Assignment2

Michael Seebregts

2025-10-27

```
## Warning: package 'magick' was built under R version 4.5.1

## Linking to ImageMagick 6.9.13.29
## Enabled features: cairo, freetype, fftw, ghostscript, heic, lcms, pango, raw, rsvg, webp
## Disabled features: fontconfig, x11

## Warning: package 'GA' was built under R version 4.5.1

## Loading required package: foreach

## Loading required package: iterators

## Package 'GA' version 3.2.4
## Type 'citation("GA")' for citing this R package in publications.

##
## Attaching package: 'GA'

## The following object is masked from 'package:utils':
##
##     de
```

#Q1

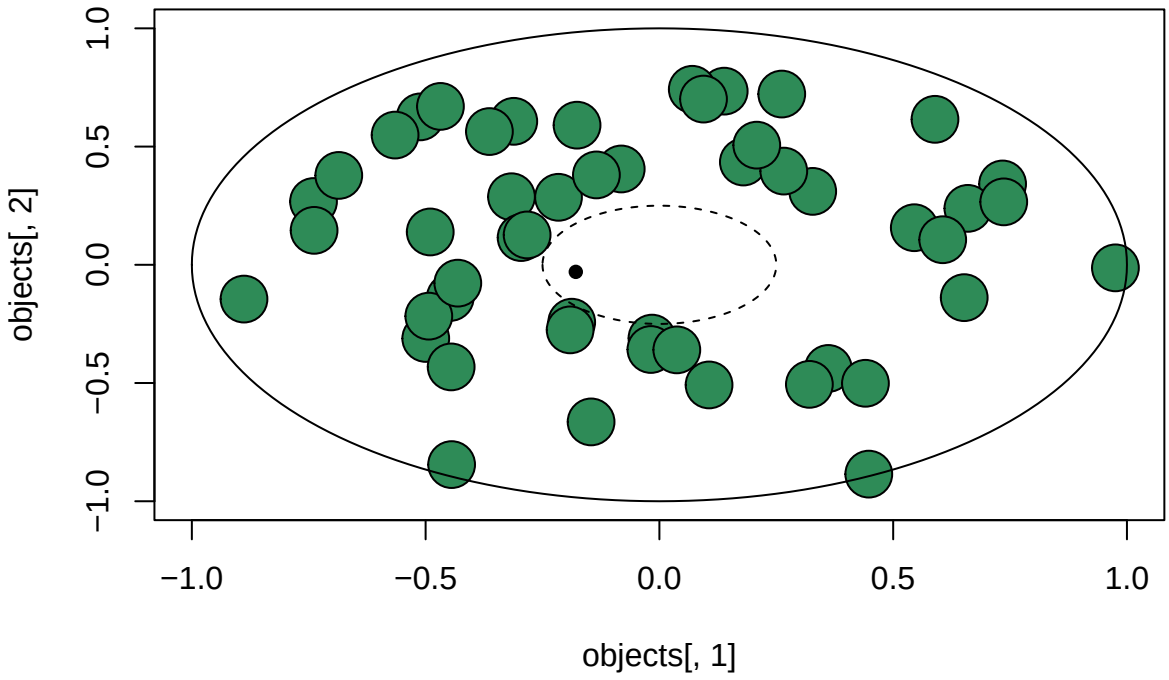
Q1a

```
game_status=function(xt,objects, rad)
{
  sk    = rep(0,dim(xt)[1])
  min_dists = rep(0,dim(xt)[1])
  J         = dim(objects)[1]
  ones      = matrix(1,J,1)
  for(i in 1:dim(xt)[1])
  {
    min_dists[i] = min(sqrt(rowSums((objects-ones%*%xt[i,])^2)))

    if (((xt[i, 1])^2 + (xt[i, 2]^2 ) >= 1))
    {
      sk[i] = 1
    }
    else if (min_dists[i] < rad)
    {
      sk[i] = -1
    }
    else
    {
      sk[i] = 0
    }
  }

  #sk = 1*(xt[,2]>1)-1*((xt[,1]< -1)|(xt[,1]> +1)|(xt[,2]< -1)|(min_dists<rad))
  ret = list(status = sk, minDist = min_dists)
  return(ret)
}
```

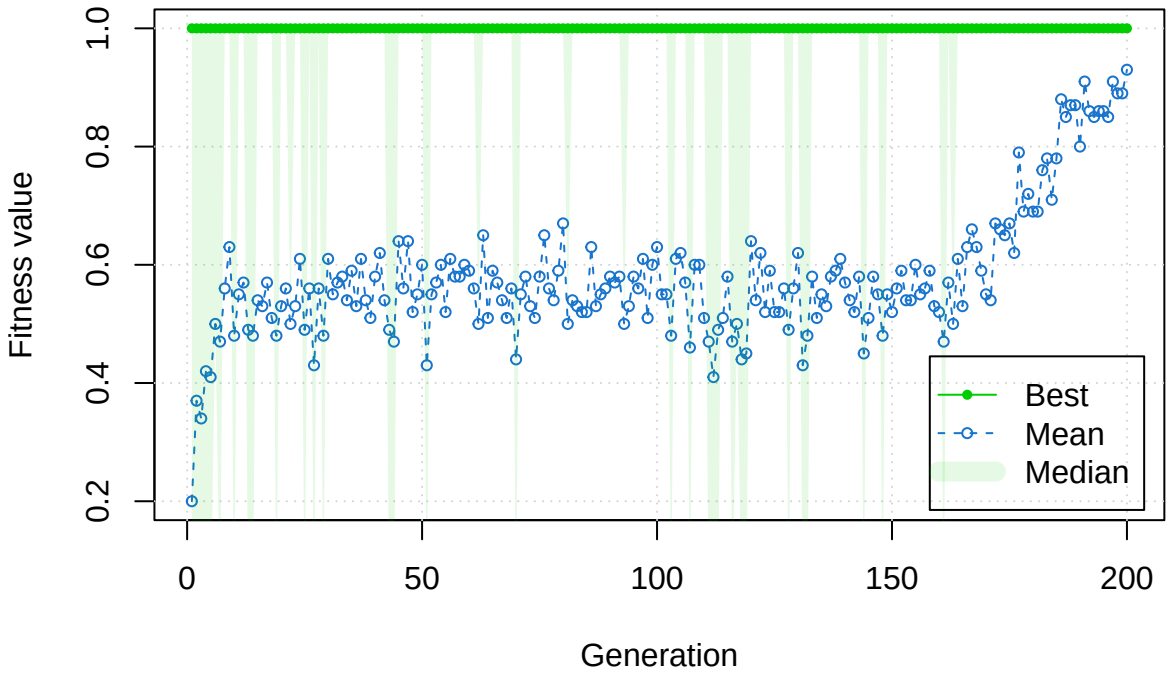
Q1b

i

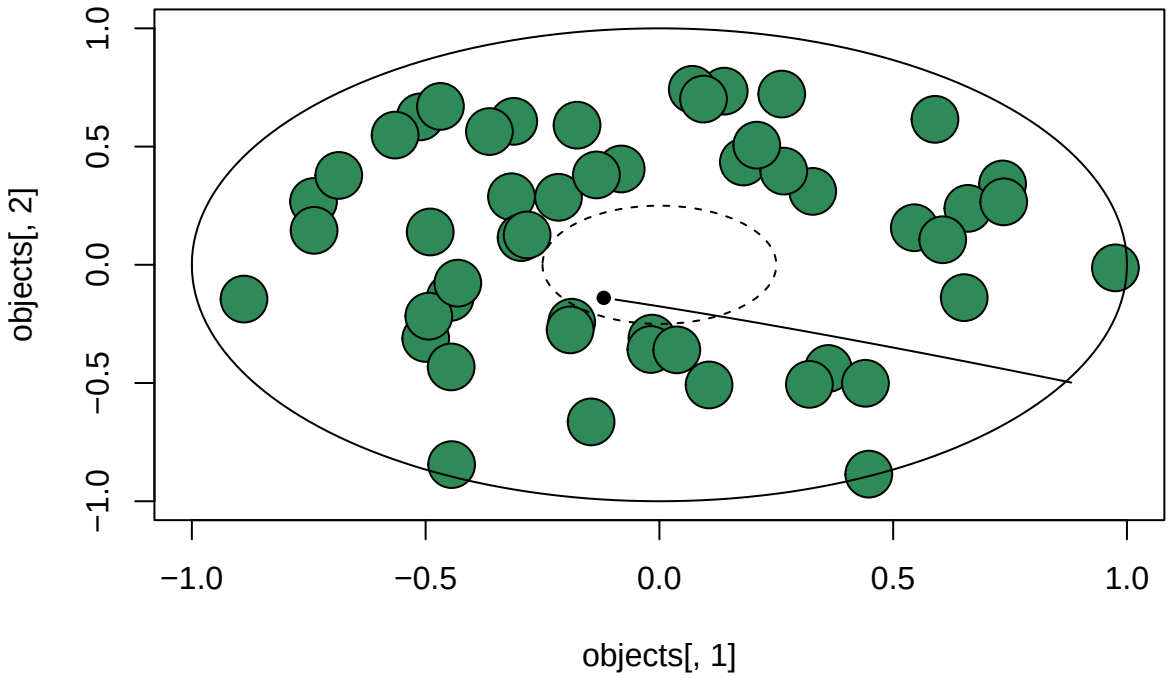
```
J       =  50
objects = draw_objects(J)
xt      = draw_starts()
delt    = 0.025             # How big are the steps you can take?
rad     = 0.05              # How close to an object before you crash?

plot_game(xt,objects,rad,'black')
```

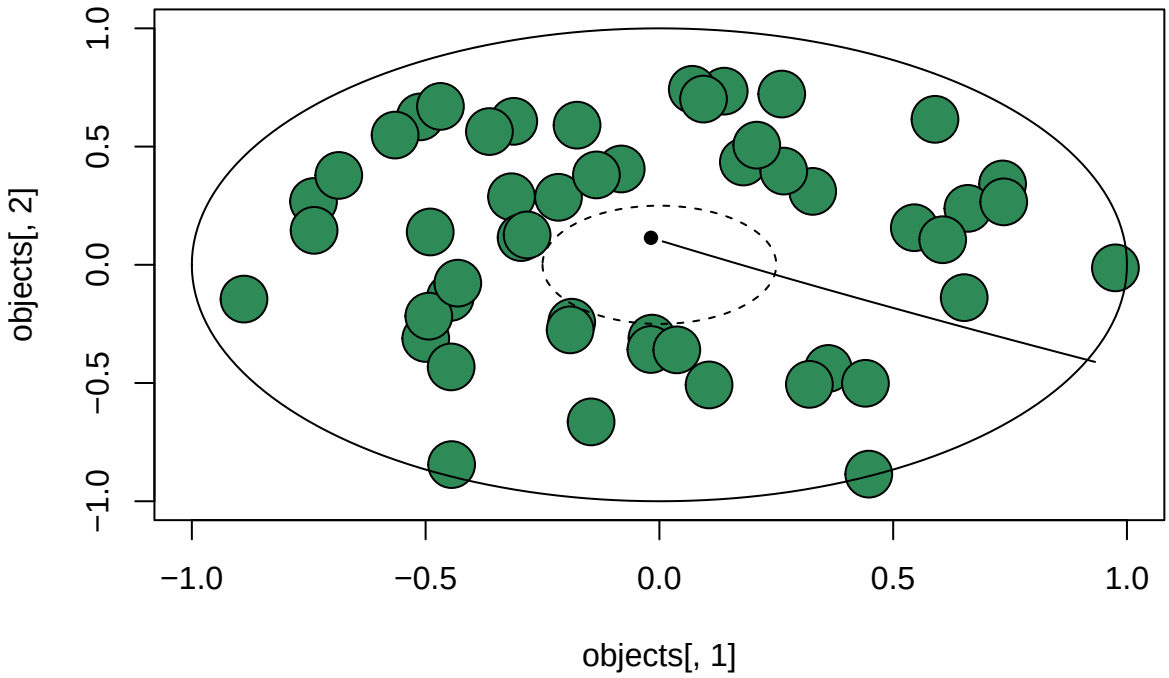


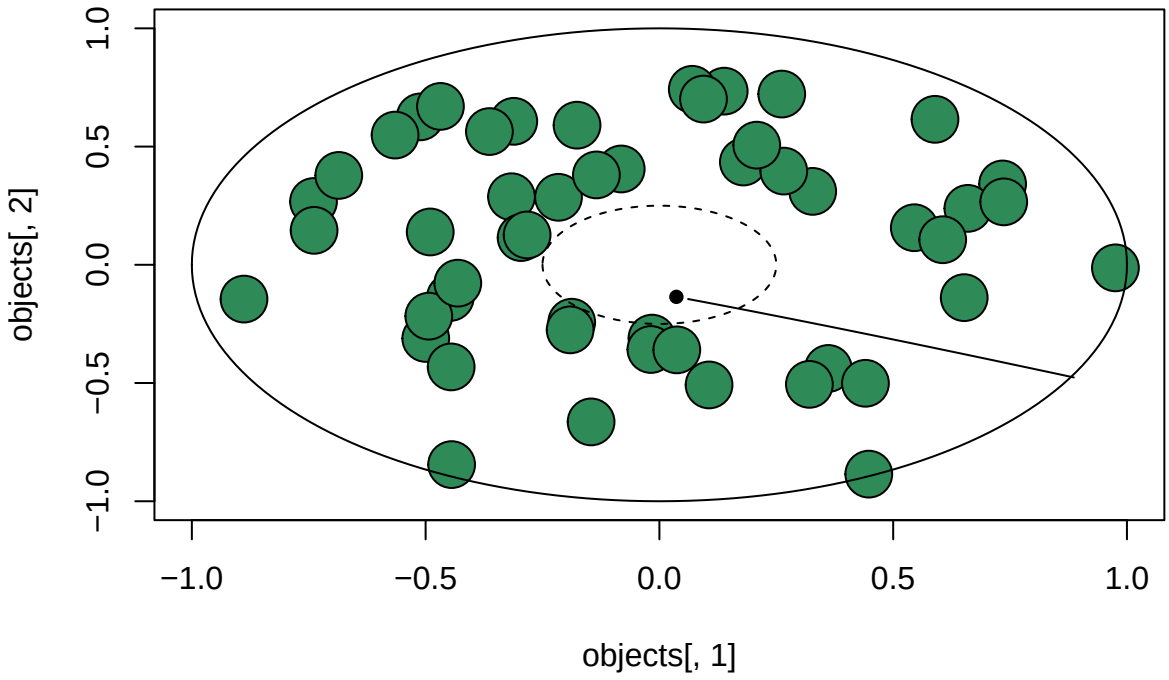

```
play_a_game = function(theta)
{
  xt    = draw_starts()
  res   = play(xt,delt,objects,rad,theta, trace = TRUE)
  score = mean(res$status==1)
  return(score)
}
# play_a_game(theta_rand)

p     = 2
q     = 2
nodes = 3
npars = p*nodes+nodes*nodes+nodes*q+nodes+nodes+q
theta_rand = runif(npars,-1,1)

obj = play_a_game
GA  = ga(type = 'real-valued',fitness = play_a_game,lower = rep(-10,npars),upper = rep(10,npars), popSi
plot(GA)
```




```
theta_hat = GA@solution[1,]
#theta_hat

win = c()
for(i in 1:100)
{

  #objects = draw_objects(J)
  xt_try    = draw_starts()
  res_final = play(xt_try,delt,objects,rad,theta_hat,plt = FALSE,trace = TRUE)
  #print(typeof(res_final$trajectories))
  trajs = na.omit(res_final$trajectories)

  if (i == 1 || i == 50 || i == 100)
  {
      plot_game(xt_try,objects,rad,'black')
      lines(trajs[, 2,]~trajs[, 1, ])
  }

  win[i] = (res_final$status==1)
}
```


objects[, 2]
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
objects[, 1]


objects[, 2]
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
objects[, 1]



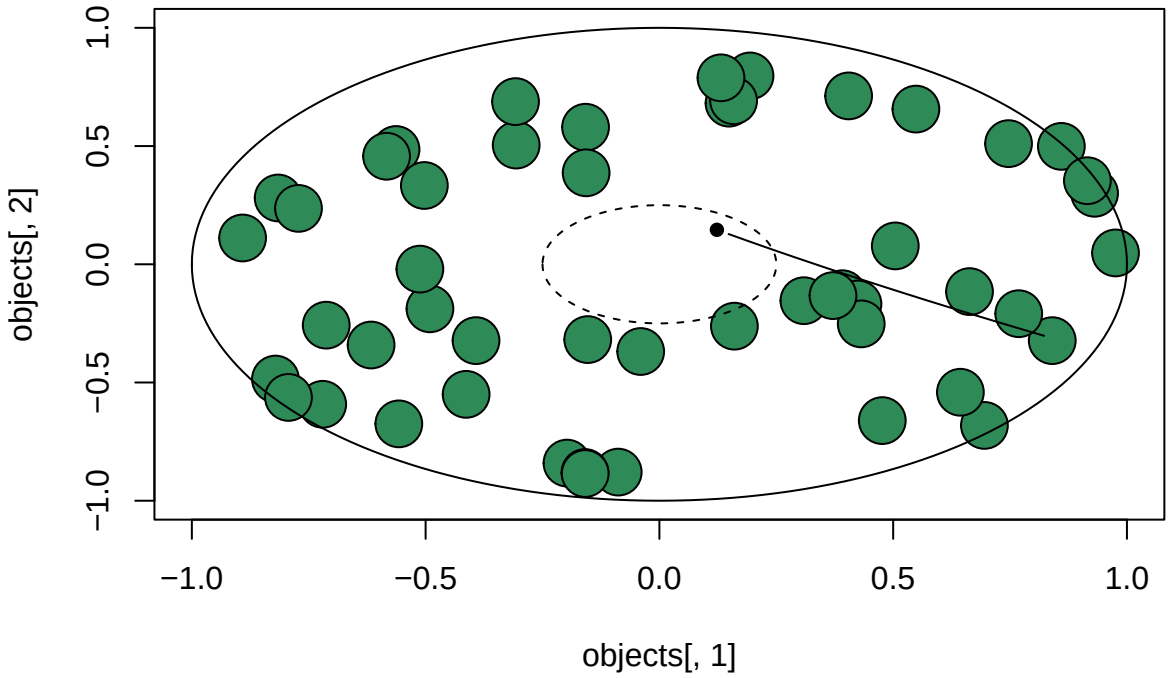

```
propWins = data.frame("Proportion of successful navigations" = mean(win))
kable(propWins)
```

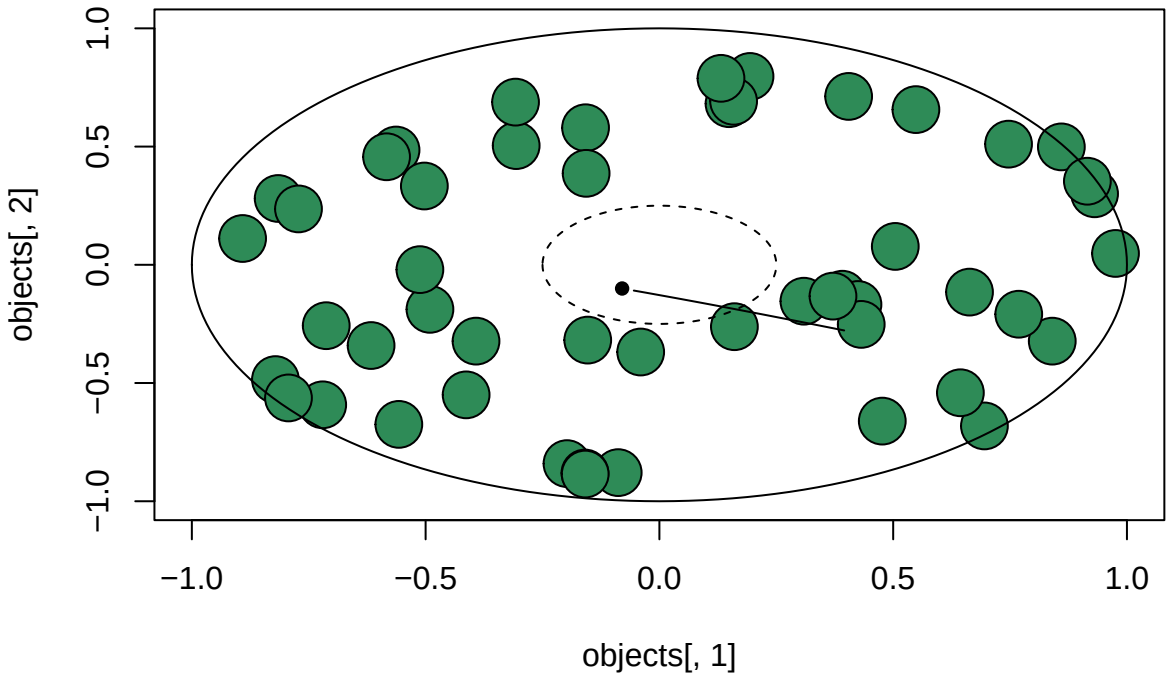
Proportion.of.successful.navigations
0.99

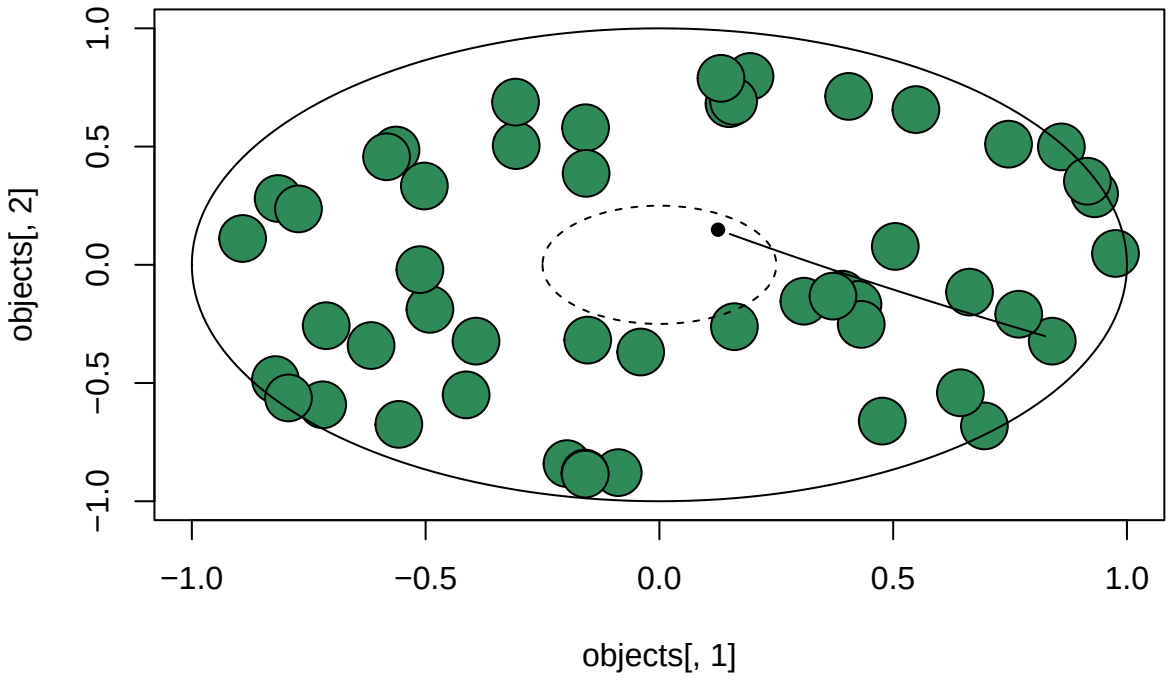
ii

```
winNewObjs = c()
objectsNew = draw_objects(J)
for(i in 1:100)
{

  #objects = draw_objects(J)
  xt_try    = draw_starts()
  res_final = play(xt_try,delt,objectsNew,rad,theta_hat,plt = FALSE,trace = TRUE)
  #print(typeof(res_final$trajectories))

  trajs = na.omit(res_final$trajectories)

  if (i == 1 || i == 50 || i == 100)
  {
```

```
      plot_game(xt_try,objectsNew,rad,'black')
      lines(trajs[, 2,]~trajs[, 1, ])
  }

  winNewObjs[i] = (res_final$status==1)
}
```




objects[, 2]
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
objects[, 1]




```
propNewWins = data.frame("Proportion of successful navigations" = mean(winNewObjs))
kable(propNewWins)
```

Proportion.of.successful.navigations
0

iii

Can see that when measuring performance against the same trees, the model performs relatively well with a proportion of 0.73 passing. However, when using new trees, the model performs much worse with only a proportion of 0.21 passing. This means that the model has not learned how to avoid objects, but simply how to avoid the given set of objects.

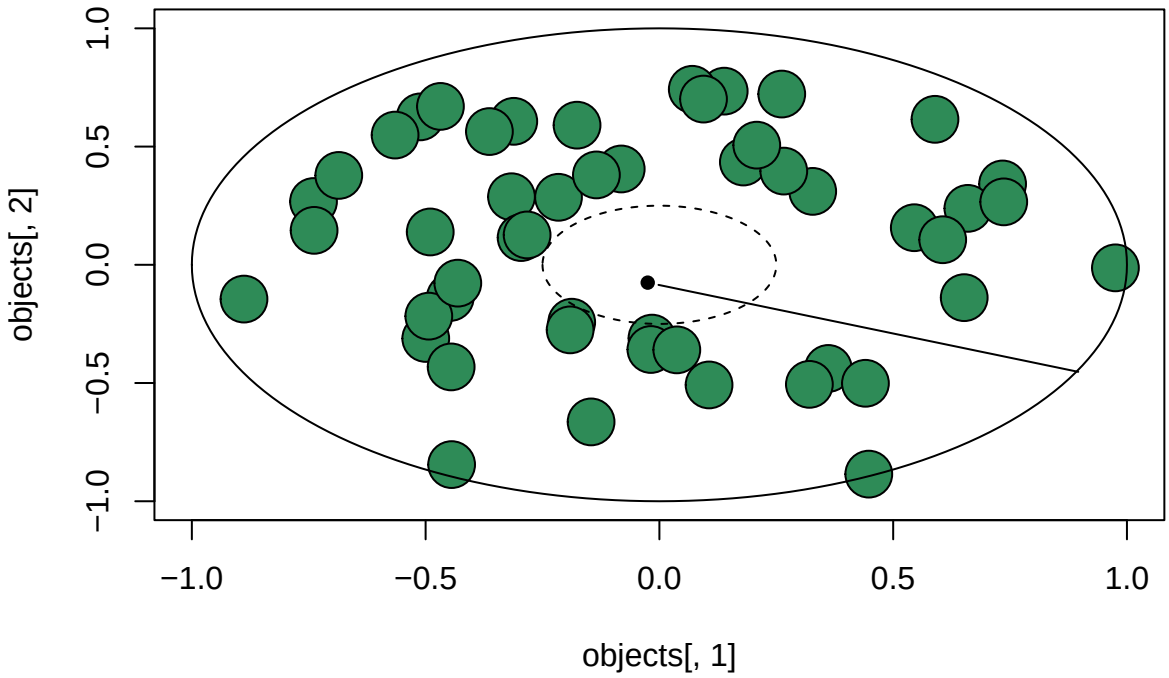
Q1c

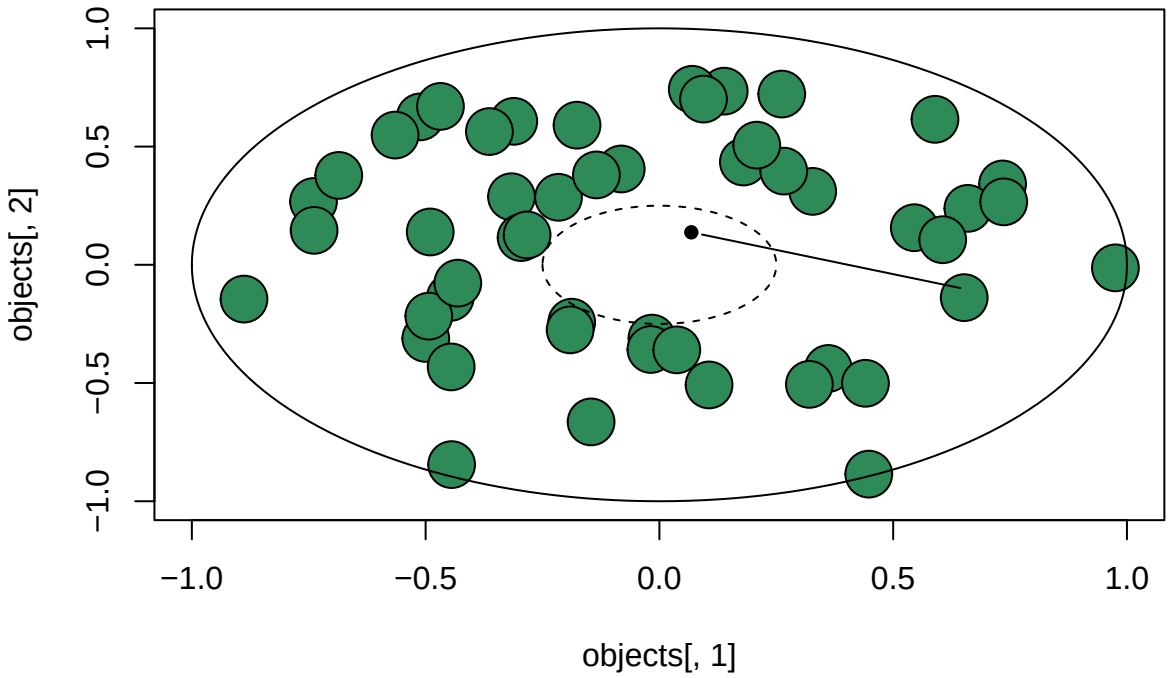
i

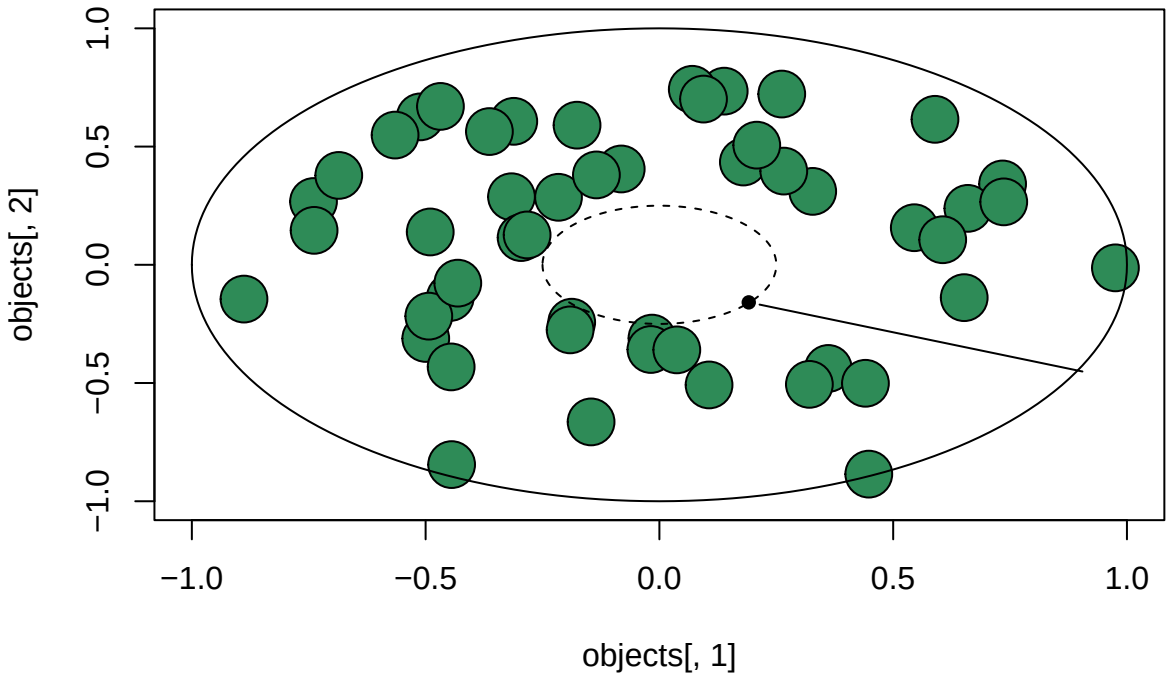
```
playAdj = function(x0,delt,objects,rad,theta,plt = FALSE,trace = FALSE)
{
  k           =   0 # Count how many steps
  xt          = x0   # Set the initial coordinate(s) for the drone.
```

```
  trajectories = NULL
  # Check the game status:
  #print(xt)
  #print(rad)
  res_status  = game_status(xt, objects, rad)
  status      = res_status$status
  minDists = res_status$minDist
  #print(minDists)

  inverseDist = 1/(minDists - rad)
  # Check which games are still active:
  terminal      = (status != 0)
  if(trace)
  {
    trajectories = array(dim = c(dim(xt)[1],dim(xt)[2],101))
    trajectories[,,1] = xt
  }
  if(plt){plot_game(xt,objects,rad,'black')}
  while((any(status==0))&(k<100))
  {
    k = k + 1

    # Now, let the model update the position of the pieces:
    #print("1")
    ct = controlAdjusted(xt, theta, inverseDist)
    #print(ct)
    xt = xt+ct*delt*cbind(1-terminal,1-terminal)

    # Checkk the game status after the positions are updates:
    res_status  = game_status(xt, objects, rad)
    status      = res_status$status
    minDists = res_status$minDist
    inverseDist = 1/(minDists-rad)
    terminal    = (status != 0)
    if(trace){trajectories[,,k] = xt}
    if(plt){plot_game(xt,objects,rad,c('red','black','green')[status+2])}
  }
  return(list(k = k, status = status,xt= xt,trajectories = trajectories))
}


controlAdjusted = function(xt,pars, inverseDist)
{
  xt_aug = cbind(xt, inverseDist)
  res_model = model(xt_aug,pars,rep(nodes,2))
  #print(res_model)
  return(res_model$a3)
}

play_a_game_adj = function(theta)
{
  xt    = draw_starts()
  res   = playAdj(xt,delt,objects,rad,theta, trace = TRUE)
```

```
  score = mean(res$status==1)
  return(score)
}

p     = 3
q     = 2
nodes = 3
npars = p*nodes+nodes*nodes+nodes*q+nodes+nodes+q
# npars
theta_rand = runif(npars,-1,1)

objAdj = play_a_game_adj
GA_adj  = ga(type = 'real-valued',fitness = play_a_game_adj,lower = rep(-10,npars),upper = rep(10,npars)
# plot(GA_adj)


theta_hat_adj = GA_adj@solution[1,]
# theta_hat_adj

win_adj = c()
for(i in 1:100)
{

  #objects = draw_objects(J)
  xt_try    = draw_starts()
  res_final = playAdj(xt_try,delt,objects,rad,theta_hat_adj,plt = FALSE,trace = TRUE)
  #print(typeof(res_final$trajectories))

  trajs = na.omit(res_final$trajectories)

  if (i == 1 || i == 50 || i == 100)
  {
      plot_game(xt_try,objects,rad,'black')
      lines(trajs[, 2,]~trajs[, 1, ])
  }

  win_adj[i] = (res_final$status==1)
}
```


objects[, 2]
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
objects[, 1]


objects[, 2]
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
objects[, 1]

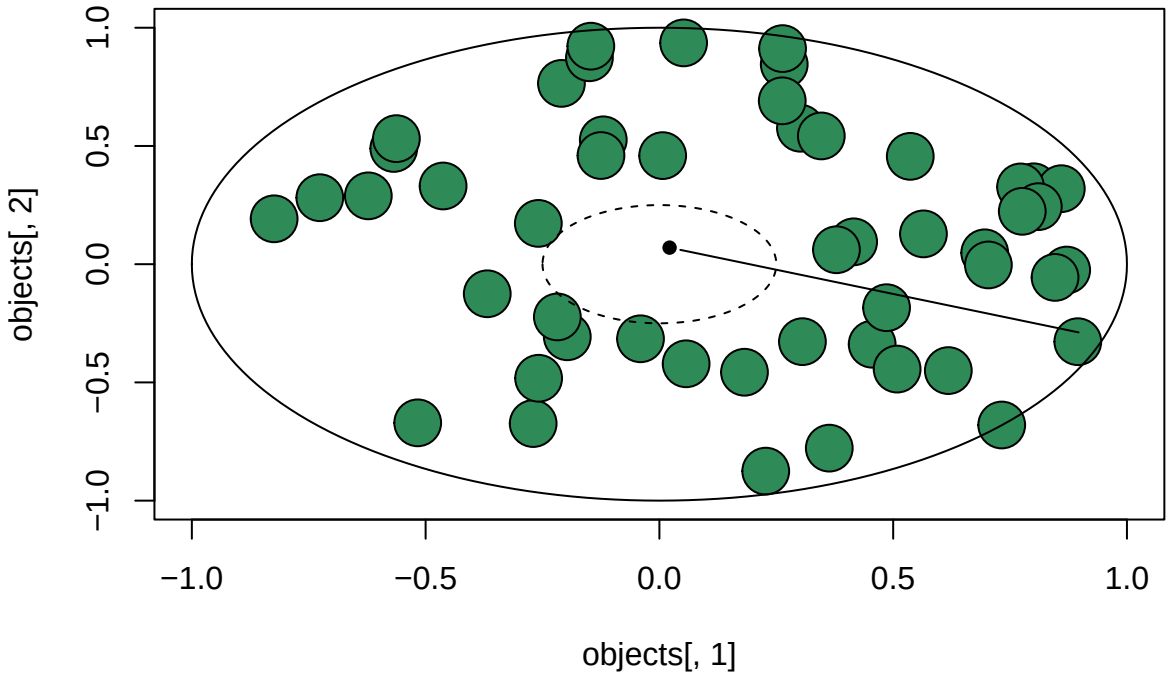


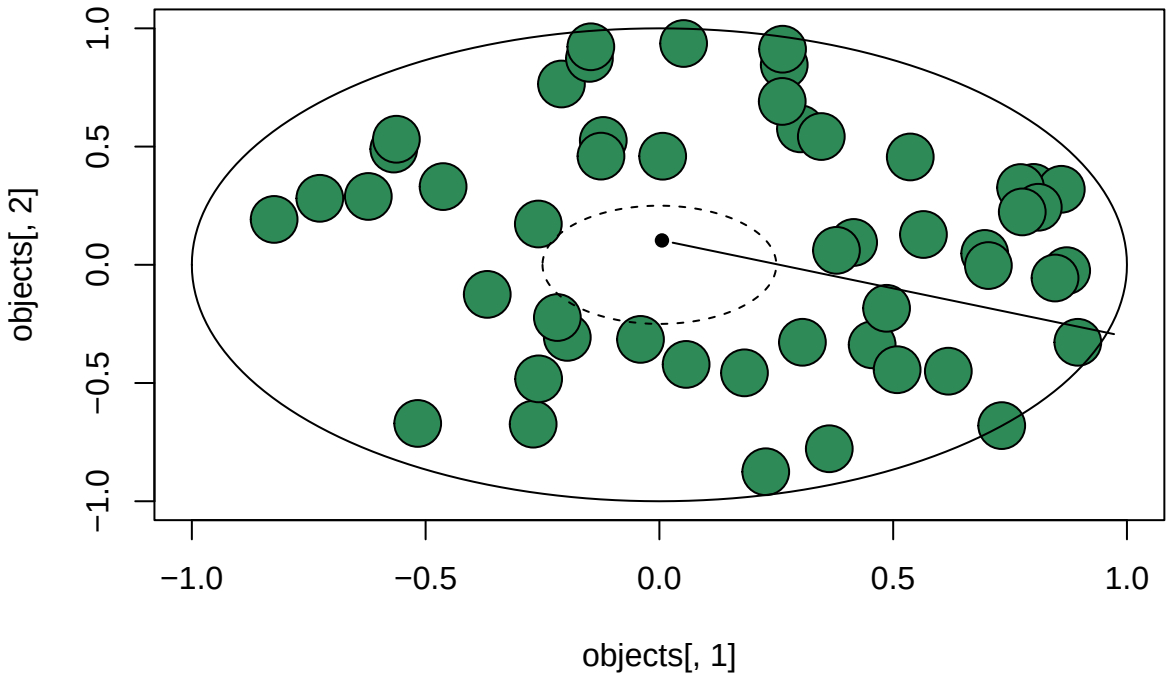

```
propWinsAdj = data.frame("Proportion of successful navigations" = mean(win_adj))
kable(propWinsAdj)
```

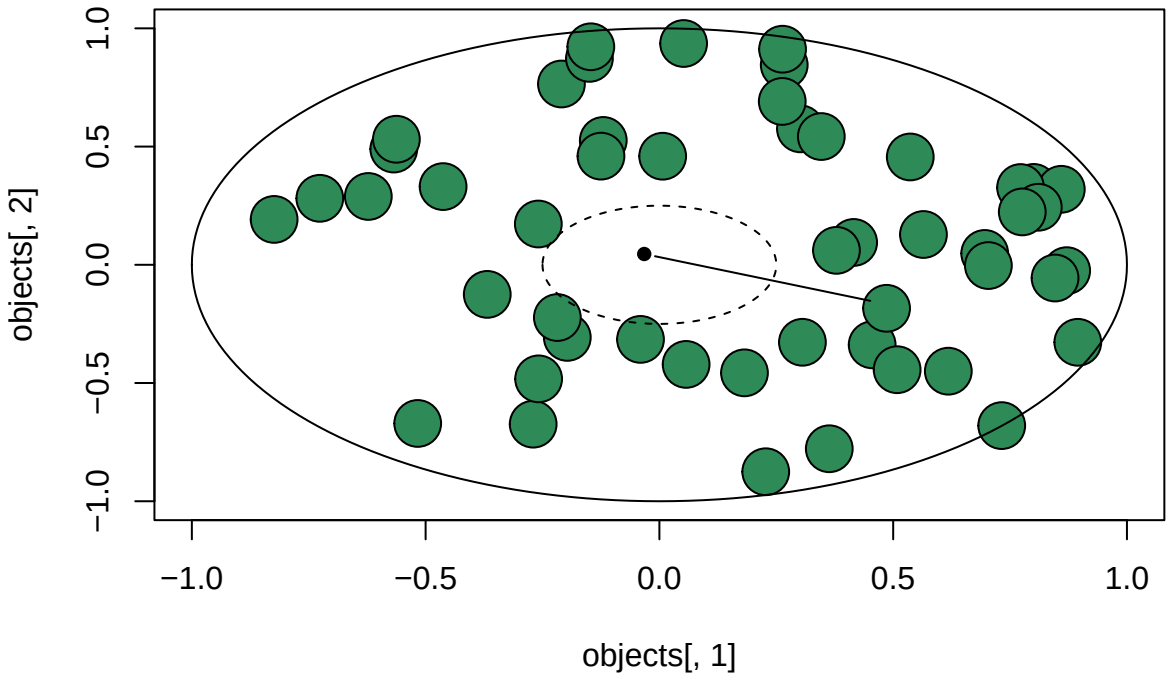
Proportion.of.successful.navigations
0.81

```
winNewObjs_adj = c()

objectsNew = draw_objects(J)
for(i in 1:100)
{
  xt_try    = draw_starts()

  res_final = playAdj(xt_try,delt,objectsNew,rad,theta_hat_adj,plt = FALSE,trace = TRUE)
  #print(typeof(res_final$trajectories))

  trajs = na.omit(res_final$trajectories)

  if (i == 1 || i == 50 || i == 100)
  {
      plot_game(xt_try,objectsNew,rad,'black')
      lines(trajs[, 2,]~trajs[, 1, ])
  }
```

```
  winNewObjs_adj[i] = (res_final$status==1)
}
```




objects[, 2]
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
objects[, 1]




```
#mean(winNewObjs_adj)
propNewWinsAdj = data.frame("Proportion of successful navigations" = mean(winNewObjs_adj))
kable(propNewWinsAdj)
```

Proportion.of.successful.navigations
0.24

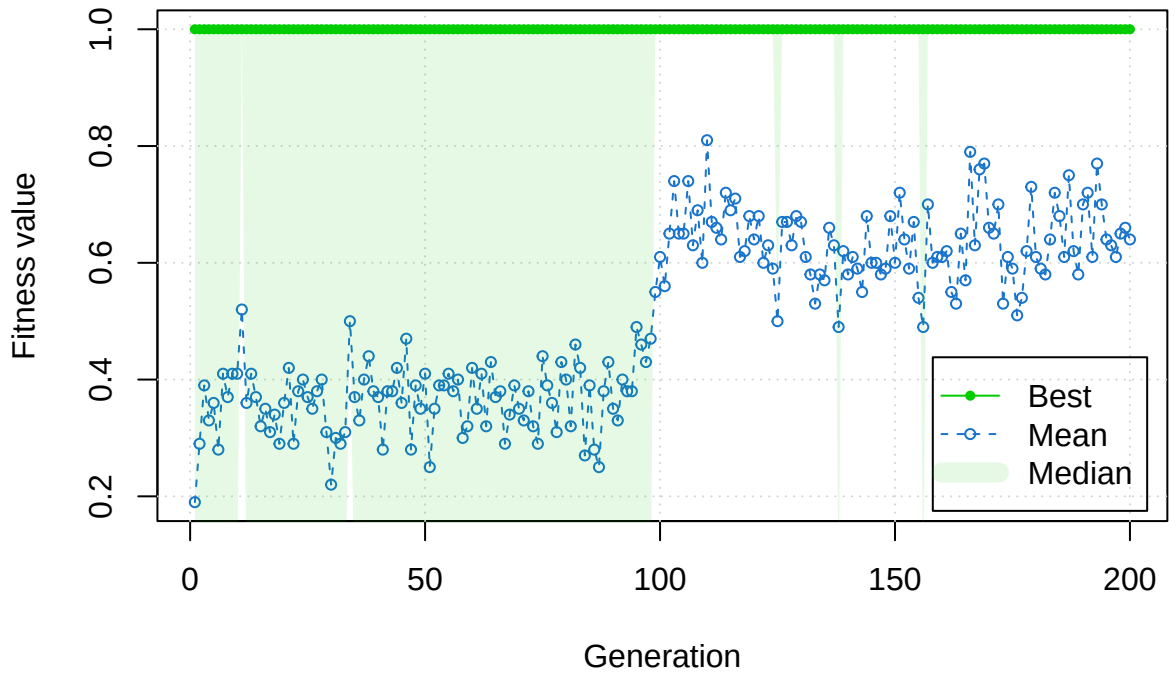
ii

We can see that in sample win probability has increased by , from to . Similarly, when looking at win probability with new objects, we note an increase as well from , to .

Q1d

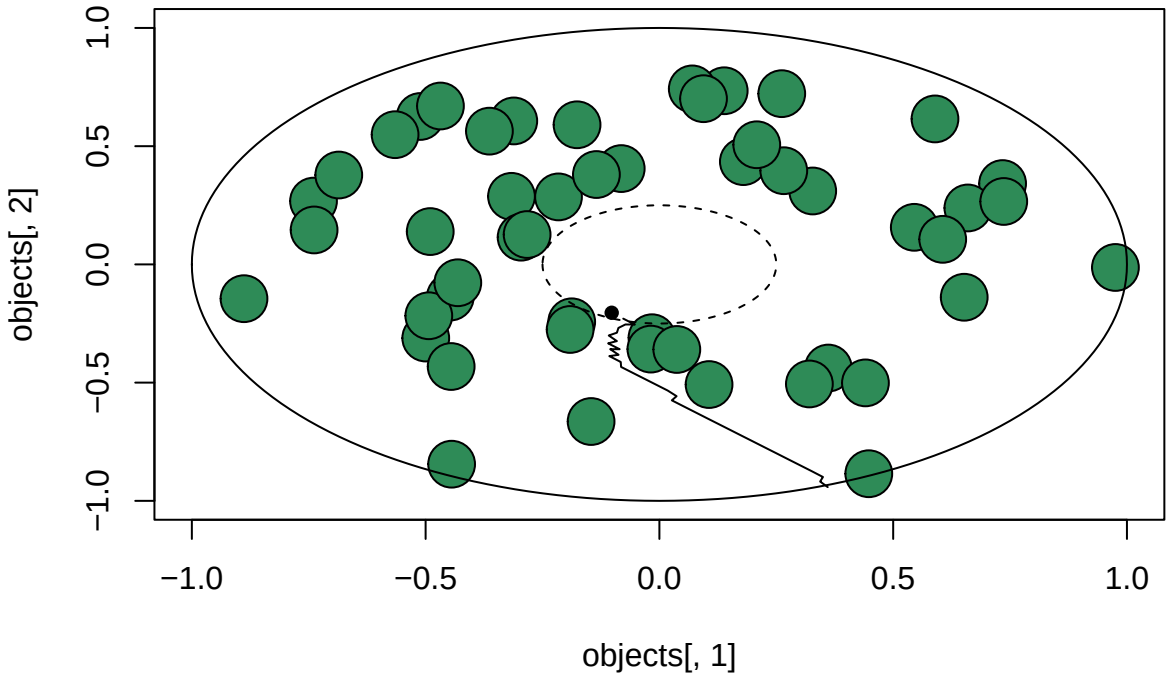
```
### d
play_a_game_adj_newObs = function(theta)
{
  xt     = draw_starts()
  objects = draw_objects(J)
  res    = playAdj(xt,delt,objects,rad,theta, trace = TRUE)
  score = mean(res$status==1)
```

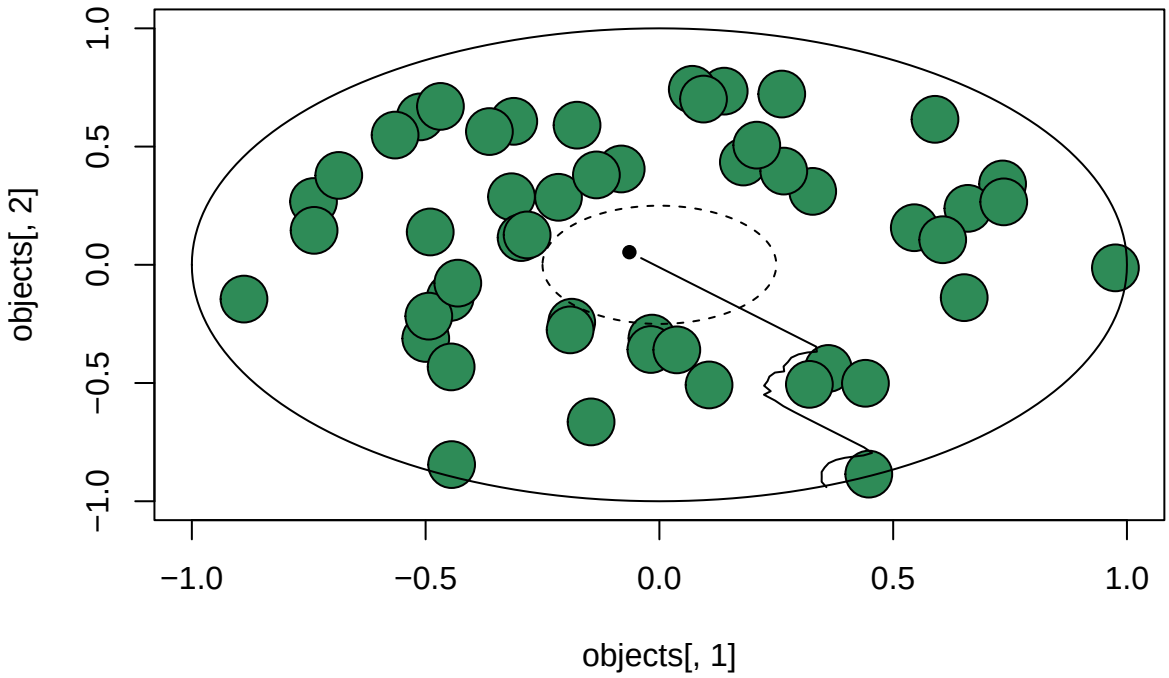
```
  return(score)
}


objAdj = play_a_game_adj_newObs
GA_adj_newObs  = ga(type = 'real-valued',fitness = play_a_game_adj_newObs,lower = rep(-10,npars),upper =
plot(GA_adj_newObs)
```

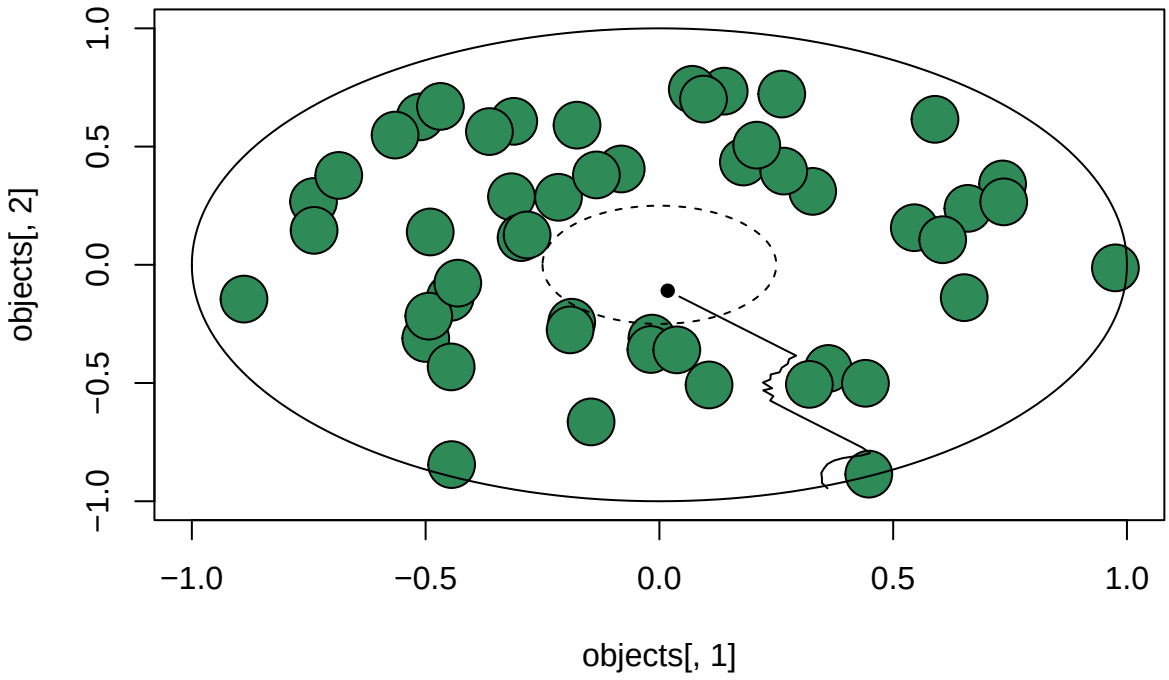



```
theta_hat_adj_newObs = GA_adj_newObs@solution[1,]
# theta_hat_adj_newObs

win_adj_newObs = c()
for(i in 1:100)
{

  #objects = draw_objects(J)
  xt_try    = draw_starts()
  res_final = playAdj(xt_try,delt,objects,rad,theta_hat_adj_newObs,plt = FALSE,trace = TRUE)
  #print(typeof(res_final$trajectories))

  trajs = na.omit(res_final$trajectories)

  if (i == 1 || i == 50 || i == 100)
  {
      plot_game(xt_try,objects,rad,'black')
```

```
      lines(trajs[, 2,]~trajs[, 1, ])
  }

  win_adj_newObs[i] = (res_final$status==1)
}
```



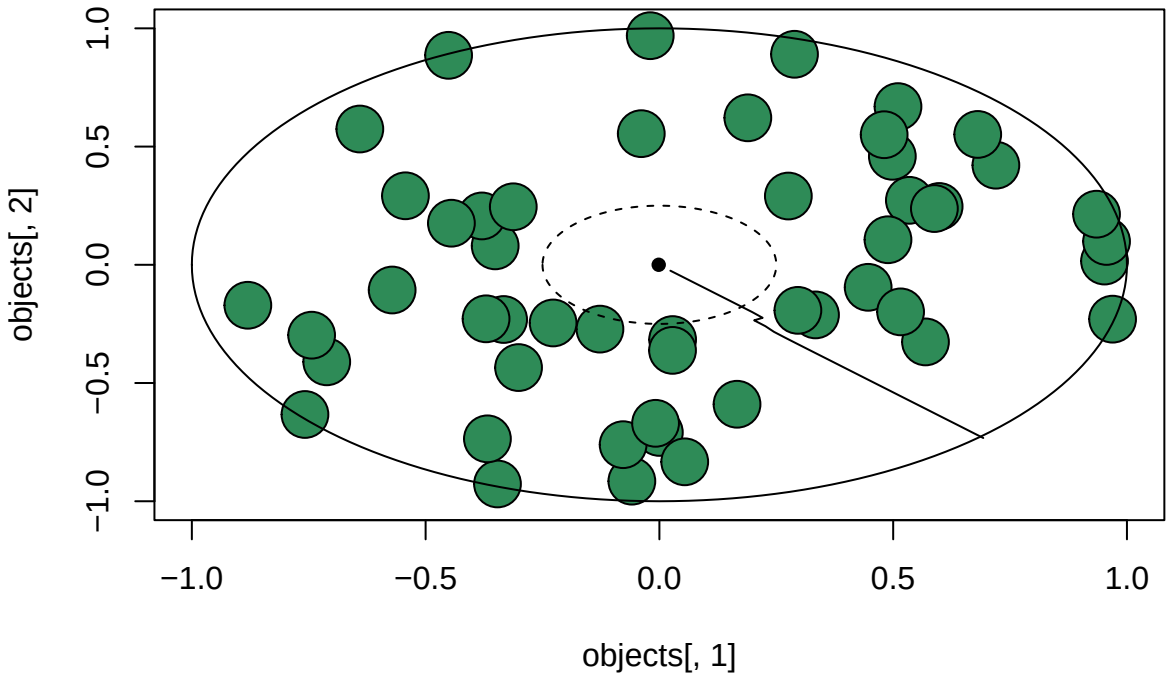

objects[, 2]
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
objects[, 1]

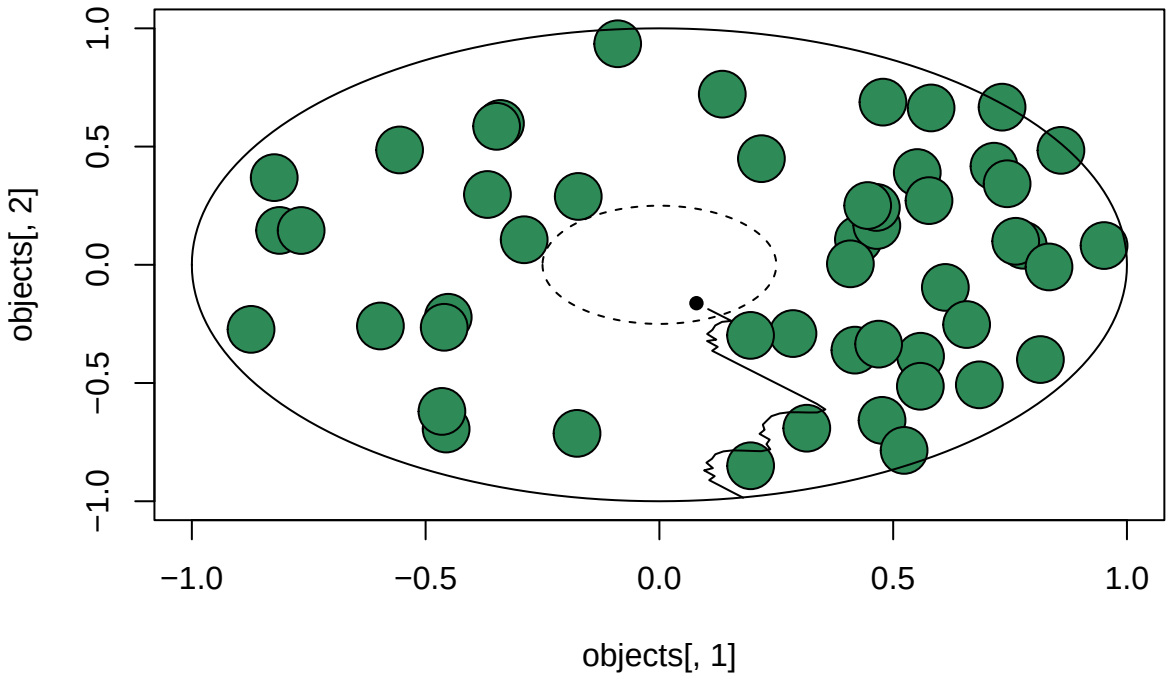


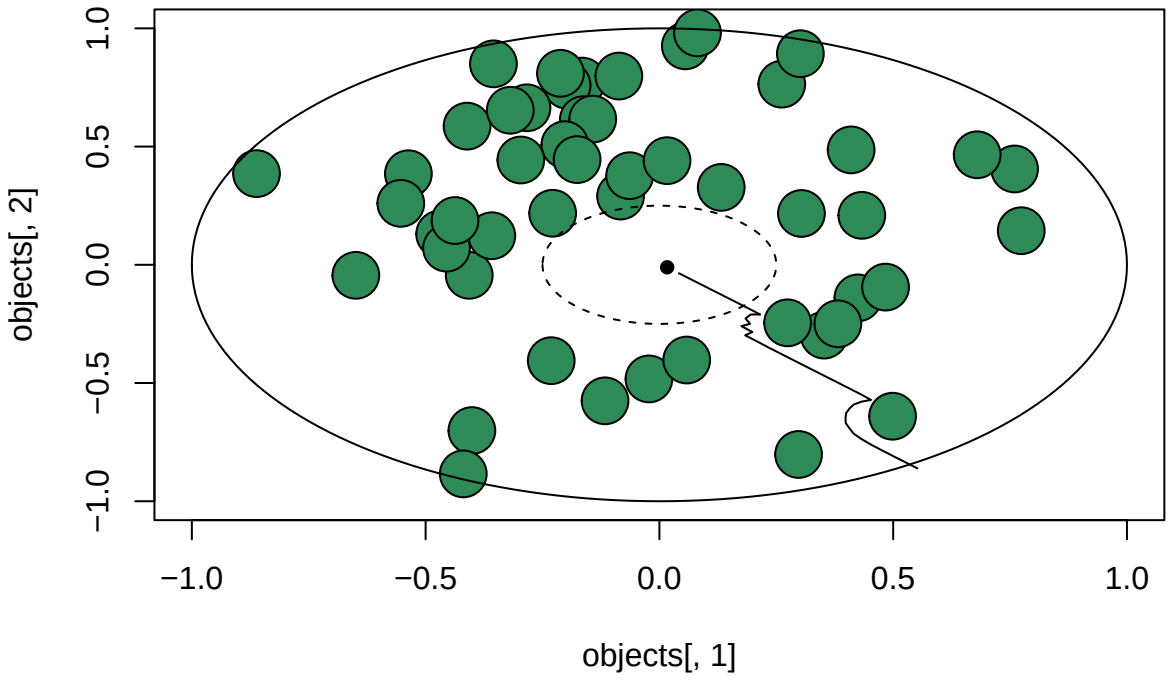

```
prop_win_adj_newObs = data.frame("Proportion of successful navigations" = mean(win_adj_newObs))
kable(prop_win_adj_newObs)
```

Proportion.of.successful.navigations
0.88

```
winNewObjs_adj_genNewObs = c()

for(i in 1:100)
{

  objects = draw_objects(J)
  xt_try    = draw_starts()
  res_final = playAdj(xt_try,delt,objects,rad,theta_hat_adj_newObs,plt = FALSE,trace = TRUE)
  #print(typeof(res_final$trajectories))

  trajs = na.omit(res_final$trajectories)
  if (i == 1 || i == 50 || i == 100)
  {
      plot_game(xt_try,objects,rad,'black')
      lines(trajs[, 2,]~trajs[, 1, ])
  }
  winNewObjs_adj_genNewObs[i] = (res_final$status==1)
}
```


objects[, 2]
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
objects[, 1]


objects[, 2]
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
objects[, 1]




```
prop_winNewObjs_adj_genNewObs = data.frame("Proportion of successful navigations" = mean(winNewObjs_adj
kable(prop_winNewObjs_adj_genNewObs)
```

Proportion.of.successful.navigations
0.65

The in sample win probability has now decreased, as we are taking new samples each time we play a game. However, we can see that the win probability increases out of sample, after we take new

Q2

Q2a

```
game_tree= function(m,k, currMoves = 0, maxMoves = 9)
{
  g = c()
  game_state = rho(m,S)
  if(game_state$terminal)
  {
    g = c(g,game_state$winner)
    return(g)
```

```
  }
  else if(currMoves == maxMoves)
  {
    g = c(g, 2)
    return(g)
  }

  else{
    Index = which(m == 0)
    for(i in 1:length(Index))
    {
      x = m
      x[Index[i]]=k
      g = c(g,game_tree(x,-1*k, currMoves + 1, maxMoves))
    }
    return(g)
  }
}

m = as.matrix(c(0,0,0,0,0,0,0,0,0))

res5 = game_tree(m,1, 0, 5)
n_g5  =     length(res5)
Xwins5 = sum(res5==-1)
Draws5 = sum(res5== 0)
Owins5 = sum(res5==+1)
Unfinished5 = sum(res5==+2)
#Games5 = c(n_g5,Xwins5,Draws5,Owins5, Unfinished5)
games5 = data.frame("Number of moves" = 5, "Number of games" = n_g5, "X wins" = Xwins5, "O wins" = Owins

res8 = game_tree(m,1, 0, 8)
n_g8  =     length(res8)
Xwins8 = sum(res8==-1)
Draws8 = sum(res8== 0)
Owins8 = sum(res8==+1)
Unfinished8 = sum(res8==+2)
#Games8 = c(n_g8,Xwins8,Draws8,Owins8, Unfinished8)
games8 = data.frame("Number of moves" = 8, "Number of games" = n_g8, "X wins" = Xwins8, "O wins" = Owins

res9 = game_tree(m,1, 0, 9)
n_g9  =     length(res9)
Xwins9 = sum(res9==-1)
Draws9 = sum(res9== 0)
Owins9 = sum(res9==+1)
Unfinished9 = sum(res9==+2)
CombinedFull = data.frame("Number of Games" = n_g9,"X wins" = Xwins9,"Draws" = Draws9, "O wins" = Owins9
CombinedFullWinProb = data.frame("Prop X wins" = Xwins9/n_g9,"Prop Draws" = Draws9/n_g9, "Prop O wins" =

datQ2A = rbind(games5, games8)
kable(datQ2A)
```

Number.of.moves	Number.of.games	X.wins	O.wins	Draws
5	15120	0	1440	0
8	255168	77904	49392	23040

It is assumed that O moves first. From the table, we can see that there is 1440 wins in exactly 5 moves and 23040 draws in exactly 8 moves. Interestingly, it is better to play second if there is a maximum of 8 moves available to be played, as X has greater number of possible wins. It is better to play second, likely because if the O has not won the game in 5 moves, then X is able to win on move 6 or move 8 whereas O is only able to win on move 7.

Q2b

```
mcts = function(m, k, alpha)
{
  g = c()
  game_state = rho(m,S)

  randAlpha = runif(1, 0, 1)

  if(game_state$terminal)
  {
    g = c(g,game_state$winner)
    return(g)
  }

  else
  {
    if (randAlpha <= alpha)
    {
      Index = which(m == 0)
      for(i in 1:length(Index))
      {
        x = m
        x[Index[i]]=k
        g = c(g,mcts(x,-1*k, alpha))
      }
      return(g)
    }
    else
    {
      g = c(g,2)
      return(g)
    }

  }
}

m = as.matrix(c(1,1,0,0,0,0,0,0,-1))

resMCTS = mcts(m,-1, 0.95)
n_gMCTS  =  length(resMCTS)
```

```
XwinsMCTS = sum(resMCTS==-1)
DrawsMCTS = sum(resMCTS== 0)
OwinsMCTS = sum(resMCTS==+1)
UnfinishedMCTS = sum(resMCTS==+2)
MCTS = data.frame("Games" = n_gMCTS,"X Wins" = XwinsMCTS, "Draws" = DrawsMCTS, "O wins" = OwinsMCTS, "U
MCTSWinProb = data.frame("Prop X wins" = XwinsMCTS/n_gMCTS, "Prop Draws" = DrawsMCTS/n_gMCTS,"Prop O wi

kable(MCTS, caption = "Table of results using Monte Carlo Tree Search, alpha = 0.95")
```

Table 8: Table of results using Monte Carlo Tree Search, alpha = 0.95

Games	X.Wins	Draws	O.wins	Unfinished.Games
402	146	64	171	21

```
kable(CombinedFull, caption = "Table of results using Tree Search")
```

Table 9: Table of results using Tree Search

Number.of.Games	X.wins	Draws	O.wins	Unfinished.games
255168	77904	46080	131184	0

```
kable(MCTSWinProb, caption = "Table of proportions using Monte Carlo Tree Search, alpha = 0.95")
```

Table 10: Table of proportions using Monte Carlo Tree Search, alpha = 0.95

Prop.X.wins	Prop.Draws	Prop.O.wins
0.3631841	0.159204	0.4253731

```
kable(CombinedFullWinProb, caption = "Table of proportions using Tree Search")
```

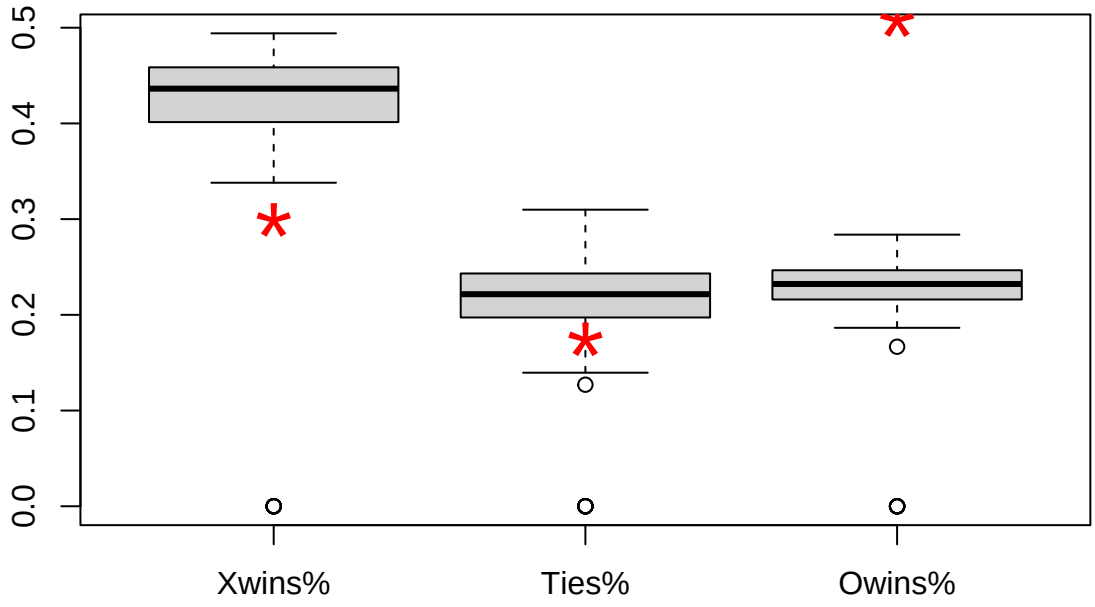
Table 11: Table of proportions using Tree Search

Prop.X.wins	Prop.Draws	Prop.O.wins
0.3053047	0.1805869	0.5141084

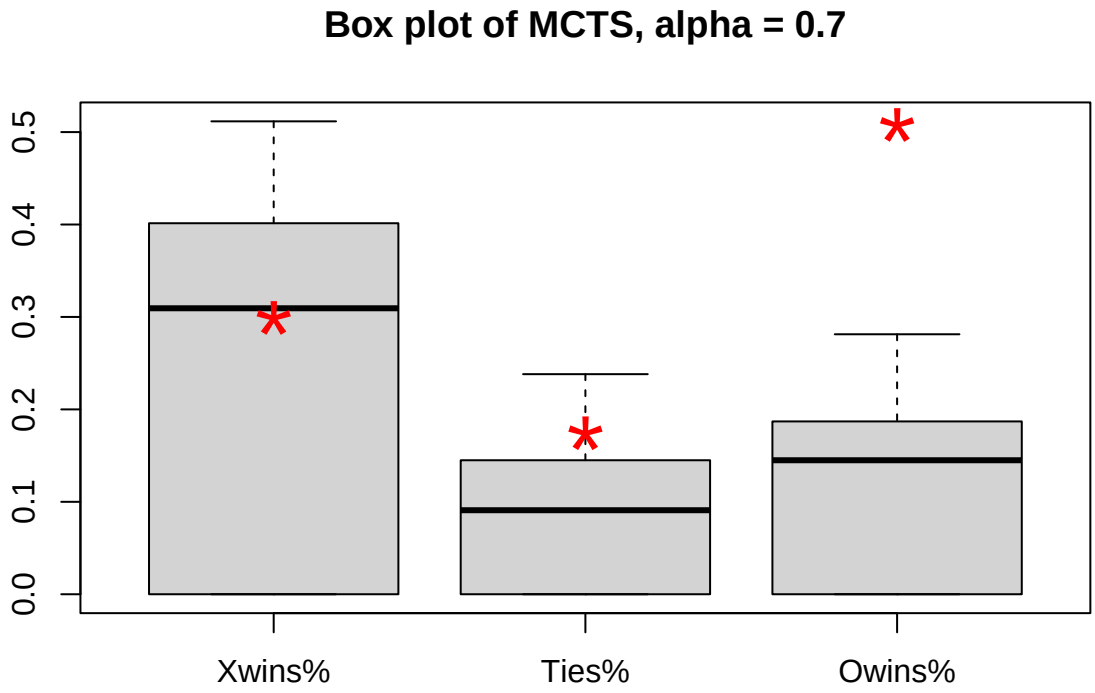
Can see that there are quite large differences between the true value and the estimated value. X wins are overestimated by 26%, O wins are underestimated by 20% and draws are underestimated by 16%. Considering these are points estimates, there is large amounts of volatility in the results and so would make sense that they are inaccurate.

Q2c

```
m = as.matrix(c(1,1,-1,0,0,0,0,0,-1))

df90 = data.frame()
df70 = data.frame()

for(i in 1:100)
{
  resMCTS90 = mcts(m,1, 0.9)
  resMCTS70 = mcts(m,1, 0.7)

  n_gMCTS90  =  length(resMCTS90)
  XwinsMCTS90 = sum(resMCTS90==-1)
  DrawsMCTS90 = sum(resMCTS90== 0)
  OwinsMCTS90 = sum(resMCTS90==+1)
  UnfinishedMCTS90 = sum(resMCTS==+2)
  Combined90 = c(n_gMCTS90,XwinsMCTS90/n_gMCTS90,DrawsMCTS90/n_gMCTS90,OwinsMCTS90/n_gMCTS90, Unfinishe

  n_gMCTS70  =  length(resMCTS70)
  XwinsMCTS70 = sum(resMCTS70==-1)
  DrawsMCTS70 = sum(resMCTS70== 0)
  OwinsMCTS70 = sum(resMCTS70==+1)
  UnfinishedMCTS70 = sum(resMCTS70==+2)
  Combined70 = c(n_gMCTS70,XwinsMCTS70/n_gMCTS70,DrawsMCTS70/n_gMCTS70,OwinsMCTS70/n_gMCTS70, Unfinishe

  df90 = rbind(df90, Combined90)
  df70 = rbind(df70, Combined70)
}

colnames(df90) = c("Games", "Xwins%", "Ties%", "Owins%", "Incomplete")
colnames(df70) = c("Games", "Xwins%", "Ties%", "Owins%", "Incomplete")

boxplot(df90[, 2:4], main = "Box plot of MCTS, alpha = 0.9")
points(c(CombinedFullWinProb$Prop.X.wins, CombinedFullWinProb$Prop.Draws, CombinedFullWinProb$Prop.O.wi
```




```
boxplot(df70[, 2:4], main = "Box plot of MCTS, alpha = 0.7")
points(c(CombinedFullWinProb$Prop.X.wins, CombinedFullWinProb$Prop.Draws, CombinedFullWinProb$Prop.O.wi
```



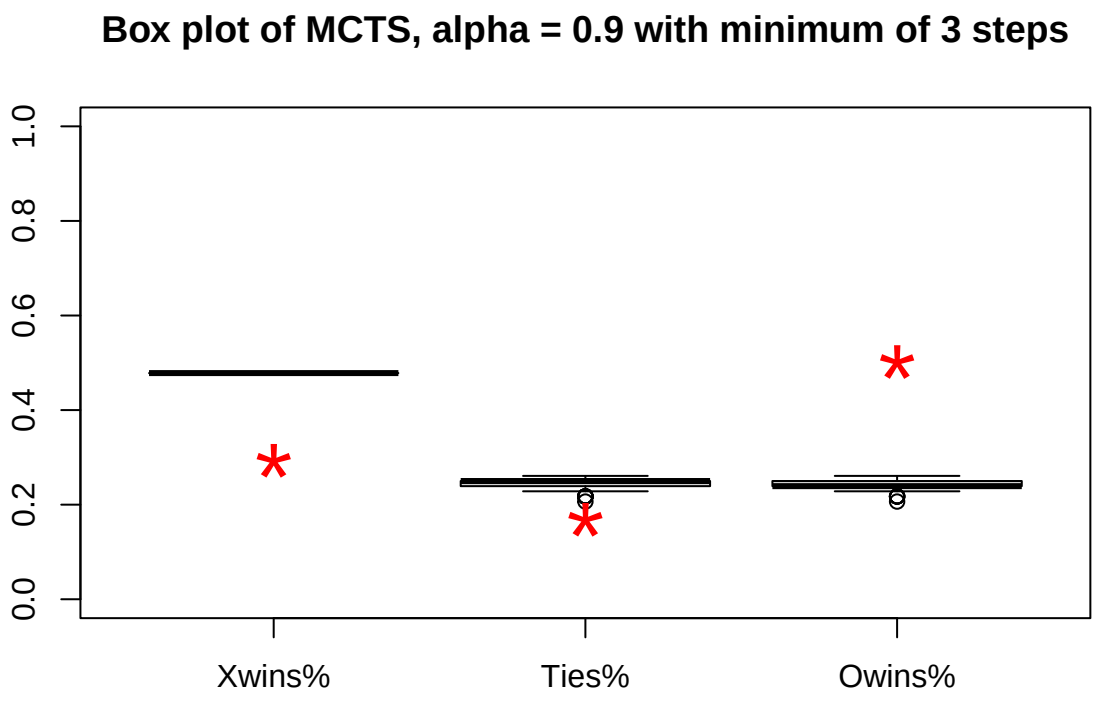
Can see that under alpha = 0.7, the median correctly captures the proportion of X wins, and the number of draws is within the range, and just outside the upper quartile. Under alpha = 0.9, we can see that X wins falls just outside the range, and considered an outlier while draws fall just outside the lower quartile. Under both alphas, we can see we underestimate the number of O wins. This is likely due to most O wins occuring towards the end of the game, which is not evaluated as frequently, as there is only a 60% chance that a branch is evaluated completely under alpha = 0.9 and 0.17 under alpha = 0.7. Therefore we get O wins being consistently under estimated.

The distribtion of alpha = 0.7, does appear to slightly perform better in estimating the win probabilities. Draws occur mostly early on in the game, and thus is has a much greater accuracy due to not having to evaluate many long branches.

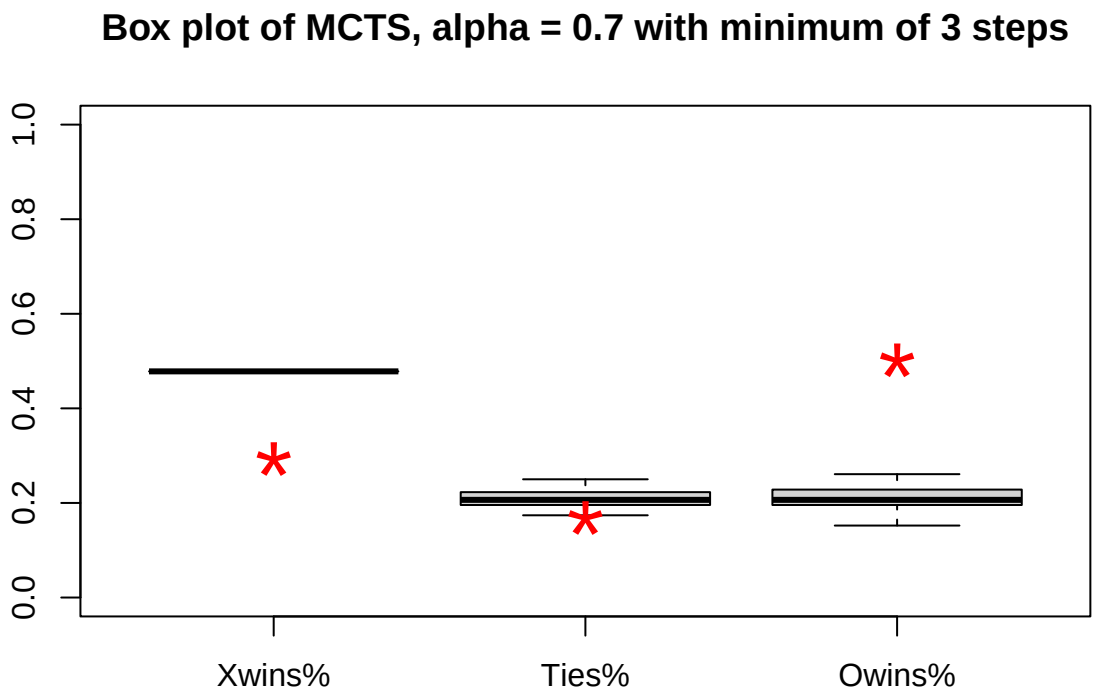
```
mcts_minBranch = function(m, k, alpha, currMoves, minBranch)
{
  g = c()
  game_state = rho(m,S)

  randAlpha = runif(1, 0, 1)

  if(game_state$terminal)
  {
    g = c(g,game_state$winner)
    return(g)
  }

  else
  {
```

```
    if (currMoves <= minBranch)
    {
      Index = which(m == 0)
      for(i in 1:length(Index))
      {
        x = m
        x[Index[i]]=k
        g = c(g,mcts_minBranch(x,-1*k, alpha, currMoves + 1, minBranch))
      }
      return(g)
    }
    else
    {
      if (randAlpha <= alpha)
      {
        Index = which(m == 0)
        for(i in 1:length(Index))
        {
          x = m
          x[Index[i]]=k
          g = c(g,mcts_minBranch(x,-1*k, alpha, currMoves + 1, minBranch))
        }
        return(g)
      }
      else
      {
        g = c(g,2)
        return(g)
      }
    }

  }
}

# m = as.matrix(c(1,1,0,0,0,0,0,0,-1))
#
# resMCTS_Branch = mcts_minBranch(m,-1, 0.95, 3, 9)
# n_gMCTS_Branch  =     length(resMCTS_Branch)
# XwinsMCTS_Branch = sum(resMCTS_Branch==-1)
# DrawsMCTS_Branch = sum(resMCTS_Branch== 0)
# OwinsMCTS_Branch = sum(resMCTS_Branch==+1)
# UnfinishedMCTS_Branch = sum(resMCTS_Branch==+2)
# c(n_gMCTS_Branch,XwinsMCTS_Branch,DrawsMCTS_Branch,OwinsMCTS_Branch, UnfinishedMCTS_Branch)

m = as.matrix(c(1,1,-1,0,0,0,0,0,-1))

df90_Branch_new = data.frame()
df70_Branch_new = data.frame()

for(i in 1:100)
{
  resMCTS90_new = mcts_minBranch(m,1, 0.9, 4, 7)
  resMCTS70_new = mcts_minBranch(m,1, 0.7, 4, 7)
```

```
  n_gMCTS90_new  =  length(resMCTS90_new)
  XwinsMCTS90_new = sum(resMCTS90_new==-1)
  DrawsMCTS90_new = sum(resMCTS90_new== 0)
  OwinsMCTS90_new = sum(resMCTS90_new==+1)
  UnfinishedMCTS90_new = sum(resMCTS90_new==+2)
  Combined90_new = c(n_gMCTS90_new,XwinsMCTS90_new/n_gMCTS90_new,
                     DrawsMCTS90_new/n_gMCTS90_new,OwinsMCTS90_new/n_gMCTS90_new, UnfinishedMCTS90_new)

  n_gMCTS70_new  =  length(resMCTS70_new)
  XwinsMCTS70_new = sum(resMCTS70_new==-1)
  DrawsMCTS70_new = sum(resMCTS70_new== 0)
  OwinsMCTS70_new = sum(resMCTS70_new==+1)
  UnfinishedMCTS70_new = sum(resMCTS70_new==+2)
  Combined70_new = c(n_gMCTS70_new,XwinsMCTS70_new/n_gMCTS70_new,
                     DrawsMCTS70_new/n_gMCTS70_new,OwinsMCTS70_new/n_gMCTS70_new, UnfinishedMCTS70_new)

  df90_Branch_new = rbind(df90_Branch_new, Combined90_new)
  df70_Branch_new = rbind(df70_Branch_new, Combined70_new)
}

colnames(df90_Branch_new) = c("Games", "Xwins%", "Ties%", "Owins%", "Incomplete")
colnames(df70_Branch_new) = c("Games", "Xwins%", "Ties%", "Owins%", "Incomplete")

boxplot(df90_Branch_new[, 2:4], ylim = c(0, 1), main = "Box plot of MCTS, alpha = 0.9 with minimum of 3
points(c(CombinedFullWinProb$Prop.X.wins, CombinedFullWinProb$Prop.Draws, CombinedFullWinProb$Prop.O.wi
```



```
boxplot(df70_Branch_new[, 2:4], ylim = c(0, 1), main = "Box plot of MCTS, alpha = 0.7 with minimum of 3
points(c(CombinedFullWinProb$Prop.X.wins, CombinedFullWinProb$Prop.Draws, CombinedFullWinProb$Prop.O.wi
```



This box plot contains the same MCTS, however there is a minimum number of 3 steps occuring always. We can see that the draws are getting approximately correctly estimated, but the wins are still being under or over estimated, likely giving worse results.

Q3

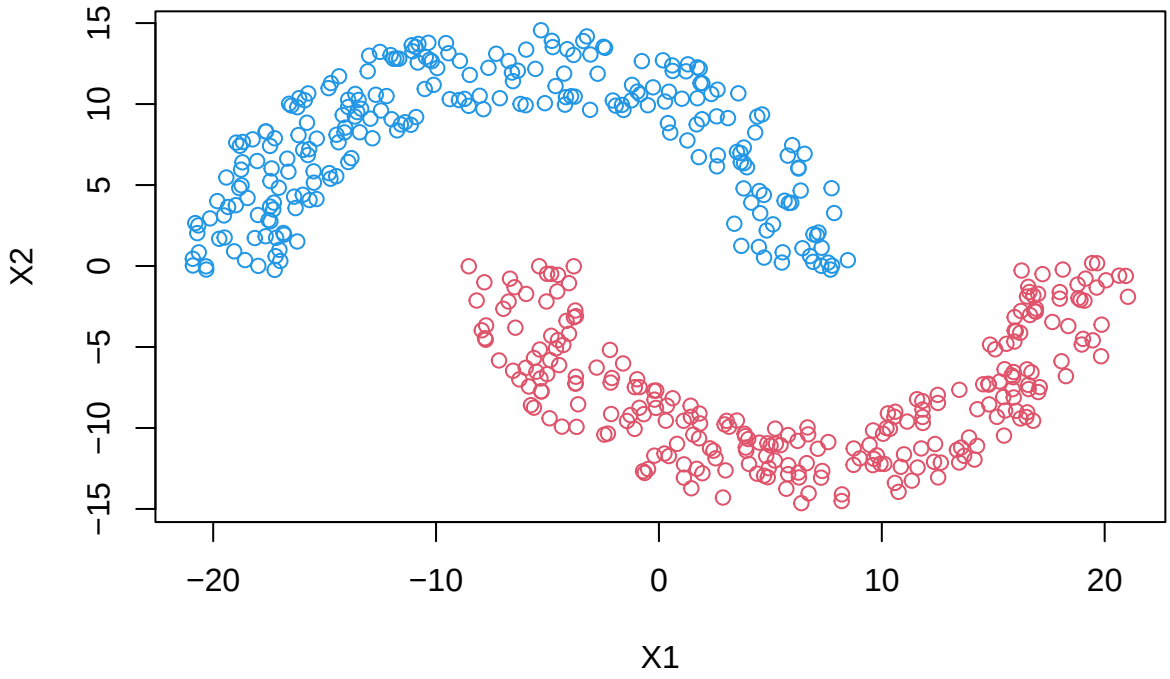
Q3a

```
library(quadprog)

polKern = function(x1, x2, gm, cf, dg)
{
  return((cf+gm*t(x1)%*%x2)^dg)
}

radialKern = function(x1, x2, gm)
{
  return(exp(-gm*(sum((x1 - x2)^2))))
}
```

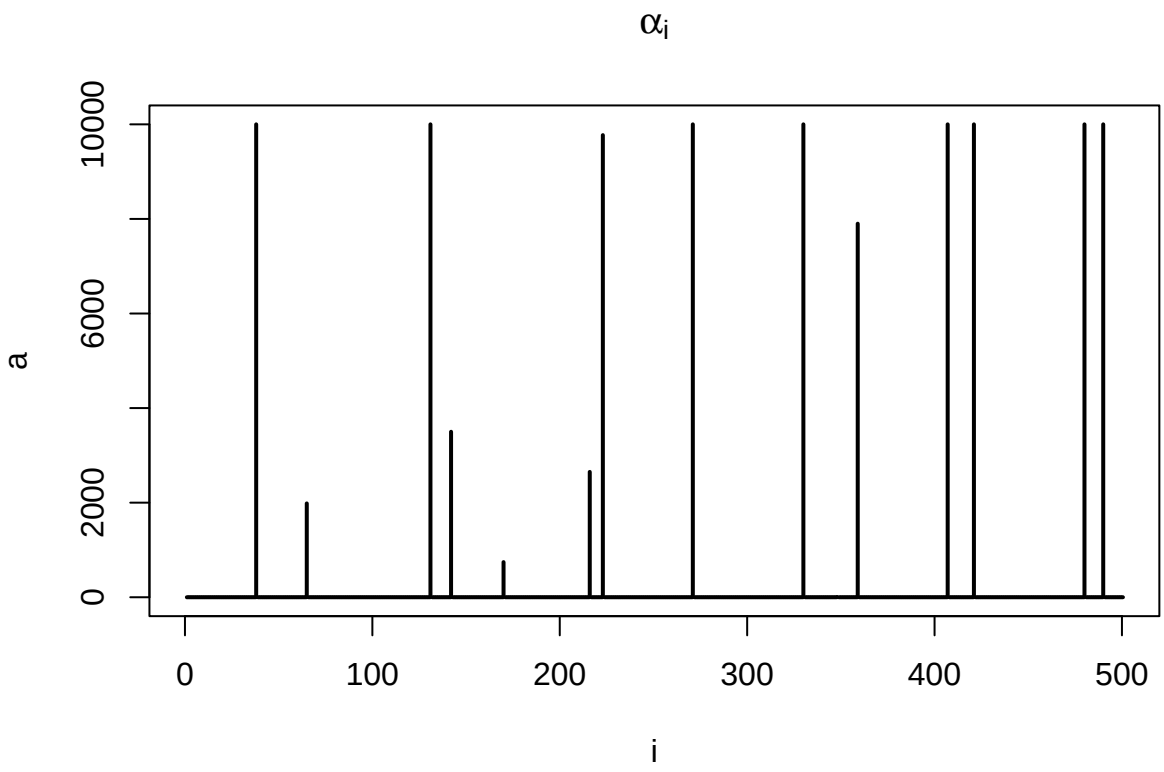
```
my_svm = function(y, x, kern, cost = 0, softMarg = FALSE, gm = 0, cf = 0, dg = 0, plt = FALSE)
{
  N = dim(x)[1]
  DD = matrix(0,N,N)

  if(kern == "none")
  {
    for(i in 1:N)
    {
      for(j in 1:N)
      {
        DD[i,j]  = y[i]*y[j]*(t(x[i,])%*%x[j,])
      }
    }
  }
  else if (kern == "poly")
  {

    for(i in 1:N)
    {
      for(j in 1:N)
      {
        KK = polKern(x[i,], x[j,], gm, cf, dg)
        DD[i,j]  = y[i]*y[j]*KK
      }
    }
  }
  else if (kern == "radial")
  {
    for(i in 1:N)
    {
      for(j in 1:N)
      {
        KK = radialKern(x[i,], x[j,], gm)
        DD[i,j]  = y[i]*y[j]*KK
      }
    }
  }

  eps = 5e-6
  DD  = DD+eps*diag(N)
  Amat = cbind(y,diag(N)) # y will be on first row of t(Amat)
  bvec = matrix(0,N+1,1)
  d    = matrix(1,N,1)

  if (softMarg == TRUE)
  {
    negativeC = (-1)*cost
    Amat = cbind(Amat, -diag(N))
    Cvec = rep(negativeC, N)
    vec0 = rep(0, N+1)
    bvec = matrix(c(vec0, Cvec), ncol = 1, nrow = (2*N + 1))
  }
```

```
#print(dim(Amat))
#print(dim(bvec))

res = solve.QP(Dmat = DD,dvec = d,Amat = Amat,bvec = bvec,meq = 1,factorized = FALSE)
a   = res$solution

if (plt == TRUE)
{
  plot(a,type= 'h', main = expression(alpha[i]),xlab = expression(i), lwd = 2)
}

pad.a     = round(a,3)
wh        = which.max(a)

if (kern == "none")
{
  ww = t(a*y)%*%X
  intercept = 1/y[wh] - X[wh, ]%*%t(ww)

  yhat = sign(X%*%t(ww) + intercept[1])

}
else if (kern == "poly")
{

  T1 = rep(0,N)
  for(i in 1:N)
  {
    KK = polKern(x[i, ], t(X), gm, cf, dg)
    T1[i] = sum(a*y*(KK))
  }
  #print(KK)

  KKNew = polKern(x[wh,], t(x), gm, cf, dg)
  intercept = 1/y[wh]-sum(a*y*KKNew)

  yhat      = sign(T1+intercept[1])
}
else if (kern == "radial")
{
  T1 = rep(0,N)
  for(i in 1:N)
  {
    T1[i] = sum(a*y*(x[i,]%*%t(x)))
  }

  KKNew = radialKern(x[wh,], t(x), gm)
  intercept = 1/y[wh]-sum(a*y*KKNew)

  yhat      = sign(T1+intercept[1])
}

res = list("yhat" = yhat, "padA" = pad.a, "a" = a, "intercept" = intercept)
```

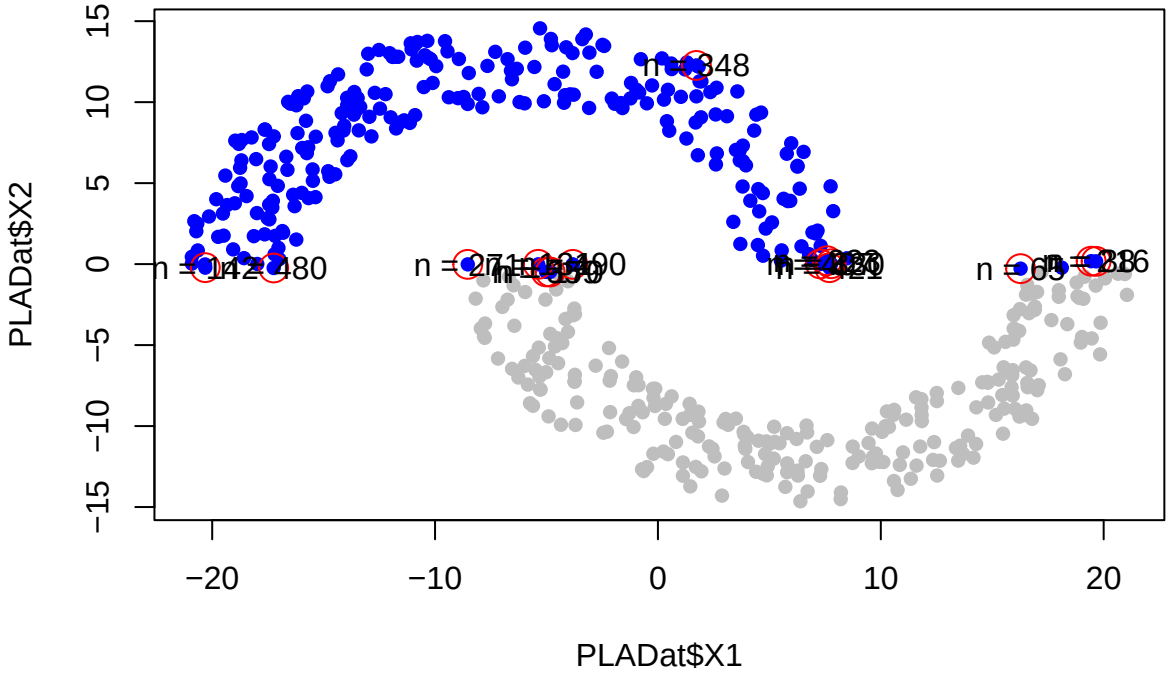
```
  return(res)

}

PLADat = read.table("PLA Dynamics.txt")

plot(X2 ~ X1, col = (Y + 3), data = PLADat)
```




```
# Create data matrix:
X  = cbind(PLADat$X1,PLADat$X2)
Y = PLADat$Y

#Y[Y == -1] = 0

gm = 1
cf = 1
dg = 2

mySVM = my_svm(Y, X, "poly", 10000, TRUE, gm, cf, dg, plt = TRUE)
```




```
yhat = mySVM$yhat
padA = mySVM$padA


#par(mfrow = c(2,2))
#plot(X2~X1,pch = c(1,16)[(Y+1)/2+1], data = PLADat)
plot(PLADat$X2~PLADat$X1, pch = 16, col = c('grey','blue')[(yhat+1)/2+1])

wh.text = which(padA!=0)
points(PLADat$X2~PLADat$X1, pch = 1, col = c(NA,'red')[(padA>0)+1],cex=2)
text(PLADat$X2[wh.text]~PLADat$X1[wh.text],labels = paste0('n = ',wh.text))
```



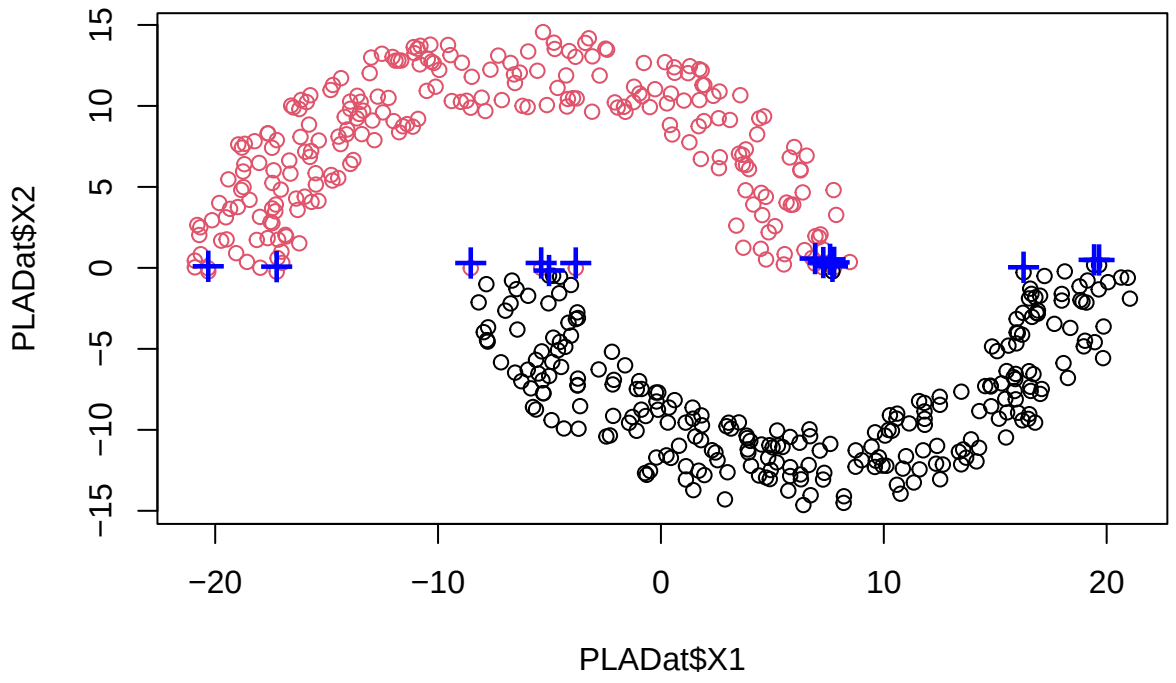
Q3b

```
library('e1071')
```

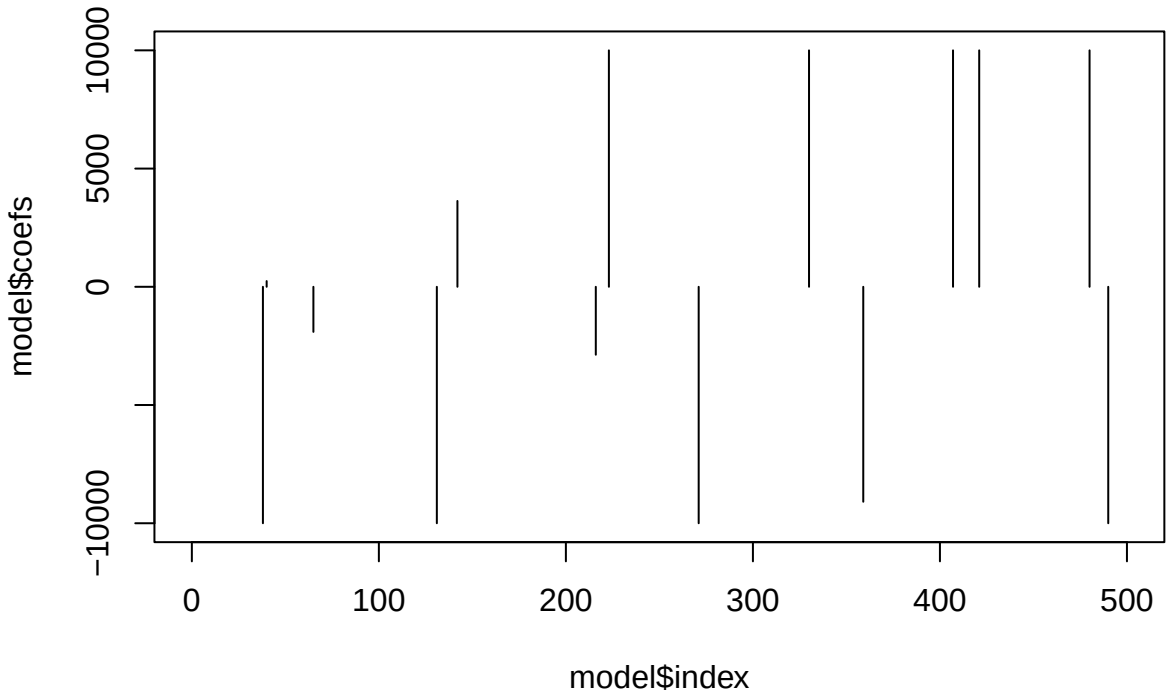
```
## Warning: package 'e1071' was built under R version 4.5.1
```

```
model  = svm(as.factor(Y)~(X1 + X2), data = PLADat, scale = TRUE,kernel = 'polynomial',degree =dg,gamma

yHatSVM = predict(model, data.frame(X1 = PLADat$X1, X2 = PLADat$X2))

# x1_range <- seq(min(PLADat$X1), max(PLADat$X1), length.out = 200)
# x2_range <- seq(min(PLADat$X2), max(PLADat$X2), length.out = 200)
# grid <- expand.grid(X1 = x1_range, X2 = x2_range)
# grid$pred <- predict(model, newdata = grid)
#
# # Add the decision surface
# z <- matrix(as.numeric(grid$pred), nrow = length(x1_range), byrow = TRUE)

# Draw the decision boundary (where predicted class changes)

plot(PLADat$X2 ~ PLADat$X1, col = yHatSVM)

# contour(x1_range, x2_range, z, levels = c(1.5), add = TRUE, drawlabels = FALSE, lwd = 2)
```

```r
# Our solution
#plot(X2~X1, data = PLADat)

center <- model$x.scale$"scaled:center"
scale  <- model$x.scale$"scaled:scale"

sv_scaled <- model$SV

sv_unscaled <- sweep(sv_scaled, 2, scale, FUN = "*")
sv_unscaled <- sweep(sv_unscaled, 2, center, FUN = "+")


points(sv_unscaled[,2]~sv_unscaled[,1],pch = '+', col = 'blue',cex = 2)
```




```r
N = dim(X)[1]
plot(model$coefs~model$index,type = 'h',xlim = c(0,N))
```



From this, we can see that both algorithims perform similarly. The number of non zero coefficients is similar, although the direction is only different due to how the SVM function displays them. We can also see similar performance in how the custom and SVM function classifies data points and the position of the support vectors.