IS 609 Final Project

Cheryl Bowersox, Conor Buckley, David Stern

November 23, 2015

Chapter 5.3, Project 1

The following are a series of functions to create the game of blackjack from a dealt card, up to a hand, up to a round, and up to a full two-deck game. The results of every game are put in an outcome vector: 1 for dealer win, 2 for player win, 0 for no winner. This outcome vector is then used to calculate the winnings/losses for the player. It is possible to pass a different standing amount to the player. This allows for different strategies to be examined.

First we have the general function to deal a card from a deck. It returns the card, the remaining deck and the last round flag. The last round occurs when the deck runs out and the last round flag is then set to 1, and the deck is replenished to allow the current hand to finish.

```
deal_card <- function(dc_deck,full2_deck){
  dc_last_round_flag <- 0
  if (length(dc_deck) == 0) {
    dc_last_round_flag <- 1
    dc_deck <- full2_deck
  }
  dc_card <- sample(dc_deck,1,FALSE)
  dc_deck <- dc_deck [! dc_deck %in% dc_card | duplicated(dc_deck)]
  return(list(dc_card,dc_deck,dc_last_round_flag))
}
```

Now we have a function to create a hand for the dealer. It returns the hand and the remaining deck.

```
dealer_hand <- function(dh_deck,full2_deck) {
  dh_hand <- 0
  dh_last_round_flag <- 0
  dh_count <- 1
  while (sum(dh_hand) < 17) {
    dh_deal <- deal_card(dh_deck,full2_deck)
    dh_hand[dh_count] <- dh_deal[[1]]
    dh_deck <- dh_deal[[2]]
    if (dh_deal[[3]] == 1) {dh_last_round_flag <- dh_deal[[3]]}
    dh_count <- dh_count+1
    if(sum(dh_hand) > 21) {
      if(11 %in% dh_hand) {
        dh_hand[match(11,dh_hand)] <- 1
      }
    }
  }
  return(list(dh_hand,dh_deck,dh_last_round_flag))
}
```

Here is the function to create a hand for the player. It alsoreturns the hand and the remaining deck.

```
player_hand <- function(ph_deck,full2_deck,p_threshold) {
  ph_hand <- 0
  ph_count <- 1
  ph_last_round_flag <- 0
  while (sum(ph_hand) < p_threshold) {
    ph_deal <- deal_card(ph_deck,full2_deck)
    ph_hand[ph_count] <- ph_deal[[1]]
    ph_deck <- ph_deal[[2]]
    if (ph_deal[[3]] == 1) {ph_last_round_flag <- ph_deal[[3]]}
    ph_count <- ph_count+1
    if(sum(ph_hand) > 21) {
      if(11 %in% ph_hand) {
        ph_hand[match(11,ph_hand)] <- 1
      }
    }
  }
  return(list(ph_hand,ph_deck,ph_last_round_flag))
}
```

Here is thefunction to create a round of play. It returns the hand of each player and the remaining deck.

```
round_of_play <- function(rop_deck,full2_deck,p_threshold) {
  #dealer hand
  d_result <- dealer_hand(rop_deck,full2_deck)
  d_hand <- d_result[[1]]
  rop_deck <- d_result[[2]]
  last_round_flag_d <- d_result[[3]]
  #player hand
  p_result <- player_hand(rop_deck,full2_deck,p_threshold)
  p_hand <- p_result[[1]]
  rop_deck <- p_result[[2]]
  last_round_flag_p <- p_result[[3]]

  if (last_round_flag_d == 1 || last_round_flag_p == 1) {
    rop_last_round_flag <- 1
  } else {
    rop_last_round_flag <- 0
  }
  return(list(d_hand,p_hand,rop_deck,rop_last_round_flag))
}
```

This function runs through two decks, creating a hand for the dealer and player in each round. It stores the results in a list of lists.

```
play_two_decks <- function(ptd_deck,full2_deck,p_threshold) {

  round_counter <- 1
  ptd_last_round_flag <- 0
  result_list.names <- c("Dealer_Hand", "Player_Hand")
  result_list <- vector("list", length(result_list.names))
  names(result_list) <- result_list.names

  while (ptd_last_round_flag != 1) {
```

```
    round_result <- round_of_play(ptd_deck,full2_deck,p_threshold)
    result_list$Dealer_Hand[round_counter] <- list(round_result[[1]])
    result_list$Player_Hand[round_counter] <- list(round_result[[2]])
    ptd_deck <- round_result[[3]]
    ptd_last_round_flag <- round_result[[4]]
    round_counter <- round_counter+1
  }
  return(result_list)
}
```

This function takes in a list containing all the hands for each player after running through two full decks, It returns a vector of 0's, 1's, and 2's, where 1 denotes a win for the dealer, 2 a win for the player, and 0 a round where there was no winner. If the dealer goes bust, the player automatically wins.

```
build_outcome_vector <- function(bov_one_set) {

  winner <- numeric(length(bov_one_set$Dealer_Hand))

  for (i in 1:length(bov_one_set$Dealer_Hand)) {
    d_Hand <- sum(bov_one_set$Dealer_Hand[[i]])
    p_Hand <- sum(bov_one_set$Player_Hand[[i]])
    if(d_Hand < 22) {
      if(p_Hand < 22) {
        if(d_Hand > p_Hand) {
          winner[i] <- 1
        } else if (d_Hand < p_Hand) {
            winner[i] <- 2
        } else if (d_Hand == p_Hand) {
            winner[i] <- 0
        }
      } else if(p_Hand >= 22) {
          winner[i] <- 1
      }
    } else if(d_Hand >= 22){
            winner[i] <- 2
      }
  }
  return(winner)
}
```

This function calcultes the player winnings. The betting assumptions are as follows: - The player bets 2 dollars on each hand. - If player wins, they receive 3 dollars. - If player loses, they lose their 2 dollar bet. - If it is a tie, then no money is exchanged. - If the dealer goes bust, it is a win for the player irrespective of the hand the player holds.

```
calculate_player_winnings <- function(results) {

  winnings <- numeric(length(results))

  for(i in 1:length(results)) {
    if (results[i] == 0) winnings[i] <- 0
    if (results[i] == 1) winnings[i] <- -2
    if (results[i] == 2) winnings[i] <- 3
```

```
  }
  return(winnings)
}
```

Finally we have a function to play one full game and calculate the winnings/losses. It uses the standing strategy provided as an argument to the function. It returns the total winnings/loses for the game.

```
play_two_deck_game <- function(ptdg_deck,p_threshold){
  #create a list of outcomes per round after running through a two full decks
  one_set <- play_two_decks(ptdg_deck,ptdg_deck,p_threshold)
  #build a vector to capture the result of each hand
  #dealer wins (1), player wins (2), no winner (0)
  outcome_vector <- build_outcome_vector(one_set)
  player_winnings <- 0
  #use the outcome vector to calculate the winnings based on
  #the betting strategy provided
  player_winnings <- calculate_player_winnings(outcome_vector)
  return(sum(player_winnings))
}
```

Here we wil create a set of two decks and simulate 12 two-deck games. We will run this same simulation for different strategies. The strategies are simple - stand at the specified amount, and the dealer always stands at 17 (as per the guidance in the book).

```
two_deck <- c(2,2,2,2,3,3,3,3,4,4,4,4,5,5,5,5,6,6,6,6,7,7,7,7,8,8,8,8,
              9,9,9,9,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,
              11,11,11,11,2,2,2,2,3,3,3,3,4,4,4,4,5,5,5,5,6,6,6,6,
              7,7,7,7,8,8,8,8,9,9,9,9,10,10,10,10,10,10,10,10,10,10,
              10,10,10,10,10,11,11,11,11)

n <- 12
vec15 <- numeric(n)
vec16 <- numeric(n)
vec17 <- numeric(n)
vec18 <- numeric(n)
vec19 <- numeric(n)
vec20 <- numeric(n)
for (i in 1:n){
  vec15[i] <- play_two_deck_game(two_deck,15)
  vec16[i] <- play_two_deck_game(two_deck,16)
  vec17[i] <- play_two_deck_game(two_deck,17)
  vec18[i] <- play_two_deck_game(two_deck,18)
  vec19[i] <- play_two_deck_game(two_deck,19)
  vec20[i] <- play_two_deck_game(two_deck,20)
}
```

Put the winnings/loses (in dollars) for each game per strategy:

```
final_output <- data.frame("Stand_At_15"=vec15,"Stand_At_16"=vec16,
                           "Stand_At_17"=vec17,"Stand_At_18"=vec18,
                           "Stand_At_19"=vec19,"Stand_At_20"=vec20)
final_output
```

```
##    Stand_At_15 Stand_At_16 Stand_At_17 Stand_At_18 Stand_At_19 Stand_At_20
## 1           15          -6          24           8          28           3
## 2            7         -10          17          12          16           1
## 3            6          16           7          28          33          -3
## 4            0           9          19          28          38          23
## 5           34          22           4          23          19          19
## 6           17          15           9          16          14          -6
## 7           14          11          18          13         -14          19
## 8            8          -6          21           0          11          21
## 9            9           1          21          -4          -4          26
## 10          17          38          13          16           6          20
## 11          14          29          -1           5          21          15
## 12          11          16          11          24           9          11
```

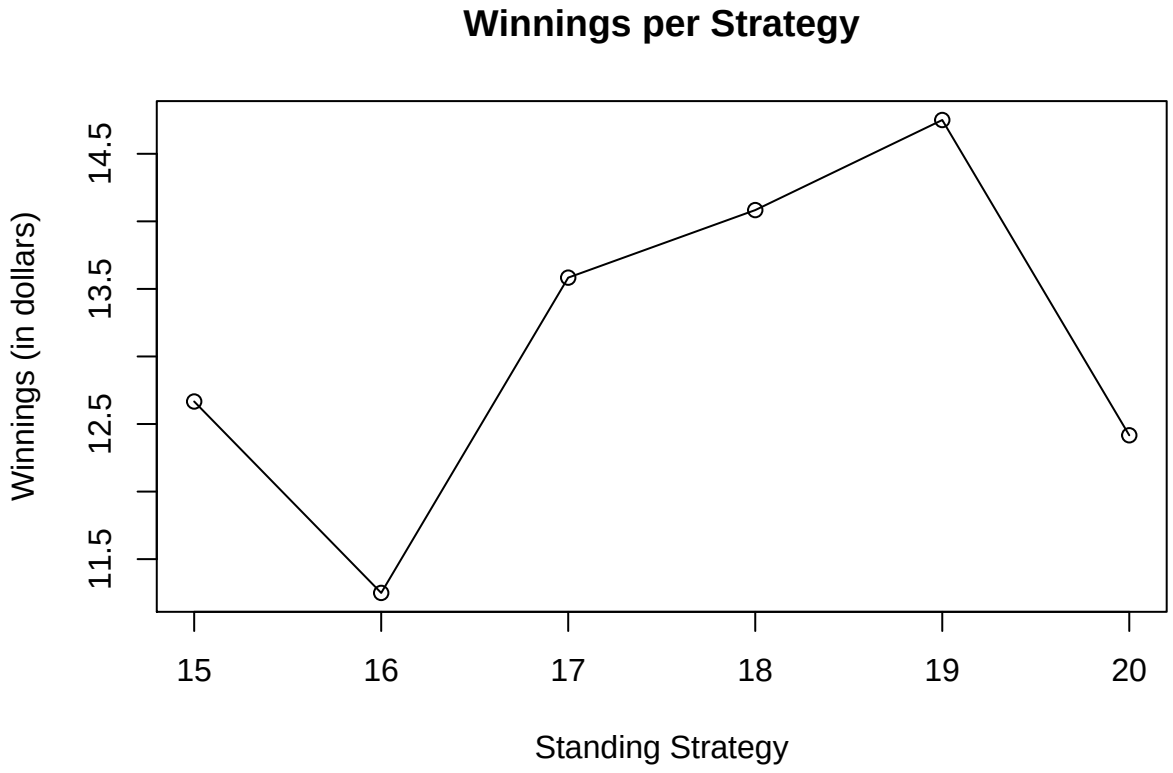
Put the average winnings/loses in a vector:

```
final_output_avg <- c("Stand_At_15"=mean(vec15),"Stand_At_16"=mean(vec16),
                      "Stand_At_17"=mean(vec17),"Stand_At_18"=mean(vec18),
                      "Stand_At_19"=mean(vec19),"Stand_At_20"=mean(vec20))
final_output_avg
```

```
## Stand_At_15 Stand_At_16 Stand_At_17 Stand_At_18 Stand_At_19 Stand_At_20
##    12.66667    11.25000    13.58333    14.08333    14.75000    12.41667
```

Create a plot of the average winnings by strategy:

```
plot(final_output_avg,type="o",xaxt="n",main="Winnings per Strategy",
     xlab="Standing Strategy",ylab="Winnings (in dollars)")
axis(1,at=1:6,labels=c("15","16","17","18","19","20"))
```



Chapter 7.5, Project 2

A farmer has 30 acres on which to grow tomatoes and corn. Each 100 bushels of tomatoes require 1000 gallons of water and 5 acres of land. Each 10 bushels of corn require 6000 gallons of water and 2.5 acres of land. Labor costs are \$1 per bushel for both corn and tomatoes. The farm has available 30,000 gallons of water and \$750 in capital. He knows he cannot sell more than 500 bushels of tomatoes and 475 bushels of corn. He estimates a profit of \$2 on each bushel of tomatoes and \$3 on each bushel of corn.

(a) How many bushels should he raise to maximize profits?

n_t = number of bushels of tomatoes (in hundreds)
n_c = number of bushels of corn (in hundreds)
w = number of gallons of water (in thousands)
p = profit (in dollars)
a = land (in acres)

Constraints: $30,000$ gallons of water available, $w \leq 30$

Each unit of n_t needs 1000 gallons water, and each unit n_c needs 6000 gallons.

$$n_t + 6n_c \leq w \leq 30$$

$$n_t \leq 30 - 6n_c$$

Each unit of n_t needs 5 acres, each unit of n_c needs 2.5 acres. Total 30 acres available:

$$5n_t + 2.5n_c \leq 30$$

$$n_t \leq 12 - 2n_c$$

There is \$750 starting capital available. Labor is \$100 per 100 bushels:

$$100(n_t + n_c) \leq 750$$

$$n_t \leq 7.5 - n_c$$

Maximum number of units that can be sold: n_t is 5, n_c is 4.50

$$n_t \leq 5$$

$$n_c \leq 4.5$$

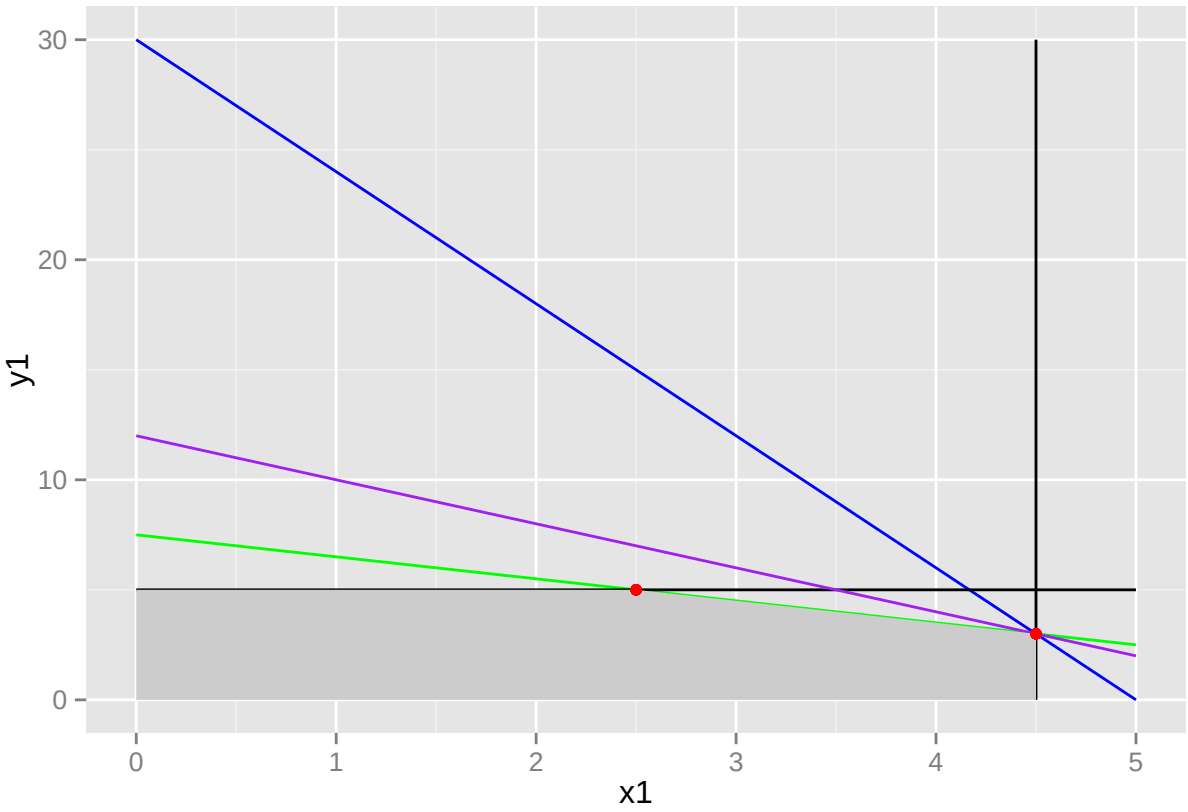
Profit for each unit n_t is \$200, n_c is \$300

$$P = 200n_t + 300n_c$$

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

f1 <- function(x) 30-6*x
f2 <- function(x) 7.5 - x
f3 <- function(x) 5
f4 <- function(x) 12-2*x
```

```
x1 = seq(0,5, by = .5)
x0 = rep(4.5,length(x1))
df = data.frame(x1,x0, y1=f1(x1), y2=f2(x1),y3= f3(x1), y4 = f4(x1))
df <-  transform(df, z =pmin(y1,y2,5,y4), z2 = pmin(x1,4.5))
ggplot(df, aes(x = x1)) +
  geom_line(aes(y = y1), colour = 'blue') +
  geom_line(aes(y = y2), colour = 'green') +
  geom_line(aes(y = y3), colour = 'black') +
  geom_line(aes(y = y4), colour = 'purple') +
  geom_line(aes(y = y1, x = x0), colour = 'black') +
  geom_ribbon(aes(ymin=0, ymax = z, x= z2), fill = 'gray80') +
  geom_point(aes(x = 4.5, y=f1(4.5)), color="red")+
  geom_point(aes(x = 2.5, y=f2(2.5)), color="red")
```



The shaded region of the graph above represent possible values for both corn and tomatoes that fit the given constraints. Evaluating the extreme points looking for highest profit:

$$n_t = 0$$
$$n_c = 5$$
$$P = 200(0) + 300(5) = 1500$$

$$n_t = 2.5$$
$$n_c = 5$$
$$P = 200(2.5) + 300(5) = 2000$$

$$n_t = 4.5$$

$$n_c = 3$$

$$P = 200(4.5) + 300(3) = 1800$$

The graph shows the when $0 \leq n_t \leq 2.5$ then n_c is bound by the maximum of 5.
Moving along this the line where $n_c = 5$, the profit increases by 200 as n_t increase 0 to 2.5, with \$2000 being the highest amount in this range.

When $2.5 < n_t \leq 4.5$ then n_c is bound by the line representing the capital constraint. Because the slope of this line is negative one, for each additional n_t there is a corresponding decrease in the amount n_c. The total profit will therefore decrease by the difference between the per unit profit for n_t and n_c, \$100.

Since the profit decreases for values less than $n_t = 2.5$, and decreases for values greater than $n_t = 2.5$, the point where $(2.5, 3)$ represents the maximum profit.

The farmer should raise 250 bushels of tomatoes and 300 bushels of corn for a maximum profit of \$2,000.

(b) Assume the farmer has the oppertunity to sign a contract with grocery store to grow and deliver at least 300 bushels of tomatoes and 500 bushels of corn. Should he sign the contract?

The two constraints that change from part (a) are:

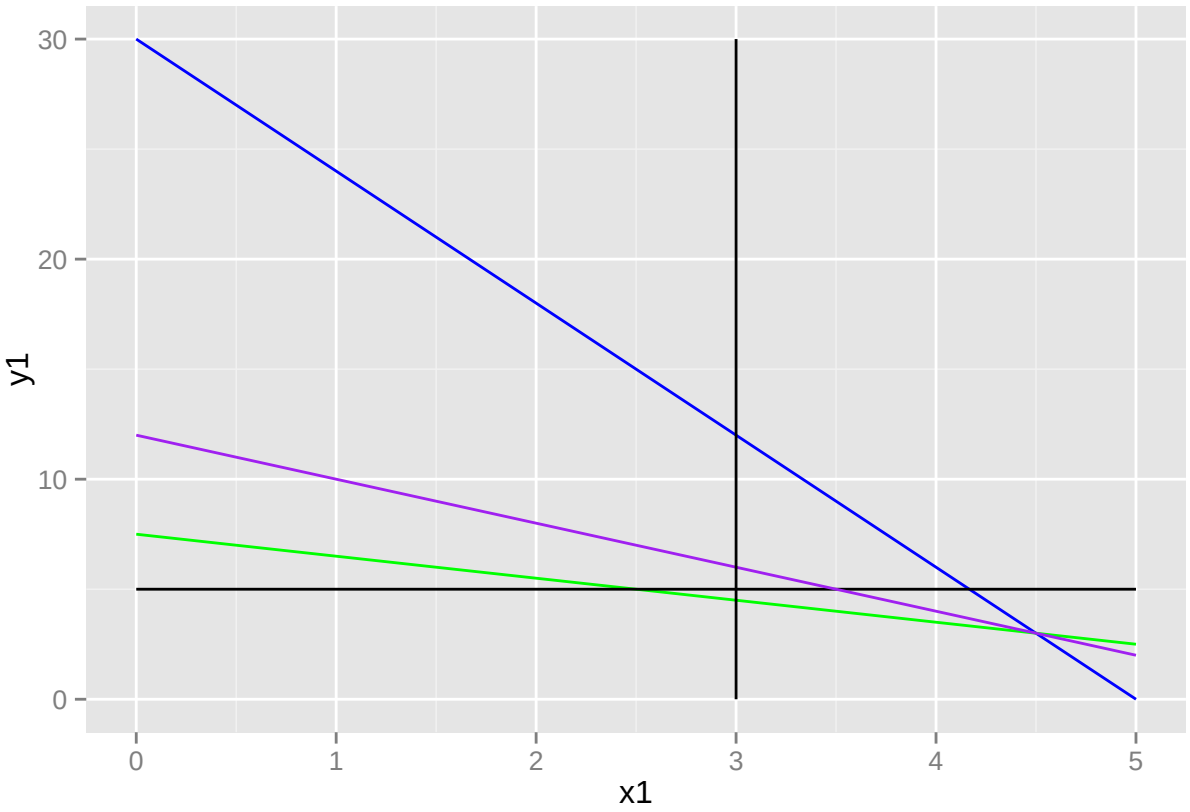
Minimum number of units that can be sold: n_t is 3, n_c is 5

$$n_t \geq 3$$

$$n_c \geq 5$$

Graphing this new system gives:

```
f1 <- function(x) 30-6*x
f2 <- function(x) 7.5 - x
f3 <- function(x) 5
f4 <- function(x) 12-2*x
x1 = seq(0,5, by = .5)
x0 = rep(3,length(x1))
df = data.frame(x1,x0, y1=f1(x1), y2=f2(x1),y3= f3(x1), y4 = f4(x1))
df <-  transform(df, z =pmax(pmin(y1,y2,y4),5), z2 = pmax(x1,3))
ggplot(df, aes(x = x1)) +
  geom_line(aes(y = y1), colour = 'blue') +
  geom_line(aes(y = y2), colour = 'green') +
  geom_line(aes(y = y3), colour = 'black') +
  geom_line(aes(y = y4), colour = 'purple') +
  geom_line(aes(y = y1, x = x0), colour = 'black') +
  geom_ribbon(aes(ymin=5, ymax = z, x= z2), fill = 'gray80')
```



There is no corresponding point that meets these new conditions because of the capital constraint (green line) where $n_t \leq 7.5 - n_c$ is always below the minimum quantity required by the new contract. There is not enough starting capital to pay for the labor needed to produce the minimum quantities.

The farmer would not be able to fulfill the contract and should not sign.

(c) Assume the farmer can obtain an additional 10,000 gallons of water for \$50 per gallon. Should he obtain the additional water?

He should not obtain the additional water. The line representing the water (blue) does not constrain the available values. They are still constrained by the available capital, meaning additional water will not result in an increase for n_t or n_c because there is not enough capital available to support the needed labor.

Additionally, because additional water will cost additional capital, obtaining more water will in fact reduce the maximum profit from the original scenario.

The new scenario changes these constraints:

40,000 gallons of water available, $w \leq 40$

Each unit of n_t needs 1k gallons water, and each unit n_c needs 6k gallons

$$n_t + 6n_c \leq w \leq 40$$

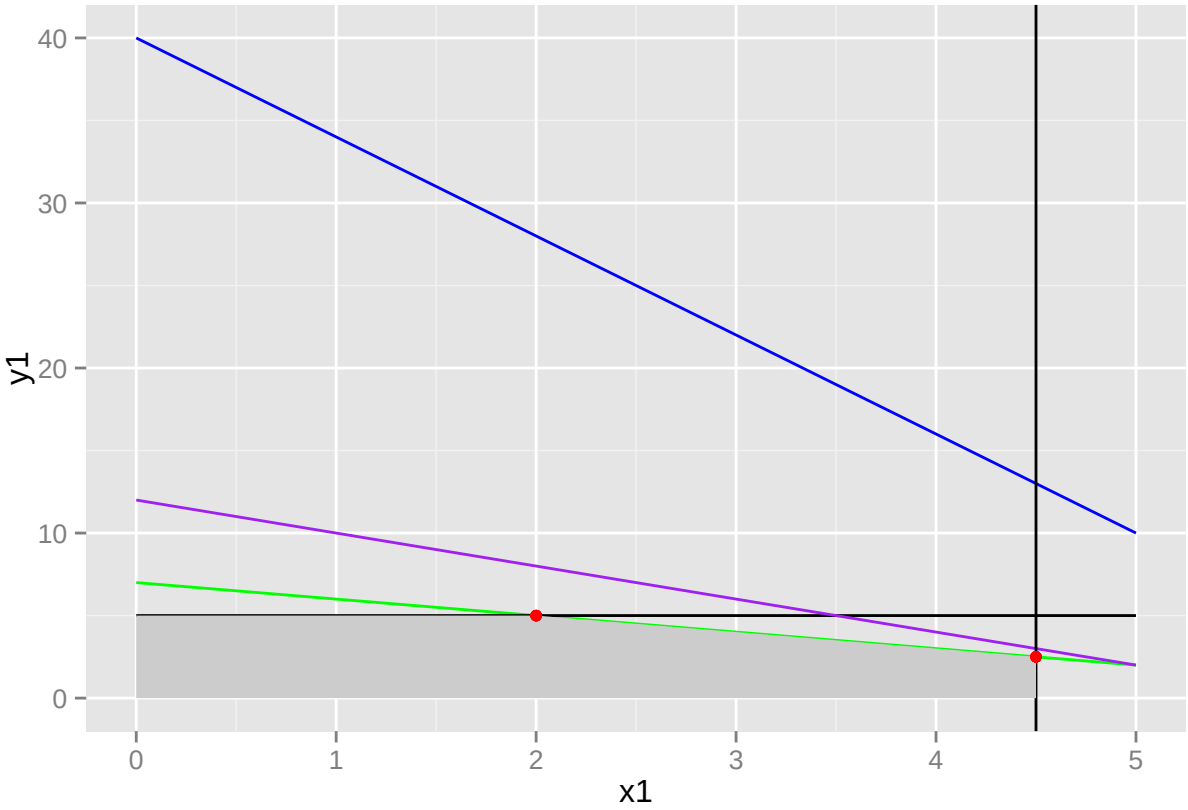
$$n_t \leq 40 - 6n_c$$

\$700 starting capital available. Labor is \$100 per 100 bushels (\$50 spend on water)

$$100(n_t + n_c) \leq 700$$

$$n_t \leq 7 - n_c$$

```
f1 <- function(x) 40-6*x
f2 <- function(x) 7 - x
f3 <- function(x) 5
f4 <- function(x) 12-2*x
x1 = seq(0,5, by = .5)
x0 = rep(4.5,length(x1))
df = data.frame(x1,x0, y1=f1(x1), y2=f2(x1),y3= f3(x1), y4 = f4(x1))
df <-  transform(df, z =pmin(y1,y2,5,y4), z2 = pmin(x1,4.5))
ggplot(df, aes(x = x1)) +
  geom_line(aes(y = y1), colour = 'blue') +
  geom_line(aes(y = y2), colour = 'green') +
  geom_line(aes(y = y3), colour = 'black') +
  geom_line(aes(y = y4), colour = 'purple') +
   geom_vline(xintercept = 4.5) +
  geom_ribbon(aes(ymin=0, ymax = z, x= z2), fill = 'gray80') +
  geom_point(aes(x = 4.5, y=f2(4.5)), color="red")+
  geom_point(aes(x = 2, y=f2(2)), color="red")
```



Reducing the available starting capital moves the point of maximum profit to $(2, 5)$ or \$1,900. Nothing is gained by adding additional water unless the available capital increases as well.

Chapter 12.5, Project 1

Using the improved Euler's method, approximate the solution to the predator–prey problem in Example 2. Compare the new solution to that obtained by Euler's method using $\Delta t = 0.1$ over the interval $0 \leq t \leq 3$. Graph the solution trajectories for both solutions.

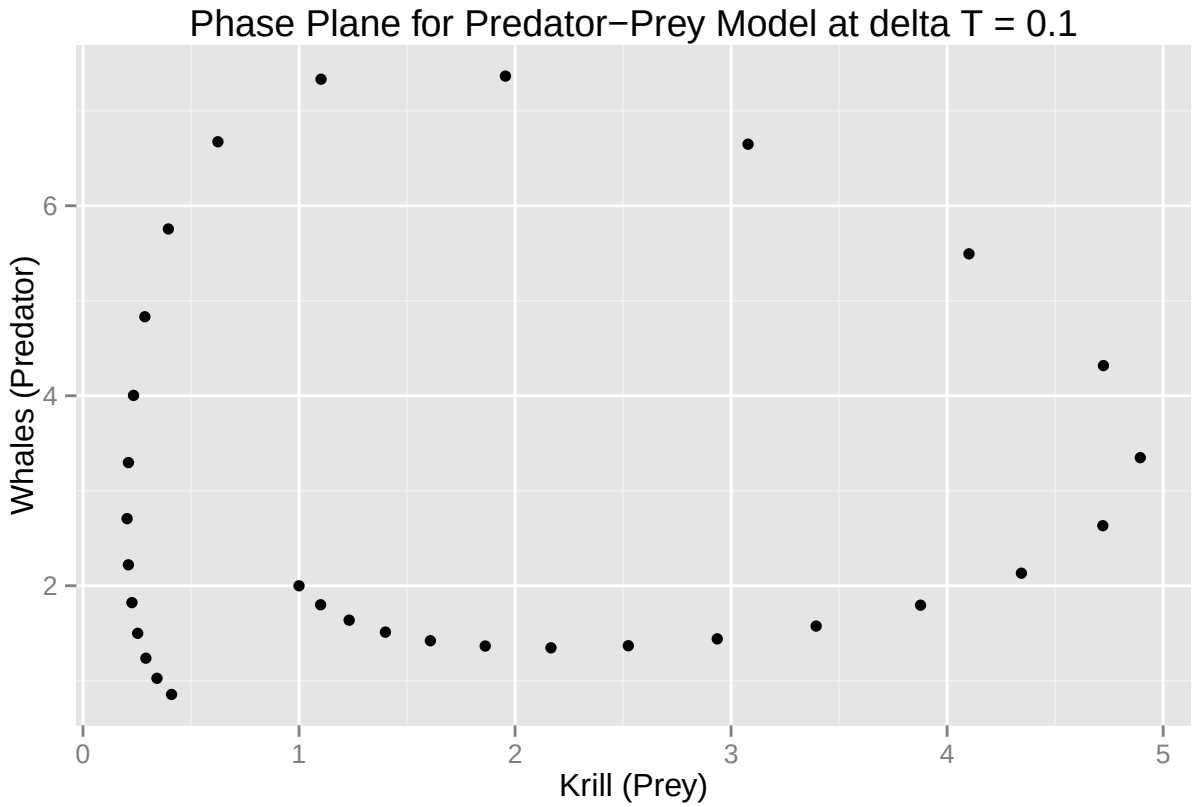
$$\frac{dx}{dt} = 3x - xy$$
$$\frac{dy}{dt} = xy - 2y$$
$$x_0 = 1, y_0 = 2$$

```
x <- c(1)
y <- c(2)
deltaT <- 0.1
t <- seq(0,3,by=deltaT)

for (i in 1:30){
 x[i+1] <- x[i] + (3*x[i] - x[i]*y[i])*deltaT
 y[i+1] <- y[i] + (x[i]*y[i] - 2*y[i])*deltaT
}

df <- data.frame(t,x,y)

ggplot(df) +
  geom_point(aes(x=x,y=y)) +
  ggtitle("Phase Plane for Predator-Prey Model at delta T = 0.1") +
  xlab("Krill (Prey)") +
  ylab("Whales (Predator)")
```



In this plot, we see an approximation to the solution trajectory for the predator-prey model. Starting at $(1, 2)$ and moving counter-clockwise, we see that the points do not cycle back through the initial point, demonstrating that it diverges from the true solution trajectory, which is periodic. We can get a better

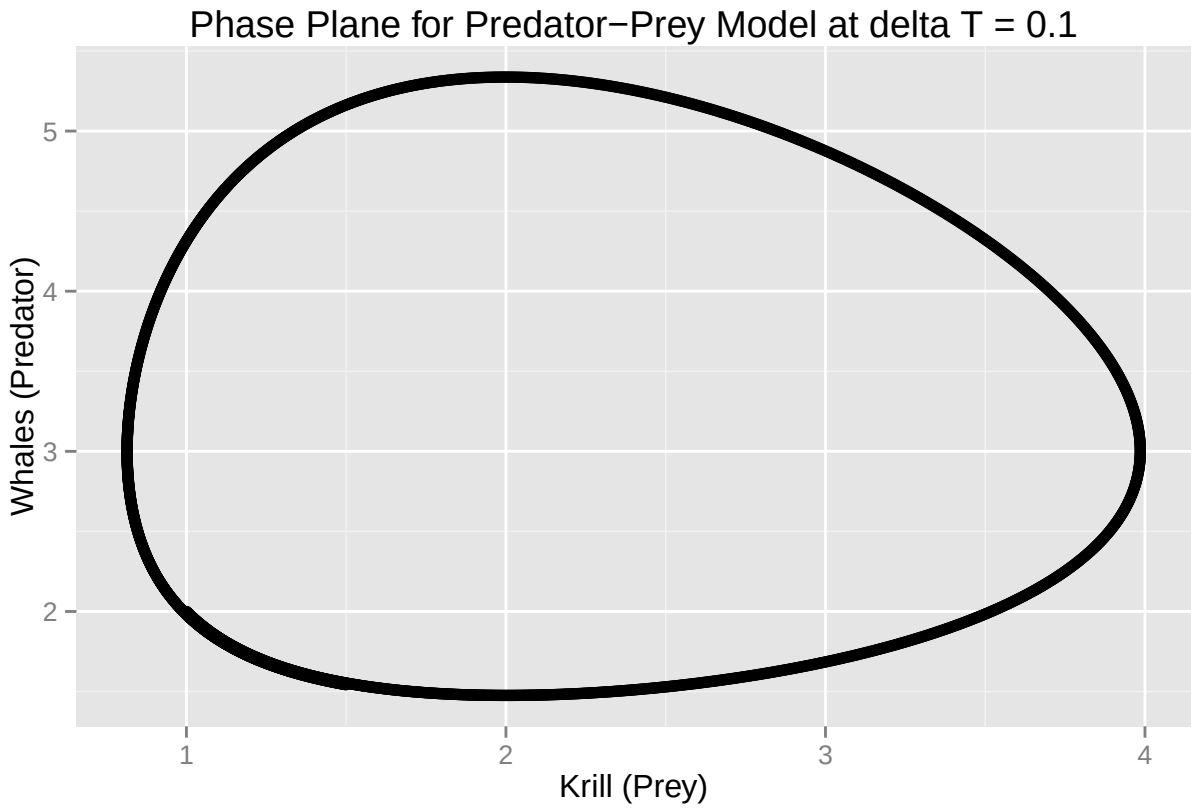
approximation to true periodicity if we reduce the value of Δt. In this next plot, we will see what happens when we $\Delta t = 0.001$, so $n = 1000$.

```
x2 <- c(1)
y2 <- c(2)
deltaT <- 0.001
t <- seq(0,3,by=deltaT)

for (i in 1:3000){
 x2[i+1] <- x2[i] + (3*x2[i] - x2[i]*y2[i])*deltaT
 y2[i+1] <- y2[i] + (x2[i]*y2[i] - 2*y2[i])*deltaT
}

df <- data.frame(t,x2,y2)

ggplot(df) +
  geom_point(aes(x=x2,y=y2)) +
  ggtitle("Phase Plane for Predator-Prey Model at delta T = 0.1") +
  xlab("Krill (Prey)") +
  ylab("Whales (Predator)")
```



Instead of reducing the value of Δt, we will see if we can improve the accuracy of the approximation using the improved Euler's method, in which the solution trajectory is found by taking the average of the two slopes:

$$x_{n+1} = x_n + \left[f(t_n, x_n, y_n) + f(t_{n+1}, x^*_{n+1}, y^*_{n+1})\right] \frac{\Delta t}{2}$$

$$y_{n+1} = y_n + \left[g(t_n, x_n, y_n) + g(t_{n+1}, x^*_{n+1}, y^*_{n+1})\right] \frac{\Delta t}{2}$$

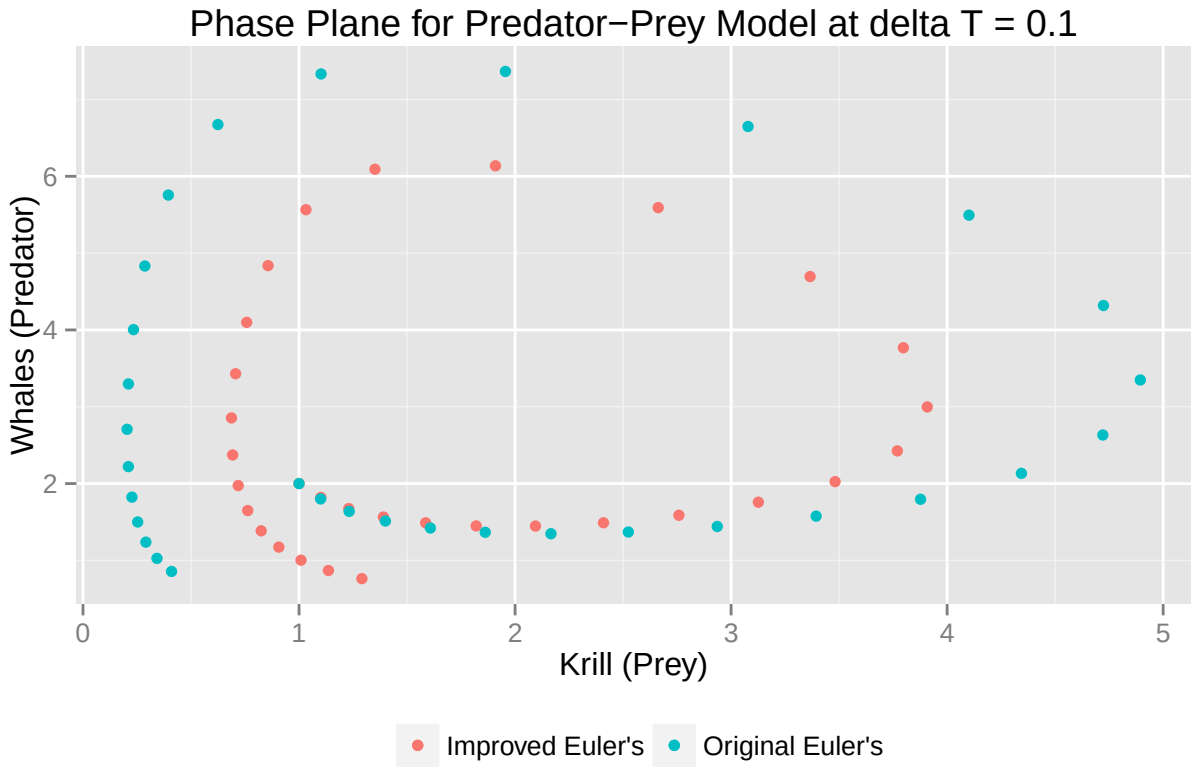
The values of x^*_{n+1} and y^*_{n+1} will be the estimates from the original Euler method above.

```
xstar <- x
ystar <- y
x <- c(1) # re-initialize vectors
y <- c(2)
deltaT <- 0.1
t <- seq(0,3,by=deltaT)

for (i in 1:30){
 x[i+1] <- x[i] + (3*x[i] - x[i]*y[i] + 3*xstar[i] - xstar[i+1]*ystar[i+1])*deltaT/2
 y[i+1] <- y[i] + (x[i]*y[i] - 2*y[i] + xstar[i+1]*ystar[i+1] - 2*ystar[i+1])*deltaT/2
}

df <- data.frame(t,x,y,xstar,ystar)

ggplot(df) +
  geom_point(aes(x=x,y=y,color="Improved Euler's")) +
  geom_point(aes(x=xstar,y=ystar,color="Original Euler's")) +
  ggtitle("Phase Plane for Predator-Prey Model at delta T = 0.1") +
  xlab("Krill (Prey)") +
  ylab("Whales (Predator)") +
  theme(legend.title = element_blank(),legend.position="bottom")
```



In this plot, we can see how this method improves the accuracy of the approximation without having to reduce the value of Δt. The points in red show us that the solution trajectory for improved Euler's is "tighter" than original Euler's. Although the approximations do not cycle around exactly to the point $(1, 2)$, the trajectory is demonstrably more periodic than the original method.