```
---
title: "Week 9 Homework"
author: "Rick Lentz""
date: "11/6/2017"
output: html_document
runtime: shiny
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

We'll define several functions for you and then you'll use the optimization tools introduced in the presentation to try to find (approximately) optimal solutions to the problems detailed below.   You should include your solutions right in this document and "knit" it into a Word document (just like you did for DS705 assignments). When you're done, submit both the Word document and this .rmd file to the D2L dropbox.

Problem 1 - Redistricting with a Genetic Algorithm

Solve problem 13.10-6. Install the 'gramEvol' package to get access to a genetic algorithm that uses integer encoding called GeneticAlg.int(). Use that algorithm to solve this problem. You'll have to read the documentation to figure out how to use the algorithm.

```{r echo=FALSE, eval = TRUE}

# don't modify this block.

# the distAssign and demrepFit functions go along with problem 13.10-6 from the textbook.
# demrepFit is the function to optimize

cities = matrix(c(152,81,75,34,62,38,48,74,98,66,83,86,72,28,112,45,93,72,
                  62,59,83,52,87,87,69,49,62,72,75,82,83,53,98,82,68,98),18,2);

distAssign <- function(x){
  # x is an input of 18 integers 1..10 that assign city 1..18 to districts 1..10
  # distAssign is a dataframe with the number of Democrats, Republicans, Total, and Winner
  # in each of the 10 districts
  sumdem = numeric(10);
  sumrep = numeric(10);
  sumtot = numeric(10);
  for (i in 1:10){
    sumdem[i] = sum( cities[x==i,1] );
    sumrep[i] = sum( cities[x==i,2] );
    sumtot[i] = sumdem[i] + sumrep[i];
  }
  distAssign <- data.frame( Dem = sumdem, Rep = sumrep, Tot = sumtot, Win = sumrep>sumdem );
}

demrepFit <- function(x){
  # x is an input of 18 integers 1..10 that assign city 1..18 to districts 1..10
  # demrepfit is the number of districts where Republicans have a majority
  # subtract a penalty proportional to amount by which constraints are violated
  #
  df <- distAssign(x);
  numRepDist = sum(df$Win);
  sumtot = df$Tot;

  # total number of voters is between 150 mil and 350 mil in each district
  # to enforce this constraint we subtract a penalty term equal to the total amount this
  # constraint is violated
  demrepFit = numRepDist - ( sum( 150-sumtot[sumtot<150] )  + sum( sumtot[sumtot>350]-350 ) );

  # if the algorithm you use minimizes the fitness instead of maximizing, then
  # uncomment the next line, if your routine maximizes then the next line should be
  # commented out
  demrepFit = -demrepFit;
}

# Notice the constraints aren't explicitly enforced, but instead a penalty term is included in the
# fitness function to encourage the genetic algorithm to seek potential solutions that satisfy
# the constraints.

library("parallel")
options(mc.cores = 8)

g_besty = 0
for(i in 1:50){
  x <- GeneticAlg.int(genomeLen = 18, codonMin = 1, codonMax = 10,
                  allowrepeat = TRUE, terminationCost = -210,
                  monitorFunc = NULL, evalFunc = demrepFit ,plapply = mclapply);

  best.result <- x$best$genome;



  # we have found a solution, inspect the results using
  besty = sum(distAssign(best.result )$Win,na.rm = TRUE)
  if (besty > g_besty)
  {
    print(x$best$genome);
```
```

```
    print(besty)
    g_besty = besty
    print(distAssign(best.result ));
  }
  print(i)
}


```

In the block below add your code to use the genetic algorithm. Either experiment with different random number seeds or use a for loop to conduct the optimization many times to find the best solution you can.

```{r}

# for this x Republicans win 8 districts 10  2  3  4  5  8  7  1  9  4  7  5  3  2  6  9  8  1
```

Make sure you print out your best solution. How many districts do Republicans win with your solution? (I don't think Republicans can win all 10 in this example, but they can get close.)

Problem 2 - TSP with a Genetic Algorithm

Use the ga() function with permutation encoding from the 'GA' package to approximate a solution to this 48 city TSP problem. Try different random number seeds and report the best result you can find. Copy the code from saTSP.R to make a graph of the tour. Your fitness function is tspFitness().

```{r echo = FALSE}
# this is supposedly the 48 captial cities in the continental United States,
# but I think things are a little off and I don't know the units.
# The data is available here
# http://people.sc.fsu.edu/~jburkardt/datasets/tsp/tsp.html

# load the data
D <- as.matrix(read.table("att48_d.txt"))
#print(D)
coord <- as.matrix(read.table("att48_xy.txt"))

# this is supposedly the best possible tour
tour_best <- as.vector(as.matrix(read.table("att48_s.txt")))
tour_best <- tour_best[1:48]
x <- coord[,1];
y <- coord[,2];
n <- length(x);
numcities <- n;

# given a tour, calculate the total distance
tourLength <- function(tour, distMatrix) {
  tour <- c(tour, tour[1])
  route <- embed(tour, 2)[, 2:1]
  sum(distMatrix[route])
}
# inverse of thetotal distance is the fitness (because the GA maximizes)
tspFitness <- function(tour, ...) 1/tourLength(tour, ...)
(tspFitness(tour_best,D))
print(tour_best)

#install.packages("GA");
library("GA");

results <- ga(type="permutation", fitness=tspFitness,D,min=1,max=48,maxiter=10000)
summary(results)
plot(results)
print(tour_best)


```
```{r}

## Combinatorial optimization: Traveling salesman problem
labs <- seq(1,48,1)

distance <- function(sq) {  # Target function
  sq2 <- embed(sq, 2)
  sum(D[cbind(sq2[,2], sq2[,1])])
}

print(distance(tour_best))
# rotate for conventional orientation
loc <- -cmdscale(D, add = TRUE)$points
x <- loc[,1]; y <- loc[,2]
s <- seq_len(47)
tspinit <- loc[tour_best,]

plot(x, y, type = "n", asp = 1, xlab = "", ylab = "",
     main = "initial solution of traveling salesman problem", axes = FALSE)
arrows(tspinit[s,1], tspinit[s,2], tspinit[s+1,1], tspinit[s+1,2],
       angle = 10, col = "green")
text(x, y, labels(labs), cex = 0.8)
```

```
## Problem 3 - TSP with Simulated Annealing

Modify the code in saTSP.R and include it below to approximate an optimal tour for the 48 city TSP problem in Problem 2. Include a graph of the best tour you are able to find.

```{r}
## Combinatorial optimization: Traveling salesman problem
library(stats) # normally loaded

labs <- seq(1,48,1)


distance <- function(sq) {  # Target function
  sq2 <- embed(sq, 2)
  sum(D[cbind(sq2[,2], sq2[,1])])
}

genseq <- function(sq) {  # Generate new candidate sequence
  idx <- seq(2, NROW(D)-1)
  changepoints <- sample(idx, size = 2, replace = FALSE)
  tmp <- sq[changepoints[1]]
  sq[changepoints[1]] <- sq[changepoints[2]]
  sq[changepoints[2]] <- tmp
  sq
}

sq <- c(1:nrow(D), 1)  # Initial sequence: alphabetic
distance(sq)
# rotate for conventional orientation
loc <- -cmdscale(D, add = TRUE)$points
x <- loc[,1]; y <- loc[,2]
s <- seq_len(nrow(D))
tspinit <- loc[sq,]

plot(x, y, type = "n", asp = 1, xlab = "", ylab = "",
     main = "initial solution of traveling salesman problem", axes = FALSE)
arrows(tspinit[s,1], tspinit[s,2], tspinit[s+1,1], tspinit[s+1,2],
       angle = 10, col = "green")
text(x, y, labels(labs), cex = 0.8)

set.seed(123) # chosen to get a good soln relatively quickly
res <- optim(sq, distance, genseq, method = "SANN",
             control = list(maxit = 10000000, temp = 2000, trace = TRUE,
                            REPORT = 500))
res  # Near optimum distance around 12842

tspres <- loc[res$par,]
plot(x, y, type = "n", asp = 1, xlab = "", ylab = "",
     main = "optim() 'solving' traveling salesman problem", axes = FALSE)
arrows(tspres[s,1], tspres[s,2], tspres[s+1,1], tspres[s+1,2],
       angle = 10, col = "red")
text(x, y, labels(labs), cex = 0.8)
```

How do the results compare to those achieved using a genetic algorithm? The SA approach requires a bit more tuning but as a function of linear time, seems to incrementaly progress towards the global optimal soltuion.

Is one algorithm significantly more efficient than the other? The GA solution seems more efficient.

Compare the total tour distances produced by the two algorithms, does one algorithm consistently produce better results? GA appears to consistently product better results.

Problem 4 - Comparing Algorithms for a 30 dimensional Rastrigin function

The 30 dimensional Rastrigin function is considered very difficult to optimize and is a test case for many optimization algorithms. We know that the global minimum value of 0 occurs at the origin. For this problem you should compare the performance of Naive Multistart, the Genetic Algorithm plus local search, and the Simulated Annealing algorithm GenSA() from the 'GenSA' package. If you can get it to work, then also try the mlsl() function in the 'nloptr' package as it should work considerably better than Naive Multistart.

This is a somewhat open problem, but at the very least you should try each algorithm multiple times (possibly in for loop) and report on which algorithms are most efficient (fewest function calls) and which are most reliable (able to consistently identify the global minimum). Experiment with the algorithm parameters (population size, number of iterations of local search, etc.) You'll likely have to increase population sizes and the maximum number of iterations to successfully solve the 30 dimensional problem. Look at the source code in the presentation .Rmd file included in the download packet for guidance in setting up your algorithms.

```{r echo = FALSE}
Rastrigin <- function(x) {
            (sum(x^2 - 10 * cos(2 * pi  * x)) + 10 * length(x))
}

# your code goes in this block
dimension = 30;
lower = rep(-5.12,dimension); upper = rep(5.12,dimension);
x0 = runif(dimension,-5.12,5.12)

#
# library("GA");
#
```
```

```
# result_ga <- ga(type="real-valued", fitness=Rastrigin,min=lower,max=upper,maxiter=10000)
# round(result_ga@solution,5)
# result_ga@fitnessValue
# summary(result_ga)
#
# require(GenSA)
# result_sa = GenSA(x0,Rastrigin,lower=lower,upper=upper,control=list(max.call=500000))
# result_sa$value
# result_sa$par


bestx = 1999
bestmin <- 1000000;
set.seed(1999);
for(j in 1:50000)
{
    x0<- as.vector(runif(2,min=-5.12, max=5.12));
    result_naive_multistart <- optim(x0,Rastrigin, method="L-BFGS-B", lower=c(-5.12,5,12),upper=c(5.12,5.12))
  if (result_naive_multistart$value<bestmin)
  {
    bestmin =result_naive_multistart$value;
    bestx = result_naive_multistart$par
  }
}
print(bestmin)
print(bestx)
```

Discuss your algorithm comparison here:
GA appears to get stuck easily.

SA works very well and seems well suited for finding the global minimum.

Naive Multistart using L-BFGS-B works well and appears to have the lowest resource requirements.

Unfortunately, nloptr seemed to buggy to install propertly