Rearrangement in R: A Vignette

Wesley Graybill Mingli Chen Victor Chernozhukov
Ivan Fernandez-Val Alfred Galichon

September 16, 2024

Abstract

Suppose that a target function $f_0: \mathbb{R}^d \to \mathbb{R}$ is monotonic, namely weakly increasing, and suppose that an initial, not necessarily monotonic, estimate $\hat{f}$ is available. The rearrangement methods of [3] can be used to transform $\hat{f}$ into a monotonic estimate $\hat{f}^*$. As was shown in [1], this rearranged estimate $\hat{f}^*$ is necessarily closer to the target function f_0 in standard metrics. The R package `Rearrangement` implements this method for both point and interval estimates. This vignette serves as an introduction to this package and displays basic functionality of the functions contained within.

1 Getting Started

To get started using the package `Rearrangement` for the first time, first issue the command

```
> install.packages("Rearrangement")
```

into your R browser to install the package to your computer. Once the package has been installed, you can then use the package `Rearrangement` during any R session by simply issuing the command

```
> library(Rearrangement)
```

Now you are ready to use the functions and data sets contained in `Rearrangement`. For general questions about the package you may type

```
> help(package="Rearrangement")
```

to view the package help file, or for more questions about a specific function you can type `help(`*function-name*`)`.For example, try:

```
> help(rearrangement)
```

2 Functions

2.1 Rearrangement of Point Estimates: `rearrangement`

First consider the univariate case. Let χ be a compact interval and $\hat{f}: \chi \to K$ be a measurable function where K is a bounded subset of $\mathbb{R}$. Without loss of generality we can take χ to be $[0,1]$. Let $F_{\hat{f}}(y) = \int_{\chi} 1\left\{\hat{f}(u) \leq y\right\} du$ denote the distribution function of $\hat{f}(X)$ where X follows the uniform distribution on χ. We define the rearrangement operator on $\hat{f}$ as follows:

$$\hat{f}^*(x) := inf\left\{y \in \mathbb{R} \left| \left[\int_{\chi} 1\left\{\hat{f}(u) \leq y\right\} du\right] \geq x\right.\right\}.$$

This function $\hat{f}^*$ is the increasing rearrangement of $\hat{f}$. Note that $\hat{f}^*$ is simply the quantile function of $\hat{f}$.

In fact, the monotonic rearranged function $\hat{f}^*$ is provably closer to the monotonic target function f in the L^p norm than the original non-monotonic estimate $\hat{f}$.

Let us show the properties by using `rearrangement` to reproduce the results of [1]. To do this we must use `GrowthChart`, a data set containing the height and ages of U.S.-born white males age 2-20.

```
> data(GrowthChart)
> attach(GrowthChart)
```

Firstly we reproduce the univariate spline regression of age versus height. To do so we must first load the splines package:

```
> library(splines)
```

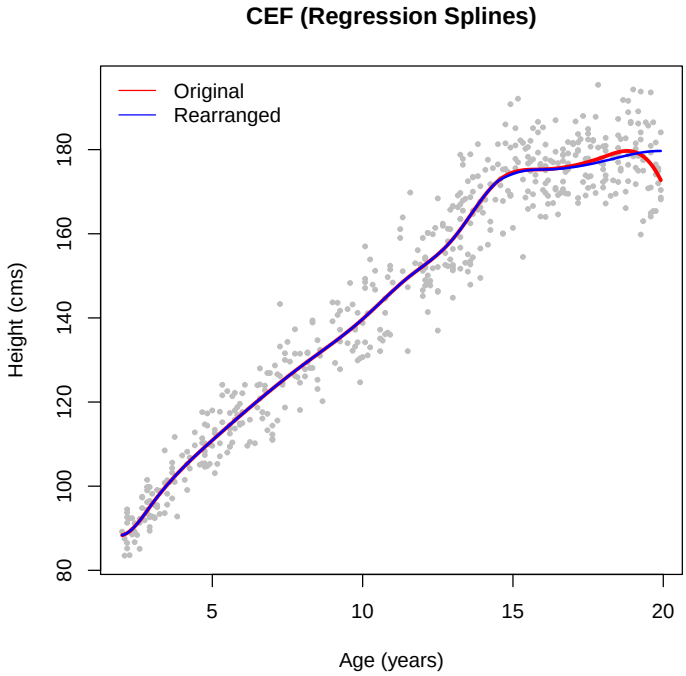
Next we will perform the spline regression:

```
> ages <- unique(sort(age))
> aknots <- c(3, 5, 8, 10, 11.5, 13, 14.5, 16, 18)
> splines_age <- bs(age,kn=aknots)
> sformula <- height~splines_age
> sfunc <- approxfun(age,lm(sformula)$fitted.values)
> splreg <- sfunc(ages)
```

Finally, we perform the rearrangement and plot the results:

```
> rsplreg <- rearrangement(list(ages),splreg)
```

```
> plot(age,height,pch=21,bg='gray',cex=.5,xlab="Age (years)",ylab="Height (cms)",main="CEF
> lines(ages,splreg,col='red',lwd=3)
> lines(ages,rsplreg,col='blue',lwd=2)
> legend("topleft",c('Original','Rearranged'),lty=1,col=c('red','blue'),bty='n')
```



Notice that the original estimate had trouble accounting for the slow down in growth around age 18. The rearranged estimate corrects for this, providing a function that is monotonic over all ages.

This improvement in the estimation properties of $\hat{f}$ can be extended to the multivariate case. Suppose $\hat{f}$ is a function in d variables. Let $\hat{f}^*$ be the result of applying the rearrangement operator to each of the d variables in some ordering. That is, we rearrange with respect to each of the d variables in some order corresponding to a permutation of the integers $\{1, \ldots, d\}$ (For more details please refer to [1]). Then $\hat{f}^*$ is provably closer to the target function f than the original estimate $\hat{f}$. Note that this property also holds for the average rearrangement over all possible orderings.

Now let us observe some of the functionality of this within R . The default call to `rearrangement` is

```
> rearrangement(x, y, n = 1000, stochastic = FALSE, avg = TRUE, order = 1:length(x))
```

In the function `rearrangement` the initial estimate of the function $\hat{f}$ is represented as a collection of values of the function at various points. That is, suppose f is a function of d variables $(x_1, x_2, \ldots, x_d)$ and we sample $\hat{f}$ at all possible combinations of the points $\{x_{1,1}, x_{1,2}, \ldots, x_{1,n_1}\}, \{x_{2,1}, x_{2,2}, \ldots, x_{2,n_2}\}, \ldots, \{x_{d,1}, x_{d,2}, \ldots, x_{d,n_d}\}$. That is, we have a collection of values $y_{i_1,\ldots,i_d} = \hat{f}(x_{1,i_1}, \ldots, x_{d,i_d})$ where i_j runs over all possible values. In the function call to `rearrangement`, `x` is a list, or

data.frame, the entries of which contain the $x_j's$, and `y` is a vector, matrix, or array containing the values of $\hat{f}$ at these points

For clarification, let's consider an example. Suppose $\hat{f}(x_1, x_2) = (x_1-1)(x_1-2)(x_1-3)(x_2-1)(x_2-2)(x_2-3)$, or in R :

```
> f <- function(x1,x2){(x1 - 1)*(x1 - 2)*(x1 - 3)*(x2 - 1)*(x2 - 2)*(x2 - 3)}
```

Let the set of x_1's be $\{0, .5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5\}$ and the set of x_2's be the same. In R we can do this with:

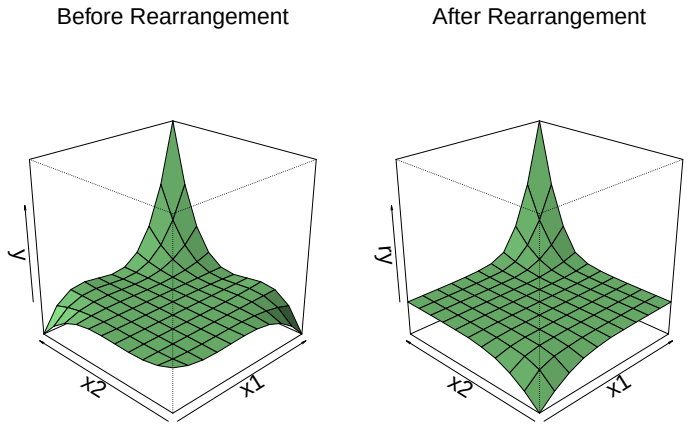
```
> x1 <- seq(0, 5, by = .5)
> x2 <- x1
> l <- NULL
> for (i in x2){l <- c(l, f(x1, i))}
> y <- matrix(l, nrow = length(x1))
> x <- list(x1,x2)
```

Using `rearrangement` on this function in R :

```
> ry <- rearrangement(x,y)
```

The plot of this function and its rearrangement are:

```
> par(mfrow=c(1,2))
> persp(x1,x2,y,col='lightgreen',theta=315,phi=25,r=6,shade=.5,font.lab=15,cex.lab=3)
> title("Before Rearrangement",cex.main=3,font.main=30)
> persp(x1,x2,ry,col='lightgreen',theta=315,phi=25,r=6,shade=.5,font.lab=15,cex.lab=3)
> title("After Rearrangement",cex.main=3,font.main=30)
```



There are several different options for the basic function `rearrangement`. Perhaps the most important parameters for a multivariate rearrangement are *avg* and *order*. If you know the order in which you would like the rearrangement to be performed, you may input this order as a list of integers into the *order* option. Note that the input to *order* must be some permutation of the dimensions of the data. That is, suppose you are performing a d dimensional rearrangement. Then *order* should be the desired permutation of the integers $1, \ldots, d$. Note that the function `rearrangement` is currently not generalized to n dimensions, but instead only works for up to a trivariate rearrangement. Another option, which is advisable to use if an optimal order is not known, is *avg*. *avg* takes a logical value which is `TRUE` by default. If `avg = TRUE`, `rearrangement` averages the rearrangements corresponding to all possible permutations and returns this averaged rearrangement. For example, if $d = 2$, `rearrangement` computes the result with `order = c(1, 2)` and `order = c(2, 1)` and averages the two results. Yet another option available to users is *stochastic*. This enables stochastic sampling of the original estimate $\hat{f}$ as described in [2]. Unless you know stochastic sampling to be preferred, it is advised that this option be set to `FALSE`.

2.2 Rearrangement of Interval Estimates: `simconboot` and `rconint`

In the same way that rearrangement of a function improves its estimation properties, rearrangement of a confidence interval improves its inferential properties, decreasing the length of the interval while increasing its coverage probability.

Suppose we have an initial simultaneous confidence interval

$$[l, u] = \left\{[l(\vec{x}), u(\vec{x})], \vec{x} \in \chi^d\right\}$$

where l and u are the upper and lower end-point functions such that $l(\vec{x}) \leq u(\vec{x}) \quad \forall \quad \vec{x} \in \chi^d$.

If we apply the rearrangement operator to both end-point functions, the resulting monotonic confidence interval

$$[l^*, u^*] = \left\{[l^*(\vec{x}), u^*(\vec{x})], \vec{x} \in \chi^d\right\}$$

is necessarily an improvement on the original not necessarily monotonic estimate. Here l^* and u^* denote the increasing rearrangements of l and u respectively.

Within the package `Rearrangement` the function `simconboot` is used to construct simultaneous confidence intervals using a bootstrap method, while `rconint` is used to rearrange these confidence intervals. Let's look at an example from [1]. Again we will use the data from `GrowthChart`.

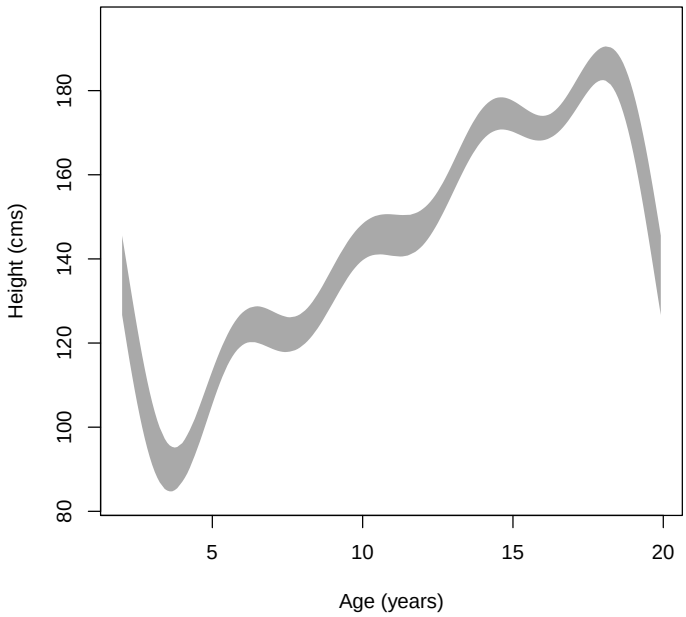
```
> data(GrowthChart)
> attach(GrowthChart)
```

First let's construct an estimate of the confidence interval. For the sake of argument, suppose we wish to use a Fourier series to model the data. Here we will use a Fourier model with 8 terms, 4 sine and 4 cosine.

```
> ##Normalize the ages to the interval [0, 2*pi]
> nage <- 2 * pi * (age - min(age)) / (max(age) - min(age))
> formula <- height ~ I(sin(nage)) + I(cos(nage)) + I(sin(2*nage)) + I(cos(2*nage)) + I(sin
> j <- simconboot(nage, height, lm, formula)
> class(j)

[1] "conint" "list"

> names(j)

[1] "x"       "y"       "sortedx" "Lower"   "Upper"   "cef"
```

Note that the output of `simconboot`, as well as that of `rconint`, is an object of class `conint` containing the elements `x, y, sortedx, Lower, Upper and cef`. Here `x` contains the original x data from the call to simconboot, `y` contains the original y data, and `sortedx` contains the x data sorted with repeated elements removed. `Lower` and `Upper` are the lower and upper end-point functions represented as a vectors containing the values of the functions evaluated at the points in `sortedx`. `cef` contains the original estimate.

```
> plot(j, border=NA, col='darkgray', xlab='Age (years)', ylab='Height (cms)', xaxt = "n")
> axis(1, at = seq(-2*pi*min(age)/(max(age)-min(age)), 2*pi+1, by=5*2*pi/(max(age)-min(age))
```



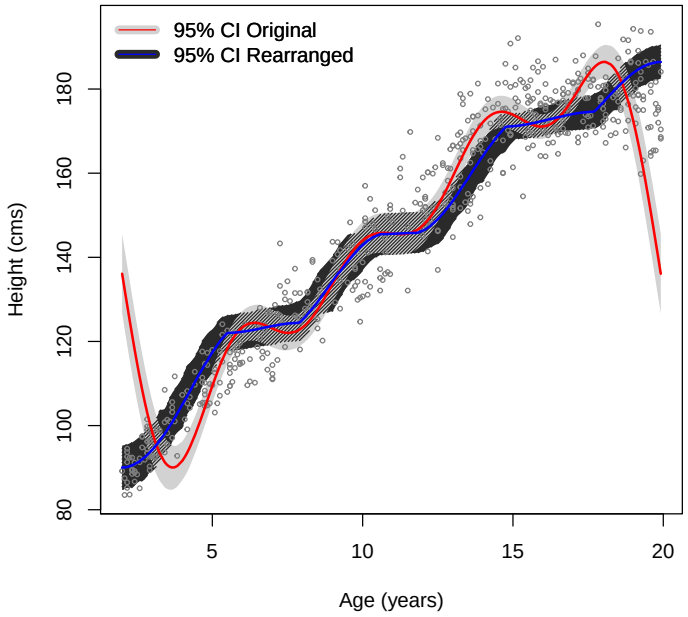
Notice in the plot above that our Fourier series estimate of the confidence interval is clearly non-monotonic. While a Fourier model may still have desirable approximation properties for our given data set, it is clear that the results do not conform to the inherent monotonic property we expect the data to follow. In general, we may have a particular model or estimator for a set of data that has desirable properties but may produce a non-monotonic estimate. In this case applying the rearrangement operator to the estimate will necessarily improve upon the original estimate while preserving the properties that made the original estimator attractive.

Let's observe what happens when we apply this to our growth chart example. First let's rearrange the confidence interval and then plot the results.

```
> k <- rconint(j)
```

```
> ages <- unique(sort(age))
> ffunc <- approxfun(age,lm(formula)$fitted.values)
> freg <- ffunc(ages)
> rfreg <- rearrangement(list(ages),freg)
> plot(k, border=NA, col='#2A2A2A', xlab='Age (years)', ylab='Height (cms)', xaxt = "n")
> axis(1, at = seq(-2*pi*min(age)/(max(age)-min(age)), 2*pi+1, by=5*2*pi/(max(age)-min(age))
> polygon.conint(j, border=NA, col='#D2D2D2')
> polygon.conint(k, border=NA, col='#2A2A2A', density=50)
> points(nage,height, cex = .5, col = '#7E7E7E')
```

```
> nages <- unique(sort(nage))
> lines(nages,freg, col='red',lwd=2)
> lines(nages,rfreg, col='blue',lwd=2)
> legend("topleft",c('95% CI Original','95% CI Rearranged'),lty=1,lwd=7,col=c('#D2D2D2','#2
> legend("topleft",c('95% CI Original','95% CI Rearranged'),lty=1,col=c('red','blue'),bty='
```



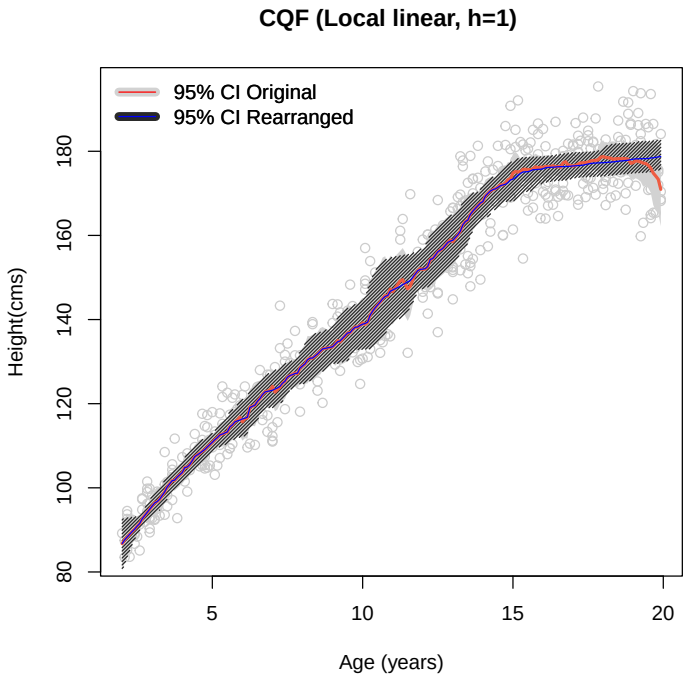
Notice that, as predicted, the rearranged confidence interval seems to be a better fit to the data than the initial one.

Suppose we also wish to use this example to show the properties after rearrangement of a local nonparametric approximation to the conditional median function. We compute point and interval local linear quantile regression estimates by calling the fuction `simboot` with the estimator `lprq2`.

```
> library(quantreg)
> ages <- unique(sort(age))
> ## Univariate
> j2 <- simconboot(age, height, lprq2, formula=0,B=20,h=1,xx=ages,tau=0.5)
> k2 <- rconint(j2)
> rcqf50 <- rearrangement(data.frame(j2$sortedx),j2$cef)
```

Note that the input `xx` of `simconboot` includes the points to evaluate the local linear quantile regression fit. Here xx contains the x data sorted with repeated elements removed.

```
> plot(age,height,xlab="Age (years)",ylab = "Height(cms)",col='grey80')
> polygon.conint(j2, border=NA, col='#D2D2D2')
> polygon.conint(k2, border=NA, col='#2A2A2A', density=50)
> lines(j2$sortedx,j2$cef,lty=1,lwd=2.5,col="tomato2")
> lines(j2$sortedx,rcqf50,lty=1,lwd=1,col="blue")
> legend("topleft",c('95% CI Original','95% CI Rearranged'),lty=1,lwd=7,col=c('#D2D2D2','#2A
> legend("topleft",c('95% CI Original','95% CI Rearranged'),lty=1,col=c('red','blue'),bty='n
> title(main="CQF (Local linear, h=1) ")
```



3 A Numerical Example: Monte-Carlo Experiment

The following experiment quantifies the improvement in point estimatation. We will replicate a result from [1]. As you will see, rearrangement of an original estimate decreases the L^p errors.

Let $Z(X)$ be a vector of a piecewise linear transformation of X defined as follows:

$$Z(X) = (1, X, 1\{X > 5\} * (X - 5), 1\{X > 10\} * (X - 10), 1\{X > 15\} * (X - 15))$$

Consider an experiment design where the dependent variable Y equals our func-

tion plus a disturbance independent of our regressor

$$Y = Z(X)'\beta + \epsilon$$

This design implies the conditional expectation function

$$E[Y|X] = Z(X)'\beta$$

We choose parameters for the experiment to match those of the original growth chart data. For the error ϵ, we draw samples from a normal distribution with mean and variance equal to that of the residuals from our regression $\epsilon = Y - Z(X)'\beta$. The following table reflects the mean L^p estimation errors of the Splines, Fourier, Local Polynomial, and Kernel estimates of the conditional expectation function after 20 replications. Errors are expressed as ratios of the errors of the rearranged estimates with those of the original estimates.

```
> formula0        <- height ~ age + I((age >= 5)*(age - 5)) + I((age >= 10)*(age - 10))  + I
> cef                <- approxfun(age, lm(formula0)$fitted.values);
> ages                <- unique(sort(age));
> mheight         <- cef(ages);
> ###### Compute Locally Linear Mean Regression ##############################################
> # (Modification of Koenker's lprq with box kernel)
>
> # Residuals
> residuals0        <- lm(formula0)$residuals;
> set.seed(8);
> n                <- length(height);         # sample size
> B                <- 20;                 # Number of replications
> h                <- .5
> formula1         <- heightb ~ as.factor(age);
> nknots                <- 9;
> knots_age        <- c(3.0, 5.0, 8.0, 10, 11.5, 13, 14.5, 16, 18);
> splines_age     <- bs(age, degree = 3, intercept = FALSE, knots = knots_age);
> formula2        <- heightb ~ splines_age;
> nage                <- 2 * pi * (age - min(age)) / (max(age) - min(age));
> formula5        <- heightb ~ nage + I(nage^2) + I(sin(nage)) + I(cos(nage)) + I(sin(2*nage
> mheight0        <- NULL;
> mheight2        <- NULL;
> rmheight2        <- NULL;
> mheight5        <- NULL;
> rmheight5        <- NULL;
> mheight7        <- NULL;
> rmheight7        <- NULL;
> mheight8        <- NULL;
> rmheight8        <- NULL;
> for (s in 1:B) {;
+
```

```
+ res         <- rnorm(n, mean = mean(residuals0), sd = sd(residuals0));
+ heightb <- cef(age) + res;
+
+ mheight0        <- rbind(mheight0, mheight);
+
+
+ cef2                <- approxfun(age, lm(formula2)$fitted.values);
+         mheight2        <- rbind(mheight2, cef2(ages));
+ rmheight2        <- rbind(rmheight2, rearrangement(list(ages), mheight2[s, ]));
+
+ cef5                <- approxfun(age, lm(formula5)$fitted.values);
+         mheight5        <- rbind(mheight5, cef5(ages));
+ rmheight5        <- rbind(rmheight5, rearrangement(list(ages), mheight5[s, ]));
+
+ cef7                <- lplm(age, heightb, h, ages)$fitted.values;
+         mheight7        <- rbind(mheight7, cef7);
+ rmheight7        <- rbind(rmheight7, rearrangement(list(ages), mheight7[s, ]));
+
+         cef8                <- lclm(age, heightb, h, ages)$fitted.values;
+         mheight8        <- rbind(mheight8, cef8);
+         rmheight8        <- rbind(rmheight8, rearrangement( list(ages), mheight8[s, ]));
+
+ }
> table <- matrix(0, nrow = 3, ncol = 12, dimnames = list(c("L1", "L2", "Linf"), c("Splines"
> table[1,1] <- mean(apply(abs(mheight2 - mheight0), 1, mean));
> table[1,2] <- mean(apply(abs(rmheight2 - mheight0), 1, mean));
> table[1,3] <- table[1,2]/table[1,1];
> table[1,4] <- mean(apply(abs(mheight5 - mheight0), 1, mean));
> table[1,5] <- mean(apply(abs(rmheight5 - mheight0), 1, mean));
> table[1,6] <- table[1,5]/table[1,4];
> table[1,7] <- mean(apply(abs(mheight7 - mheight0), 1, mean));
> table[1,8] <- mean(apply(abs(rmheight7 - mheight0), 1, mean));
> table[1,9] <- table[1,8]/table[1,7];
> table[1,10] <- mean(apply(abs(mheight8 - mheight0), 1, mean));
> table[1,11] <- mean(apply(abs(rmheight8 - mheight0), 1, mean));
> table[1,12] <- table[1,11]/table[1,10];
> table[2,1] <- mean(apply(abs(mheight2 - mheight0)^2, 1, mean)^(1/2));
> table[2,2] <- mean(apply(abs(rmheight2 - mheight0)^2, 1, mean)^(1/2));
> table[2,3] <- table[2,2]/table[2,1];
> table[2,4] <- mean(apply(abs(mheight5 - mheight0)^2, 1, mean)^(1/2));
> table[2,5] <- mean(apply(abs(rmheight5 - mheight0)^2, 1, mean)^(1/2));
> table[2,6] <- table[2,5]/table[2,4];
> table[2,7] <- mean(apply(abs(mheight7 - mheight0)^2, 1, mean)^(1/2));
> table[2,8] <- mean(apply(abs(rmheight7 - mheight0)^2, 1, mean)^(1/2));
> table[2,9] <- table[2,8]/table[2,7];
> table[2,10] <- mean(apply(abs(mheight8 - mheight0)^2, 1, mean)^(1/2));
```

```
> table[2,11] <- mean(apply(abs(rmheight8 - mheight0)^2, 1, mean)^(1/2));
> table[2,12] <- table[2,11]/table[2,10];
> table[3,1] <- mean(apply(abs(mheight2 - mheight0), 1, max));
> table[3,2] <- mean(apply(abs(rmheight2 - mheight0), 1, max));
> table[3,3] <- table[3,2]/table[3,1];
> table[3,4] <- mean(apply(abs(mheight5 - mheight0), 1, max));
> table[3,5] <- mean(apply(abs(rmheight5 - mheight0), 1, max));
> table[3,6] <- table[3,5]/table[3,4];
> table[3,7] <- mean(apply(abs(mheight7 - mheight0), 1, max));
> table[3,8] <- mean(apply(abs(rmheight7 - mheight0), 1, max));
> table[3,9] <- table[3,8]/table[3,7];
> table[3,10] <- mean(apply(abs(mheight8 - mheight0), 1, max));
> table[3,11] <- mean(apply(abs(rmheight8 - mheight0), 1, max));
> table[3,12] <- table[3,11]/table[3,10];

> table
```

	Splines	Rearranged	Ratio(R/O)	Fourier	Rearranged	Ratio(R/O)
L1	0.8336007	0.7645912	0.9172151	0.7118916	0.687025	0.9650697
L2	1.0588792	0.9704775	0.9165139	0.8769878	0.847716	0.9666224
Linf	3.9006518	3.0718604	0.7875249	2.3899782	2.302835	0.9635381
	Local poly.	Rearranged	Ratio(R/O)	Kernel(h=1)	Rearranged	Ratio(R/O)
L1	1.042703	0.9141986	0.8767582	1.059231	0.9377153	0.8852797
L2	1.305661	1.1807139	0.9043036	1.332311	1.2216594	0.9169479
Linf	4.129122	3.7019047	0.8965356	3.989148	3.8093732	0.9549339

As we see in this experiment, all estimation methods exhibit noticable decreases in the L^p errors after rearrangement. For all four methods considered, we find that the rearranged functions more accurately estimate the original CEF with improvements in error from $3 - 22\%$.

References

[1] V. Chernozhukov, I. Fernandez-Val, and A. Galichon. Improving point and interval estimators of monotone functions by rearrangement. *Biometrika*, 96(3):559, 2009.

[2] V. Chernozhukov, I. Fernández-Val, and A. Galichon. Quantile and probability curves without crossing. *Econometrica*, 78(3):1093–1125, 2010.

[3] GH Hardy, JE Littlewood, and G. Polya. *Inequalities, 2nd ed.* Cambridge Univ. Press, Cambridge, 1952.