Advanced R for Genetic Analysis

Exercises for Session 8: Interfacing to C.

1. The files nnfind.c, bin\_heap.c, bin\_heap.h, item.h implement an algorithm based on k-d trees for finding nearest neighbours. Write an R interface to the functions in nnfind.c:  
     
   void within\_neighbours(const double \*X, int \*pNx, const int \*pp,   
    int \*neighbours, double \*dists)  
   and  
     
   void between\_neighbours(const double \*X, int \*pNx, const double \*Y,  
    const int \*pNy, const int \*pp, int \*neighbours, double \*dists)  
     
   Notes: In these functions X and Y are matrices of points in p-dimensional space, \*pNx is the number of rows in X, \*pNy is the number of rows in Y, \*pp is the dimension of the space (number of columns in X and Y), neighbours is used to return the row number of the nearest neighbour and dists returns the distance to the nearest neighbour.  
     
   The difference between the two functions is that within\_neighbours finds the nearest neighbour in X of each point in X and between\_neighbours finds the nearest neighbour in X of each point in Y. This means that neighbours and dists have length \*pNx in within\_neighbours and \*pNy in between\_neighbours.
2. A `box-car’ filter is a simple smoother; on a scatterplot of (X1, Y1), (X2, Y2), … (Yn, Xn), it provides a smooth line illustrating how Y changes with X. Formally, for given radius *r*, at point *x* it is evaluated as;  
     
    ,  
     
   in other words, it is the average of the Y’s that have X’s within *r* of *x*. Typically, we evaluate the box-car filter at x=X1, X2, … Xn.  
     
   (continues…)

In R, one simple way to implement the box-car filter is the following;

boxcar <- function(Y, X, radius, n=length(Y)){

y.smooth <- rep(0,n)

x <- 0

for (i in 1:n){

count <- 0

x <- X[i]

for (j in 1:n){

if(abs(X[j]-x)<radius){

count <- count+1

y.smooth[i] <- y.smooth[i] + Y[j]

}

}

y.smooth[i] <- y.smooth[i]/count

}

y.smooth

}

Try this code, for n=1000 data points, and then n=10,000. What takes the time? Code this approach in C, and see how much faster it becomes.

For keen people; a preliminary sort of the data enables you to implement this filter without the double loop; think of ‘sliding’ a window of radius r along the sorted X values. Implement the filter using this observation, and see what speed improvement you can achieve.