**The HATS-PR Algorithm**

In Part 1 I briefly mentioned that we would be using the HATS-PR algorithm of Robinette *et al.* (Robinette et al. 2011). I also discussed the choice of objective function which is used to report on the quality of the alignment. HATS-PR stands for “Hierachical Alignment of Two-Dimensional Spectra - Pattern Recognition”. In ChemoSpec2D the algorithm is implemented in the hats\_alignSpectra2D function. Here are the major steps of the HATS-PR algorithm:

1. Contruct a *guide tree* using hierarchical clustering (HCA): compute the distance between the spectra, and use these distances to construct a dendrogram (the guide tree). As the name suggests, this tree is used to guide the alignment. The most similar spectra are aligned first, then the next most similar, and so on. In later rounds one applies the alignment procedure to sets of spectra that have already been aligned. In Robinette *et al.* they use the Pearson correlation coefficient as the distance measure. In ChemoSpec2D you can choose from a number of distance measures. I encourage you to experiment with the choices and see how they affect the alignment process for your data sets.
2. For each alignment event, check the alignment using the objective function, which recall is a distance measure. If the objective function is below the threshold, no alignment is needed (“below” assumes we are minimizing the objective function, but we might also be maximizing and hence trying to exceed the threshold). If alignment is necessary, move one of the spectra relative to the other *in some fashion*, checking each new postion with the objective function until the best alignment is found. This is an exercise in optimization.

**Finding the Optimal Alignment**

The heart of the task is in the phrase *in some fashion*. At one extreme, one can imagine holding one spectrum fixed, and sliding the other spectrum left and right, up and down, over some range of values – essentially a grid of data points. At each position on the virtual grid, evaluate the objective function and keep track of the results. This will always find the answer, but such a brute force search will be very time-consuming and undesirable, especially if the search space is large. Alternatively, do something more efficient! Robinette *et al.* use a “simple gradient ascent” approach, but there is a vast literature on optimization strategies that we can consider. In ChemoSpec2D we use a machine learning approach (details next), but the function is written in such a way that one can add other optimization approaches seamlessly. *Anything* is better than a brute force approach.

**Optimizing with mlrMBO**

The name mlrMBO comes from “machine learning with R model-based optimization.” mlrMBO is a powerful and flexible package for general purpose optimization, especially in the cases where the objective function is computationally expensive. There is a nice [introductory vignette](https://cran.r-project.org/web/packages/mlrMBO/vignettes/mlrMBO.html).

The basic steps in the model-based optimization using mlrMBO as implemented in hats\_alignSpectra2D in package ChemoSpec2D are as follows:

1. Define your objective function. Our choice of the Euclidean distance was described in Part 1, along with other options. Most distance measures are not computationally expensive in terms of code. However, the huge number of data points in a typical 2D NMR spectrum bogs things down considerably. The approach taken in model-based optimation mitigates this to a great deal, since the objective function is only used for the initial response surface.
2. Generate an “initial design”, by which we mean a strategy to search the possible optimization space. hats\_alignSpectra2D takes arguments maxF1 and maxF2 which define the space that will be considered as the two spectra are shifted relative to each other. The space potentially covered is -maxF1 to maxF1 and similarly for the F2 dimension. We take advantage of concepts from the design of experiments field, and use the [lhs package](https://cran.r-project.org/web/packages/lhs/vignettes/lhs_basics.html) to generate a Latin Hypercube Sample of our space.
3. The sample points selected by lhs are evaluated using the objective function.
4. The values of the objective function at the sample points are used to create a surrogate model, essentially a response surface. The key here is that the surrogate model is computationally fast and will stand in for the actual objective function during the optimization. mlrMBO provides many [options](https://mlr.mlr-org.com/articles/tutorial/integrated_learners.html) for the surrogate model. For hats\_alignSpectra2D we use a response surface based on kriging, which is a means of interpolating values that was originally developed in the geospatial statistics world.
5. New samples points are suggested by the kriging algorithm, evaluated using the surrogate function, and used to update (improve) the model. Each iteration improves the quality of the model.
6. After reaching the designated threshold or the number of iterations specified, the best answer is returned. In this case the best answer is the optimal shift of one spectrum relative to the other, in each dimension.

**Other Details**

In addition to the differences noted above, the implementation of HATS-PR in ChemoSpec2D carries out only *global* alignment. The algorithm described by Robinette *et al.* includes *local* alignment steps which I have not implemented. Local alignment is a possible future addition.

**Configure Your Workspace**

If you are going to actually execute the code here (as opposed to just reading along), you’ll need the development version of ChemoSpec2D (I improved some of the plots that track the alignment progress since the last CRAN release). And you’ll need certain packages. Here are the steps to install everything:

chooseCRANmirror() # choose a CRAN mirror

install.packages("remotes")

library("remotes")

# devel branch -- you need 0.4.156 or higher

library("ChemoSpec2D")

# other packages needed

install.packages("mlrMBO") # will also install mlr, smoof, ParamHelpers

install.packages("lhs")