We investigate using k-nearest neighbors regression to approximate the drag function. For the reasons listed below, this is not really a smart thing to do; we're doing it simply to learn syntax not to usefully approximate the function. What wrong with KNN regression here? First, because the results are based on just 22 physical hulls, there is not a lot of diversity in the data. If we, as usual, selected a random test set, then, when tested, KNN would immediately find training cases where the four hull geometry factors exactly matched the one being tested (i.e., unfairly the results will be based on the exact same hull as there is zero distance for those variables). The result will be an average over that hull's nearby Fr that happened to be in the training set. KNN would mostly be acting as a table lookup function for the hull. Second our first step in dealing with this data will be to scale, and we know that boat speed is by far the most significant factor. To get around the first problem instead of picking a random subset of data points to be in the test set, I'm going to select two random hulls (12 & 22) to be in the test set.

KNN regression (in contradistinction to KNN classification) is not in base R, so:

```
library("FNN")
df=read.csv("yacht_hydrodynamicsH.csv")
str(df)
```

Note that this data.frame includes the `hull` number. We learned in our first encounter with this dataset that there was multicollinearity: `P.C` ≈ 1.54 `B.D` `L.B`^{2}/`L.D`^{3} so we'll drop that variable. I select `hull` 12 & 22 to be in the test set (and not in the training set) and then also drop `hull` from the datasets.

```
dftest=df[df$hull==12 | df$hull==22,c(1,3,4,5,6,7)]
dftrain=df[!(df$hull==12 | df$hull==22),c(1,3,4,5,6,7)]
str(dftest)
'data.frame': 28 obs. of  6 variables:
 $ CoB: num  0 0 0 0 0 0 0 0 0 0 ...
 $ L.D: num  5.1 5.1 5.1 5.1 5.1 5.1 5.1 5.1 5.1 5.1 ...
 $ B.D: num  3.94 3.94 3.94 3.94 3.94 3.94 3.94 3.94 3.94 3.94 ...
 $ L.B: num  3.51 3.51 3.51 3.51 3.51 3.51 3.51 3.51 3.51 3.51 ...
 $ Fr : num  0.125 0.15 0.175 0.2 0.225 0.25 0.275 0.3 0.325 0.35 ...
 $ RR : num  0.08 0.26 0.5 0.83 1.28 1.9 2.68 3.76 5.57 8.76 ...
```

Feel free to select two other `hull`s to be in your test set. Next, following the usual rule book, we treat all the remaining independent variables as co-equal (even though we know `Fr` is most important) by scaling. (Distances will now be calculated using comparably-valued coordinates.) Note that we must not include the test dataset in determining the scale and that we should then scale the test set using the scale determined from the training dataset, exactly as if we had never seen the test data. Also it turns out that the KNN functions really don't want the dependent variable variable in with the independent variables, so we'll strip that out also.

```
dftrain0=scale(dftrain[,1:5])
stdev=apply(dftrain[,1:5], 2, sd)
avg=colMeans(dftrain[,1:5])

dftest0=scale(dftest[,1:5],center=avg,scale=stdev)
```

Now get the results:

```
train.out=knn.reg(dftrain0,,dftrain$RR,k=5)
```

```
sqrt(sum((train.out$pred-dftrain$RR)^2)/train.out$n)
[1] 6.9026
```

Not nearly as good as we did with regression.

```
test.out=knn.reg(dftrain0,dftest0,dftrain$RR,k=5)
sqrt(sum((test.out$pred-dftest$RR)^2)/test.out$n)
[1] 4.650391
```

Usually the test error will be more than the training error.

I want to see (plot) the function that KNN has created for each test hull; for a fixed `hull` the KNN function just depends on `Fr`. I need a data.frame with the fixed geometry variables of a test hull but with a nearly continuous set of `Fr`. And I must scale those `Fr` just as the training set were scaled. I will build a data.frame that replicates the geometry parameters from hull 12 (for me a test hull) and all those `Fr`. Note that the first 14 rows of the test dataset are hull 12, and the following rows are hull 22.

```
fr=seq(.125,.45,length=100)
frs=scale(fr,center=avg["Fr"],scale=stdev["Fr"])
dfplot12=as.data.frame(dftest0[rep(1,100),1:4])
dfplot12$Fr=frs

predict12=knn.reg(dftrain0,dfplot12,dftrain$RR,k=5)
plot(dftest$Fr[1:14],dftest$RR[1:14])
lines(fr,predict12$pred)
```

It does OK until the $Fr = .45$ datapoint (which is a boundary of the training data, limiting neighbors). Now do the same for hull 22:

```
dfplot22=as.data.frame(dftest0[rep(15,100),1:4])
dfplot22$Fr=frs

predict22=knn.reg(dftrain0,dfplot22,dftrain$RR,k=5)
plot(dftest$Fr[15:28],dftest$RR[15:28])
lines(fr,predict22$pred)
```

Not at all good. We might as well look at hulls that were in the training set.

```
dfplot1=as.data.frame(dftrain0[rep(1,100),1:4])
dfplot1$Fr=frs

predict1=knn.reg(dftrain0,dfplot1,dftrain$RR,k=5)
plot(dftrain$Fr[1:14],dftrain$RR[1:14])
lines(fr,predict1$pred)

dfplot2=as.data.frame(dftrain0[rep(15,100),1:4])
dfplot2$Fr=frs

predict2=knn.reg(dftrain0,dfplot2,dftrain$RR,k=5)
plot(dftrain$Fr[15:28],dftrain$RR[15:28])
lines(fr,predict2$pred)
```