CSC 120 (R Section, L0201), Spring 2016 — Assignment #1

Worth 10% of the course grade. Due by the start of class on February 26 (instructions for how to hand it in will be posted on the course web page later). This assignment may be handed in late, with a 20% penalty, by start of class on March 1. Assignments will not usually be accepted after that. Contact the instructor as soon as possible if you have a legitimate excuse (eg, documented illness) for handing in the assignment late (without penalty).

This assignment is to be done by each student individually. You may discuss it in general terms with other students, but the work you hand in should be your own. In particular, you shouldn't leave a discussion with someone else with any written notes (either paper or electronic).

In this assignment, you will write an R program for pairing up all points in a set containing an even number of points, in a way that tries to minimize the sum of the distances between the points in each of the pairs. This problem can arise in many contexts, including design of experiments in statistics. Doing this will provide practice in basic R programming facilities such as `if` and `for`, and use of vectors and matrices, in use of R's plotting facilities, and in how to divide a program into several functions that can be understood and sometimes used separately.

The input for this problem is a set of n points in k-dimensional Euclidean space. The number of points, n, will always be even. For the actual data you will work with, k is two, so the input is a set of points in the plane, but except for the 2-dimensional plotting function described below, your program should work for any dimensionality, k, and for any even number of points, n.

The output of the program is a set of pairs of input points, with each of the input points being in exactly one pair. The order of the pairs and the order of points in each pair are not important. The objective is to try to find a set of pairs for which the total of the distances from one point in a pair to the other is small. Ideally, this total distance would be as small as possible, but the method used in this assignment is not guaranteed to find a set of pairings that achieve this minimum, it just tries to make the total distance as small as it can.

In designing this program, we must decide how to represent the input (a set of points) as an R data structure. For this assignment, we will use an n by k matrix, in which each row represents a point in k-dimensional Euclidean space. We must also decide how to represent the set of pairs of points that is the output. We will use an $n/2$ by 2 matrix for this, in which each row contains two integers between 1 and n, which index the rows for the two points in the input matrix.

We also must specify how a function to find the pairings will work. For this assignment, you should write a function called `find_pairings` that can be called with a single argument that is a matrix of points, as specified above, and which will return as its value a matrix giving the pairs of points that we hope have a small total distance. (Below, some additional optional arguments to `find_pairings` will be specified.)

For this assignment, you should test your program on a set of 60 points in 2-dimensional Euclidean space that you can read as a matrix as follows (storing it in a variable called `data`):

```
data <- as.matrix(read.table("http://www.cs.utoronto.ca/~radford/csc120/a1data.txt"))
```

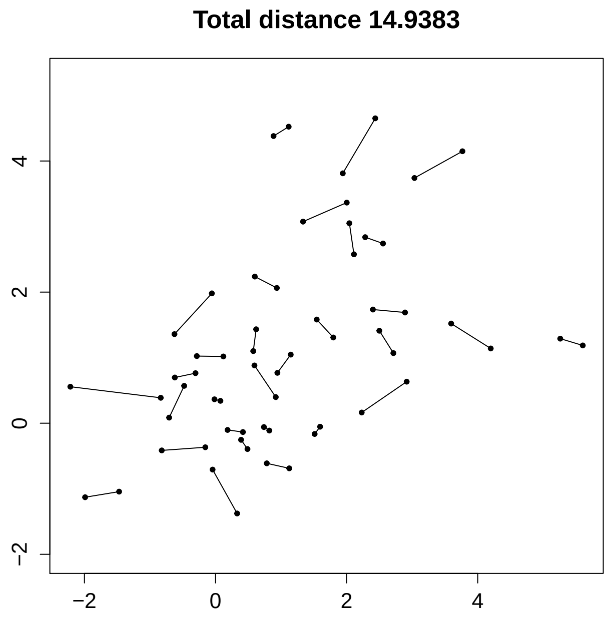
You should then be able to obtain a set of pairs with the call `find_pairings(data)`.

It's not obvious how to find a set of pairings for which the total distance between points in the pairs is small. Looking at every possible set of pairings is infeasible unless the number of points, n, is quite small, since the number of possible sets of pairings is $n! \,/\, ((n/2)!\, 2^{n/2})$, which is a formula you can try deriving yourself for the fun of it (not for credit). For $n = 30$, this is about 6×10^{15}, so many that it would take centuries to look at each of them, even with today's fast computers, and for $n = 60$, this is about 3×10^{40}, so huge that it would take many times the age of the universe to look at them all.

The approach we will take for this assignment is to start with a random set of pairings, and then try to improve this set of pairings, until we can't make any more improvements. We can then try starting with another random initial set of pairings, and improving it. After trying some number of random initial sets of pairings, we use the best that we found as our final answer. This answer isn't guaranteed to have the smallest total distance for all pairs, but we will hope it's pretty good.

The way we will try to improve a set of pairings is to consider all combinations of two pairs — say, (a, b) and (x, y) — and for each such combination consider changing to one of the other two possible pairings involving these four points — either (a, x) and (b, y) or (a, y) and (b, x). We change to whichever of these pairings have the smallest total distance, or stay with the original pairings if they have the smallest total distance.

Here is the result of applying this method (trying 10 random initial pairings) to the data obtained as shown above:



In this plot, the 60 points are shown as black dots, and the 30 pairings are shown by lines connecting the paired points. Note that this result is sensitive to the random initial pairings that were obtained, and so may not be the same as the results you get (though your result should be at least somewhat similar).

To make your program easier to write and understand, you should write several functions in addition to `find_pairings`, as described below. You will use these functions when writing `find_pairings`.

You should write a function called `distance` that takes two vectors (of equal length) as arguments, and returns the Euclidean distance between these vectors, which is the square root of the sum of the squares of the differences in elements of the two vectors. In doing this, you may find it helpful to use the R `sum` function, which computes the sum of all elements of a vector.

You should write a function called `total_distance` that takes as arguments a matrix of points (like the argument of `find_pairings`) and a matrix of pairings (like the value of `find_pairings`), and returns as its value the total distance between the points in all the pairs.

You should write a function called `plot_pairs` that takes as arguments a matrix of points and a matrix of pairings, and produces a plot like the one shown above, in which the points are shown as dots and the pairs as lines connecting these dots. (However, this function should not add a title such as above. A title can be added later with R's `title` function.)

You should write a function `random_pairings` that takes as its argument a number of points, n, which must be even, and returns a matrix with $n/2$ randomly chosen pairings of these n points (of the same form as would be returned as the value of `find_pairings`). To do this, you can use R's `sample` function, as discussed in the lecture slides.

You should write a function `improve_pairings` that takes as its arguments a matrix of points and a matrix of pairings, and returns as its value a matrix of pairings that may be improved. As described above, this should be done by considering in turn all combinations of two pairs, and modifying the way the four points involved in these two pairs are paired, if this reduces the total distance. To do this, you might write two `for` loops, one inside the other, with the outer loop looking at every pair except the last, and the inner loop looking at every pair after the pair currently being looked at by the outer loop.

Finally, you should write your `find_pairings` function, which should take a matrix of points as its first argument, and an optional second argument, called `tries`, which specifies how many random initial pairings to try (with the default value being 1). It should also take an optional third argument, called `plot`, which defaults to FALSE, but which if TRUE results in every set of pairings produced during the operation of `find_pairings` being plotted, with a suitable title that allows it to be identified (which you can put together with `paste`). Such plots may be useful when debugging your functions. You should also print (with `cat`) the total distance for the final improved set of pairings found from each random initial set of pairings. The value of `find_pairings` should be a matrix of pairs, as described above.

All your functions definitions should be properly indented, and otherwise written to be easily readable, with appropriate comments describing the arguments and values of the functions, and what is being done in the program at places where this is not obvious.

For all your functions, you do not need to check whether the arguments are of the expected form (eg, whether the number of points, n, is even). This would be a good idea, but you may omit checking this for this assignment, since we haven't covered some R facilities that would help with doing it.

You should put the definitions of all these functions in an R script file, which should contain nothing except these function definitions. These function definitions should *not* refer to the specific data file that you will be using.

In a separate R script file, you should read in these function definitions with `source`, read in the data file as described above, and then try out the `find_pairings` function on this data. You should run `find_pairings` trying 10 random initial set of pairings, with the random number seed set with `set.seed(1)`. You should produce a plot similar to the one above showing the final set of pairings found. You should also report the total distance after improvement starting from all ten random initial pairings. You should also run `find_pairings` trying just one random initial set of pairings, after setting the random seed to your student ID number, and hand in a plot showing those pairings.

How exactly to hand in your assignment will be specified on the course web page once I've decided. You will need to hand in the R script file with function definitions, the R script file that applies these functions to the data read from the course web page, the two plots that are produced by this script, and the text output showing the total distances found with the ten random initial sets of pairings.

Here is some background, which is not essential for doing this assignment.

This problem is known as "Euclidean Minimum Weight Matching", and has been studied for over 50 years. In 1965, Jack Edmunds published a fairly fast way of finding a set of pairings (also known as a matching) that achieves the smallest possible total distance (also known as weight) between points in the pairs. His method is much faster than looking at every possible set of pairings. Faster algorithms have been developed since then. The method used in this assignment is much less sophisticated than these other methods.

One can think of many contexts where one might want to solve this problem. For instance, a same-sex dating service might want to pair up their clients in a way that minimizes the average amount by which the interests of people in a pair differ (according to the numerical descriptions of their interests that clients provided).

In statistics, the variability in the result of a randomized experiment, such as a clinical trial of a drug, can be reduced if subjects in the "treatment" and "control" groups are paired according to characteristics that are expected to influence the result. For example, one might sort subjects by age, and then take consecutive subjects as pairs, which will result in most pairs being close in age. Then for each pair, one would randomly assign one subject in the pair to the control group, and one to the treatment group. This eliminates the possibility that by chance a completely random assignment might, for example, put most of the old people in the control group. It's not so obvious how to pair subjects on more than one characteristic (eg, age, weight, and blood pressure), but Euclidean Minimum Weight Matching is one possibility.