	10.000000	0.0000001
9.0	10.000000	0.0000000
9.5	10.000000	0.0000000
10.0	10.000000	0.0000000

The time at which N changes fastest for initial condition N(0)=2 is t = 0.5.

It is comparable with Part a(iii) because in Part a(iii), we found that at N=5, we have the fastest rate of change of the disease which is dN/dt = 6.25.

And here(Part D) we see that the rate of change is the fastest dN/dt=6.22 at t=0.5.

So rate of change of disease are almost equal in both Part a(iii) and this part(PartD).

dN/dt from Part a(iii) = 6.25

dN/dt from Part D = 6.22

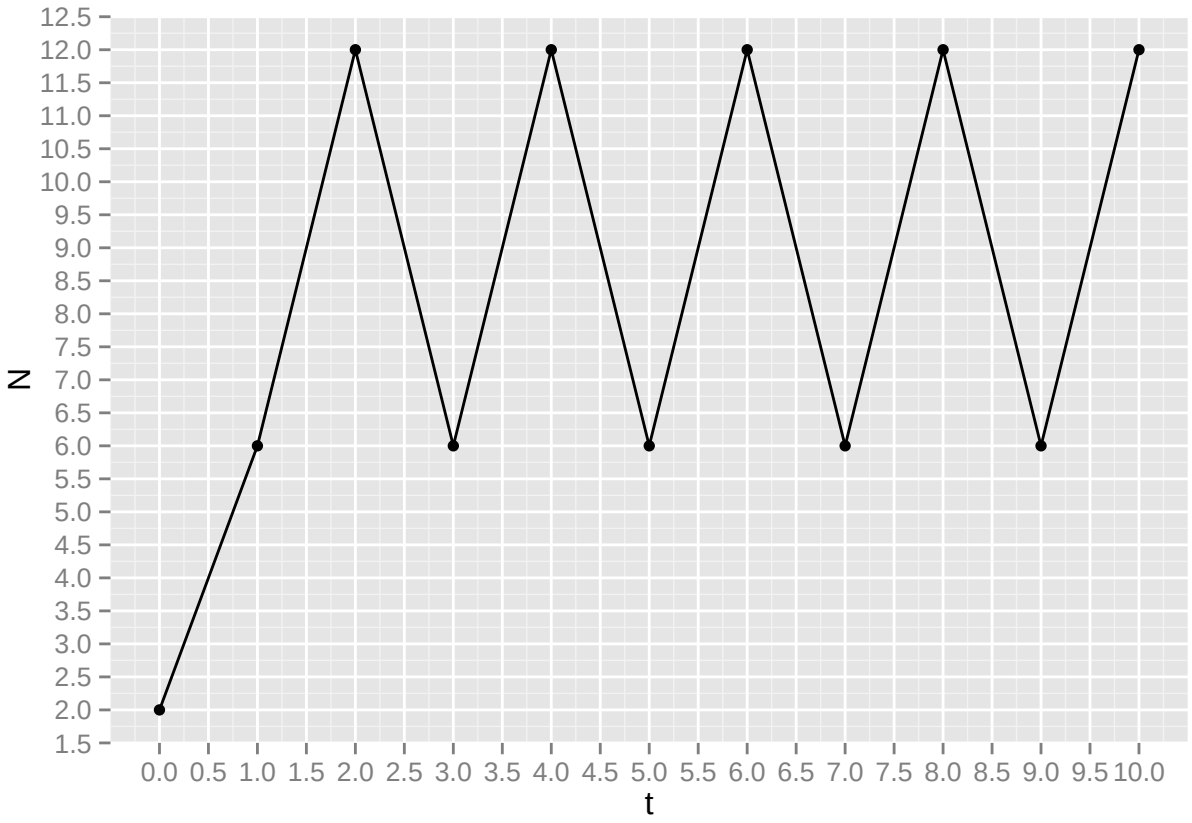
Both are almost equal.

Part E

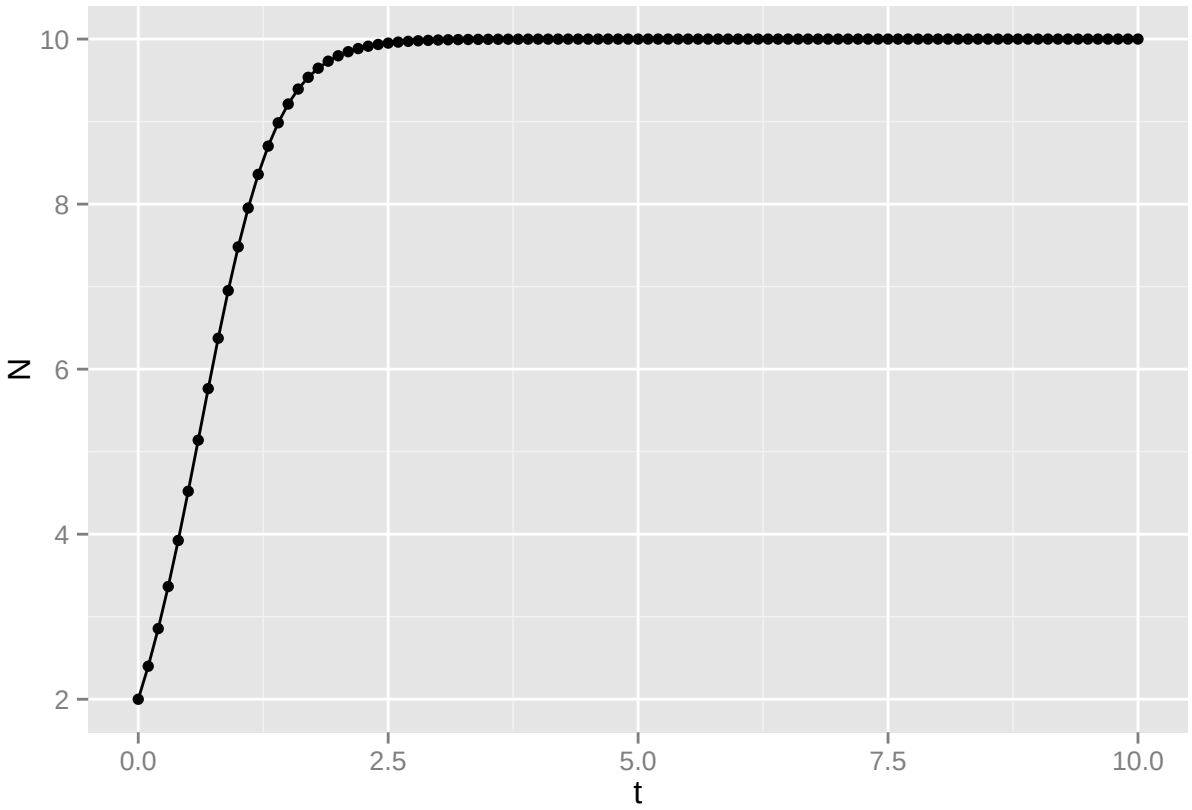
Use Euler's Method with step sizes of h=1 and then h=0.1 to approximate the solution to the differential equation for N(0.5) and N(5). Find the relative error at these values. Obtain graphical plots of the numerical solution. How do they compare to the plots of the actual solution?

Solution

```
# Function to compute numerical solution using Euler's method


eulers <- function(h)       # Accept step size

{
  N <- 0
  start <- 0                # Start of interval
  end <- 10                 # End of Interval
  N0 <- 2                   # Initial Value
  nsteps <- (end-start)/h   # Compute number of steps required
  N[1] <- N0                # Put intial value in first position of array

  t <- seq(start,end,by=h)  # Generate sequence of time(x-axis)

  for(i in 1:nsteps)        # Loop to generate data points
  {
     N[i+1] <- N[i] + ((0.25*N[i])*(10-N[i]))*h
  }

  df <- data.frame(cbind(t,N))

  return(df)
}

# Call eulers function with step sizes = 1 and plot the numerical solution

eulers.step.sz.1 <- data.frame(eulers(1))

p.eulers.step.sz1 <- ggplot(eulers.step.sz.1,aes(x=t,y=N)) + geom_point() + geom_line() + scale_x_conti

print(p.eulers.step.sz1)
```




```
# Call eulers function with step sizes = 0.1 and plot the numerical solution

eulers.step.sz.point1 <- data.frame(eulers(0.1))

p.eulers.step.sz.point1 <- ggplot(eulers.step.sz.point1,aes(x=t,y=N)) + geom_point() + geom_line()

print(p.eulers.step.sz.point1)
```



Actual Solution from PART C above is -

N = $\frac{-2.5e^{2.5t}}{-0.25e^{2.5t}-1}$ ... Eqn(4) from Part(C) above for N(0) = 2

```
############################
# Actual Solution for N(0.5)
############################

t <- 0.5
N.5 <- (-2.5 * exp(2.5*t))/((-0.25*exp(2.5*t)) - 1)
print(paste0("Actual Solution at N(0.5) ",N.5))
```

```
## [1] "Actual Solution at N(0.5) 4.65979058273264"
```

```
############################
# Actual Solution for N(5)
############################

t <- 5
N5 <- (-2.5 * exp(2.5*t))/((-0.25*exp(2.5*t)) - 1)
print(paste0("Actual Solution at N(5) ",N5))
```

```
## [1] "Actual Solution at N(5) 9.99985093609516"
```

```
dfact <- as.data.frame(rbind(N.5,N5))
colnames(dfact)[1] <- c("N")
```

```
#############################################################################################
# Relative error computation between numerical solution using  Euler's method and
# actual solution
#############################################################################################

# Get N(0.5) and N(5) for h=0.1
df.subset.point1 <- subset(eulers.step.sz.point1,t==0.5|t==5)

# From the graph of numerical solution for h=1, we get N(0.5) = 4
Npoint5 <- as.data.frame(cbind(0.5,4))
colnames(Npoint5) <- c("t","N")

# Get N(5) for h=1
subset.1 <- subset(eulers.step.sz.1,t==5)

# Combine values of N(0.5) and N(5) for h=1
df.subset.1 <- rbind(Npoint5,subset.1)

# Compute relative error between actual solution and numerical solutions
eulers.dferror <- cbind(df.subset.1,df.subset.point1$N,dfact$N)
colnames(eulers.dferror) <- c("t","N.h.1", "N.h.0.1","Actual.Solution")

eulers.dferror$err.h.1 <- eulers.dferror$Actual.Solution - eulers.dferror$N.h.1
eulers.dferror$err.h.0.1 <- eulers.dferror$Actual.Solution - eulers.dferror$N.h.0.1

# Print the relative errors for N(0.5) and N(5) with h=0.1, h=1 and actual solution

print(eulers.dferror)
```

```
##     t N.h.1  N.h.0.1 Actual.Solution   err.h.1     err.h.0.1
## 1 0.5     4 4.520413        4.659791 0.6597906  0.1393773611
## 6 5.0     6 9.999963        9.999851 3.9998509 -0.0001118685
```

Eulers Method - Error between actual solution and numerical solution with step size = 1

For N(0.5), error is 0.660 (rounded) For N(5), error is 4 (rounded)

Eulers Method - Error between actual solution and numerical solution with step size = 0.1

For N(0.5), error is 0.139 For N(5), error is -0.0001

Comparison of plot of numerical solution with actual solution

Numerical solution plot generated using Euler's method for step size = 0.1 is very similar and comparable with the plot of the actual solution and both plots reach an equibrium value of 10. The actual solution plot reaches equibrium at t=7.5 and numerical solution with step size = 0.1 reaches an equilibrium at t=6.5.

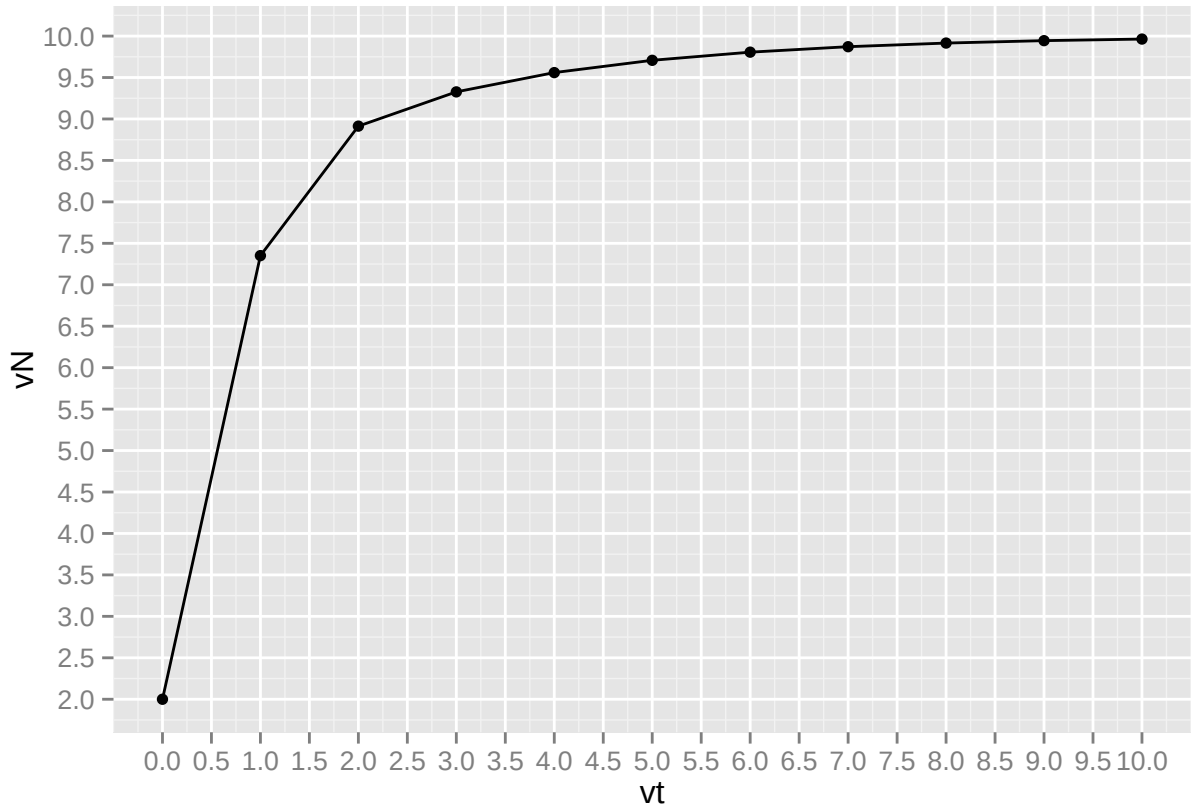
On the other hand, plot for the numerical solution with step size=1 doesn't really resemble the plot for actual solution and also it doesnt show that an equlibrium value is being reached at. It keeps swinging between N=6 and N=12 but doesnt reach equibrium at all.

Part F

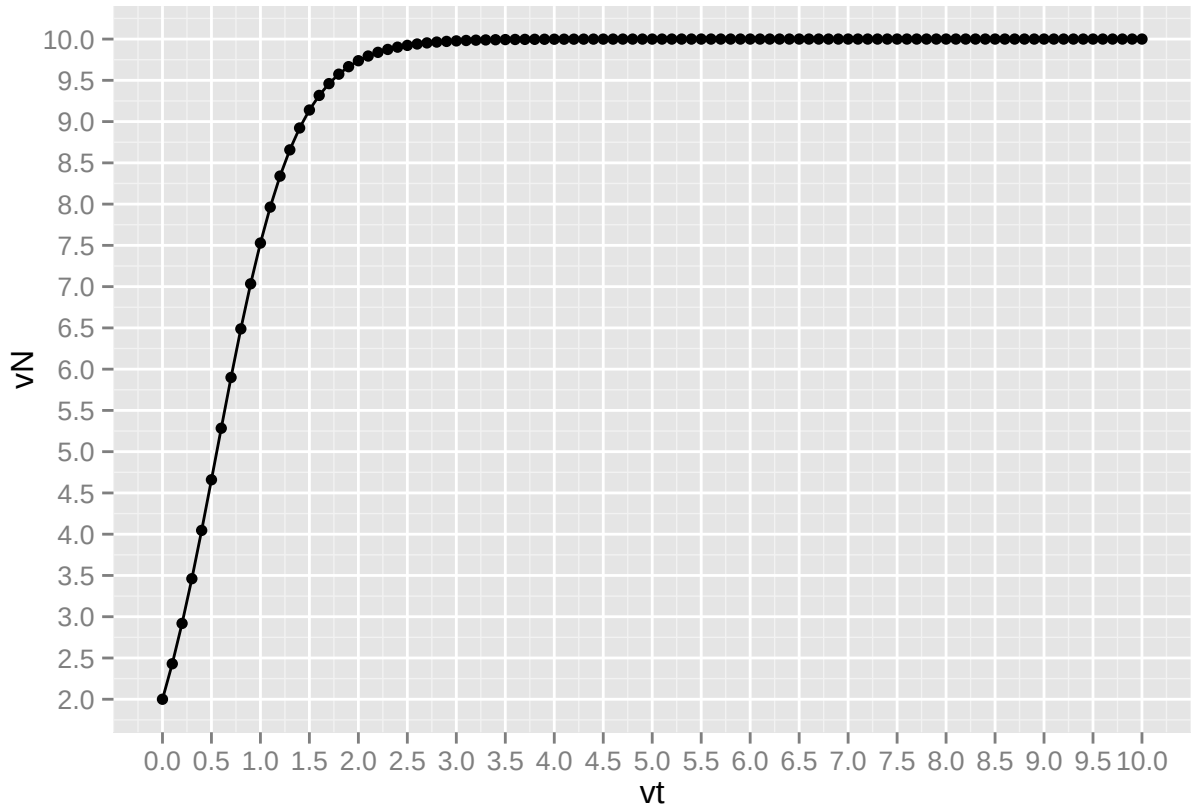
Repeat part (e) using either the Improved Euler's Method or the Runge-Kutta Method.

Solution

```
##############################################################################
# Function to compute numerical solution using 4th order Runge Kutta method
##############################################################################

rk4 <- function(f,h)

  {
    start <- 0                # Start of interval
    end <- 10                 # End of Interval
    t0 <- 0                   # Initial value of t
    N0 <- 2                   # Initial Value of N
    nsteps <- (end-start)/h   # Compute number of steps required for a given step size

    vt <- double(nsteps + 1)  # Initialize vector vt
    vN <- double(nsteps + 1)  # Initialize vector vN

    vt[1] <- t <- t0          # Put initial value of t in 1st position of array vt
    vN[1] <- N <- N0          # Put initial value of N in 1st position of array vN

    # Loop computing k1,k2,k3,k4 using Runge Kutta method

    for(i in 1:nsteps)
      {

        k1 <- f(t, N)
        k2 <- f(t + 0.5*h, N + (k1*0.5*h))
        k3 <- f(t + 0.5*h, N + (k2*0.5*h))
        k4 <- f(t+h, N + (k3*h))

        vt[i + 1] <- t <- t0 + i*h
        vN[i+1] <- N <- N + ((1/6)*(k1+(2*k2)+(2*k3)+k4)*h)

      }
        cbind(vt, vN)
}

# Call Runge Kutta function "rk4" above with step size 1

rk4.step.sz.1 <- as.data.frame(rk4(function(t, N) 0.25*N*(10-N),1))

# Call Runge Kutta function "rk4" above with step size 0.1

rk4.step.sz.point1 <- as.data.frame(rk4(function(t, N) 0.25*N*(10-N),0.1))

##############################################################################
# Plot for numerical solution using Runge Kutta method with step size 1
##############################################################################

p.rk4.step.sz.1 <- ggplot(rk4.step.sz.1,aes(x=vt,y=vN)) + geom_point() + geom_line() + scale_x_continuou

print(p.rk4.step.sz.1)
```




```
# Plot for numerical solution using Runge Kutta method with step size 0.1

p.rk4.step.sz.point1 <- ggplot(rk4.step.sz.point1,aes(x=vt,y=vN)) + geom_point() + geom_line() + scale_

print(p.rk4.step.sz.point1)
```




```
#########################################################
# Relative Error Computation using Runge-Kutta Method
#########################################################

# Get N(0.5) and N(5) for h=0.1
df.subset.point1 <- subset(rk4.step.sz.point1,vt==0.5|vt==5)

# From the graph of numerical solution for h=1, we get N(0.5) = 4.75
Npoint5 <- as.data.frame(cbind(0.5,4.75))
colnames(Npoint5) <- c("vt","vN")

# Get N(0.5) and N(5) for h=1
subset.1 <- subset(rk4.step.sz.1,vt==5)

# Combine values of N(0.5) and N(5) for h=1
df.subset.1 <- rbind(Npoint5,subset.1)

# Compute relative error between actual solution and numerical solutions

rk4.dferror <- cbind(df.subset.1,df.subset.point1$vN,dfact$N)
colnames(rk4.dferror) <- c("t","N.h.1", "N.h.0.1","Actual.Solution")

rk4.dferror$err.h.1 <- rk4.dferror$Actual.Solution - rk4.dferror$N.h.1
rk4.dferror$err.h.0.1 <- rk4.dferror$Actual.Solution - rk4.dferror$N.h.0.1

# Print the relative errors for N(0.5) and N(5) with h=0.1, h=1 and actual solution

print(rk4.dferror)
```

```
##     t    N.h.1  N.h.0.1 Actual.Solution     err.h.1    err.h.0.1
## 1 0.5 4.750000 4.659774        4.659791 -0.09020942 1.609387e-05
## 6 5.0 9.707607 9.999851        9.999851  0.29224353 5.450239e-08
```

Runge Kutta Method - Error between actual solution and numerical solution with step size = 1

For N(0.5), error is -0.09 (rounded)

For N(5), error is 0.29 (rounded)

Runge Kutta Method - Error between actual solution and numerical solution with step size = 0.1

For N(0.5), error is 0.000016 (rounded)

For N(5), error is -0.000000054(rounded)

Comparison of plot of numerical solution with actual solution

Numerical solution plot generated using Runge Kutta's method for step size = 0.1 is almost same as with the plot of the actual solution and both plots reach an equibrium value of 10. The actual solution plot reaches equibrium at t=7.5 and numerical solution with step size = 0.1 reaches an equilibrium at t=7.3.

The relative error between numerical solution and actual solution is negligible when step size=0.1 is used.

We see that Runge Kutta method is shows more accuracy compared to Eulers method as its much closer to the actual solution.

Plot for the numerical solution with step size=1 fairly resembles the plot for actual solution and almost reaches the equlibrium value. But it would need a longer time to reach equilibrium as compared to the plot with step size=0.1. The plot with step size 1 doesnt reach equibrium up until t=10, however it approaches equibrium and reaches equibrium at t=36 whereas the plot for actual solution reaches equilibrium at t=7.5 which is much faster in reaching state of equilibrium as compared to Runge Kutta method with step size 1.