# Proper Permutation Kern Dist BF

*Naveed Merchant*

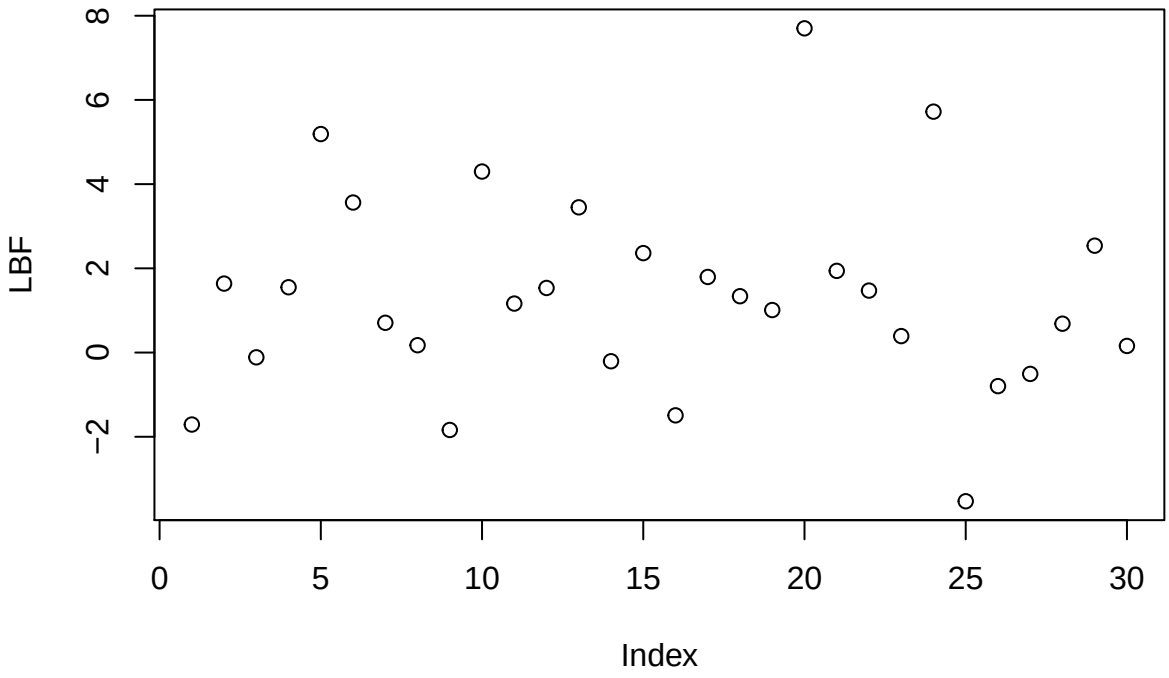
*October 16, 2018*

For now all computations are done with code provided.

We can always use importance sampling if desired.

We first get 2 distributions, and examine to see if we can compose a test that can distinguish the two.

```
set.seed(1000)
source("MarginalLikIntfunctions.R")
LBF <- c()
for(D in 1:30)
{
  dataset1<- rnorm(500)
  dataset2<- rnorm(500)
  XT1 <- dataset1[1:(length(dataset1)*.3)]
  XV1 <- dataset1[-(1:(length(dataset1)*.3))]
  XT2 <- dataset2[1:(length(dataset2)*.3)]
  XV2 <- dataset2[-(1:(length(dataset2)*.3))]
  ExpectedKernML <- logmarg.kern(XT1,XV1)
  FullKernML <- logmarg.kern(c(XT1,XT2),XV1)
  LBF[D] <- FullKernML[[2]] - ExpectedKernML[[2]]
}
plot(LBF)
```



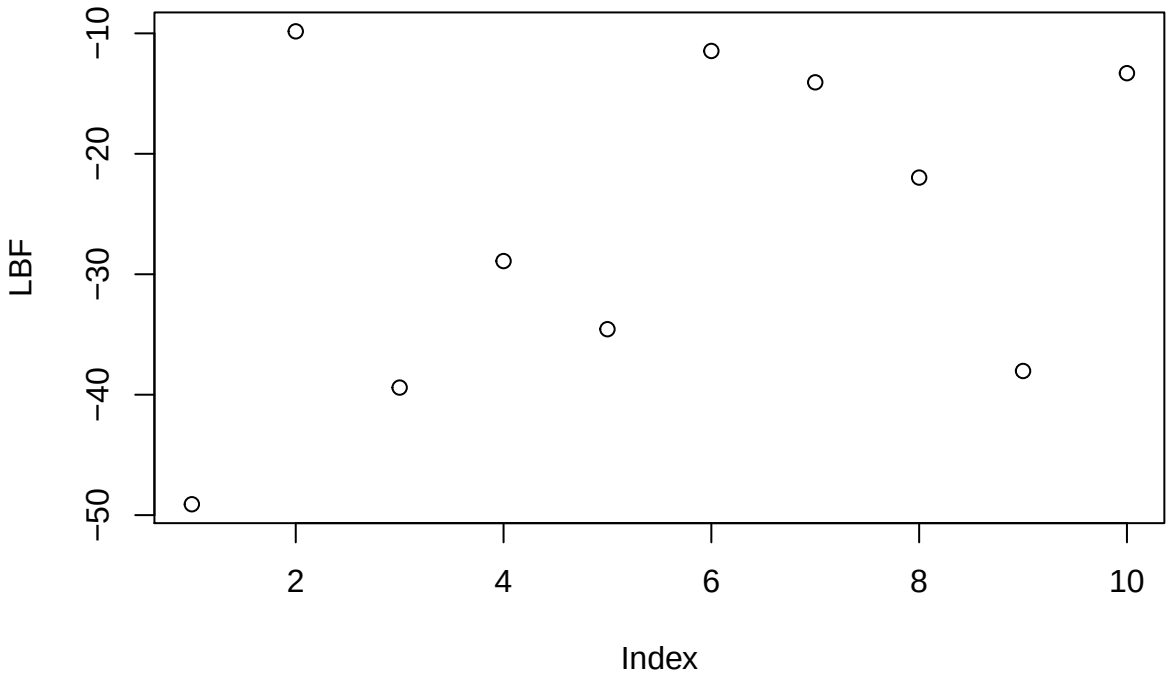

This is what we wanted!

The data is distributed as we expected it to be (positive LBF mean as there is more training data that is useful). The LBF is never too negative.

For accuracy's sake we use the integrate function, I think its slower (suprisingly) then Importance sampling and defintely slower then our very poor MC.

Let's try to use a different distribution to see how senstive this test is.

```
LBF <- c()
for(D in 1:10)
{
  dataset1<- rnorm(500)
  dataset2<- rexp(500)
  XT1 <- dataset1[1:(length(dataset1)*.3)]
  XV1 <- dataset1[-(1:(length(dataset1)*.3))]
  XT2 <- dataset2[1:(length(dataset2)*.3)]
  XV2 <- dataset2[-(1:(length(dataset2)*.3))]
  ExpectedKernML <- logmarg.kern(XT1,XV1)
  FullKernML <- logmarg.kern(c(XT1,XT2),XV1)
  LBF[D] <- FullKernML[[2]] - ExpectedKernML[[2]]
}
plot(LBF)
```

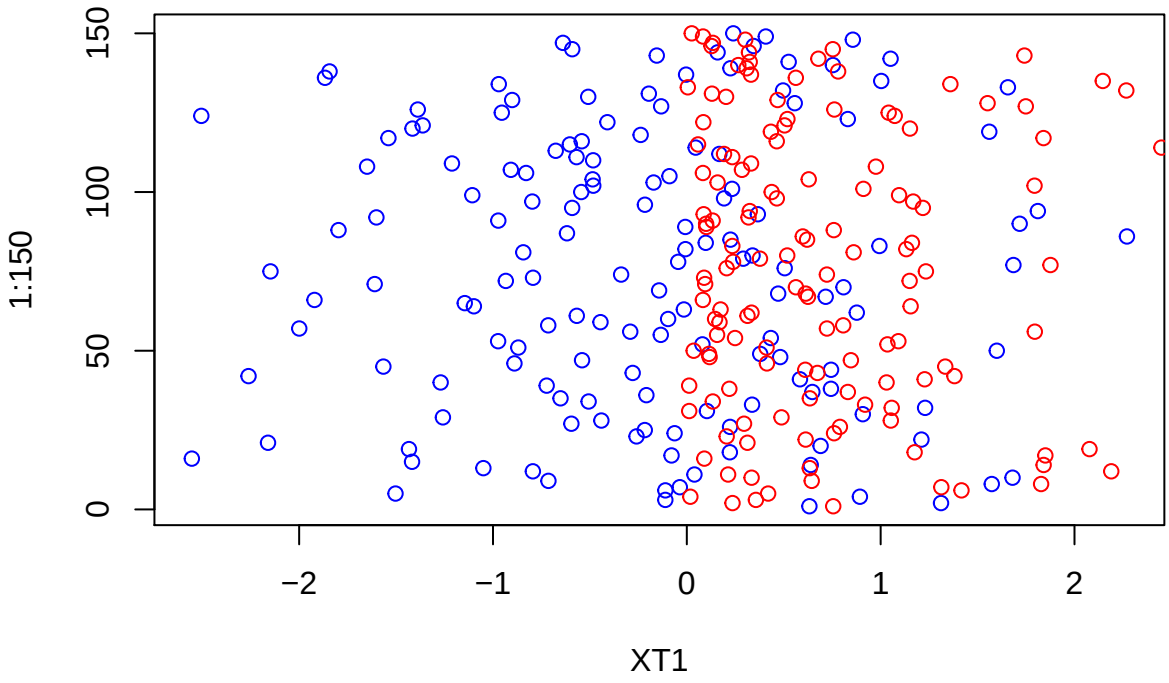



That's not a good sign.

The test was doing fine, and then started to misbehave.

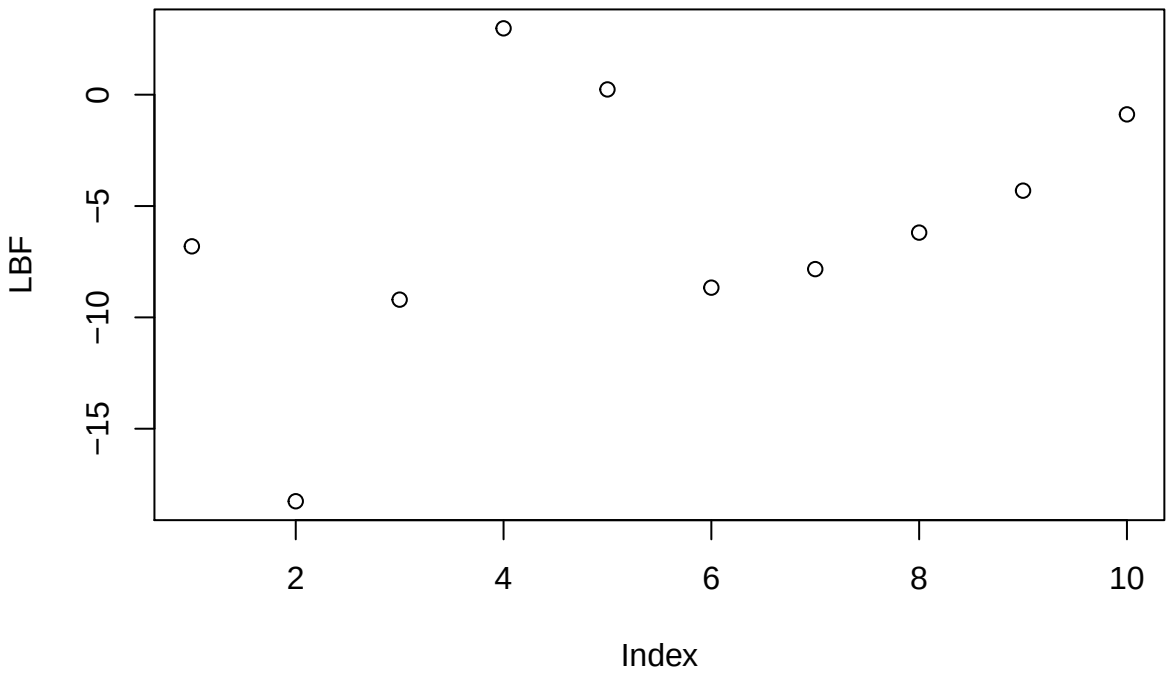
We plot graphs.

```
plot(XT1,1:150,col = "blue")
points(XT2,1:150,col = "red")
```




In hindsight choosing exponential distribution is a very questionable idea in general, we choose a t distribution with heavy tails (as it looks a little more familiar)

```
LBF <- c()
for(D in 1:10)
{
  dataset1<- rnorm(500)
  dataset2<- rt(500, df = 4)
  XT1 <- dataset1[1:(length(dataset1)*.3)]
  XV1 <- dataset1[-(1:(length(dataset1)*.3))]
  XT2 <- dataset2[1:(length(dataset2)*.3)]
  XV2 <- dataset2[-(1:(length(dataset2)*.3))]
  ExpectedKernML <- logmarg.kern(XT1,XV1)
  FullKernML <- logmarg.kern(c(XT1,XT2),XV1)
  LBF[D] <- FullKernML[[2]] - ExpectedKernML[[2]]
}
plot(LBF)
```




Interesting?

We are currently testing to see if given two datasets, we are concerned with whether they come from a common distribution, (I believe this is the same as asking if they are "identically distributed").

If we want to see if we have 1 dataset, and we are checking whether that dataset has everything coming from a common distribution, I think we can reduce it to this problem, but I want to make sure...

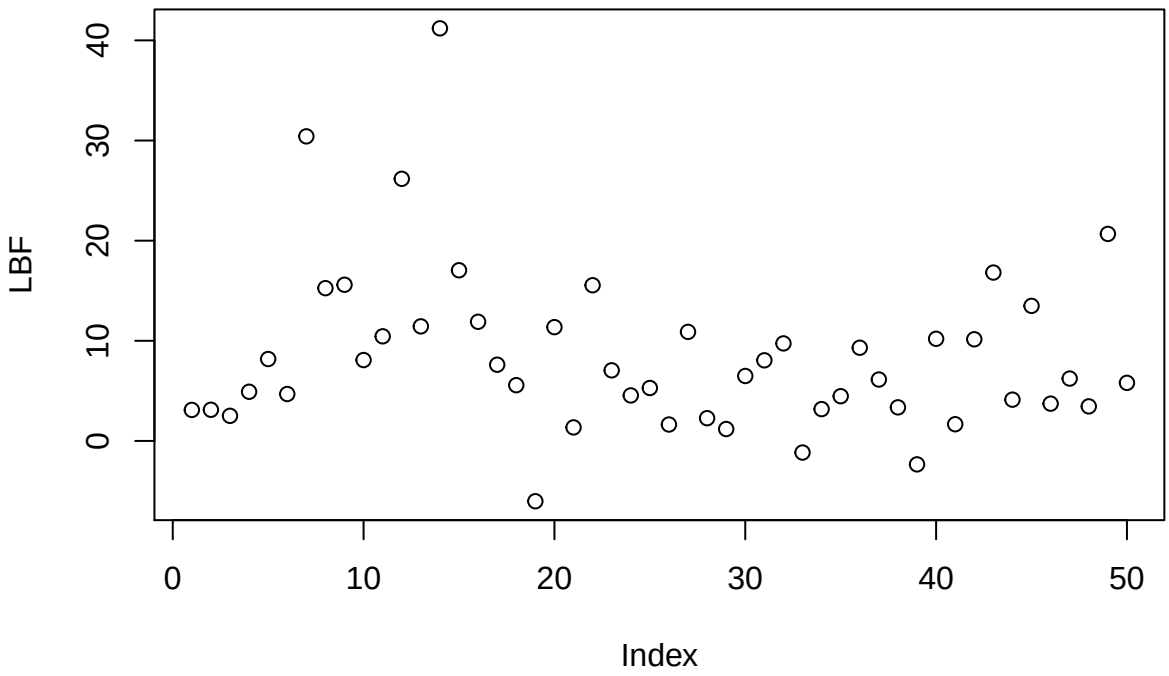
Regardless, before jumping to that ship, we examine the same idea with a Prior Free setting.

After reading Van der Wall's paper, it may be a good idea to consider CV as a choice for choosing b's and then going along with this method as well.

If 1-fold CV is time consuming (which it probably will be), we can consider Monte Carlo CV.

```
LBF <- c()
for(D in 1:50)
{
  dataset1<- rnorm(500)
  dataset2<- rt(500, df = 4)

  Y1 <- dataset2[1:(length(dataset2)*.3)]
  Y2 <- dataset2[(length(dataset2)*.3 + 1):(length(dataset2)*.6)]
  Y3 <- dataset2[(length(dataset2)*.6 + 1):(length(dataset2))]

  X1 <- dataset1[1:(length(dataset1)*.3)]
  X2 <- dataset1[(length(dataset1)*.3 + 1):(length(dataset1)*.6)]
  X3 <- dataset1[(length(dataset1)*.6 + 1):(length(dataset1))]

  GaussKernLik <- function(h,X1,X2)
```

```
  {
    prod <- 0
    k <- length(X1)
    n <- length(X2)
    for(j in 1:n)
    {
      sum <- 0
      for(i in 1:k)
      {
        sum <- sum + exp(-.5*((X2[j]-X1[i])/h)^2)
      }
      prod <- prod + log(sum) + log((1/sqrt(2*pi))) + log((k*h)^(-1))
    }
    return(prod)
  }
    Optimumbandwidthreg <- optimize(GaussKernLik, X1 = X1, X2 = X2, lower = .00001, upper = 5, maximum =

  Optimumaugbandwidthreg <- optimize(GaussKernLik, X1 = c(X1,Y1), X2 = c(X2,Y2), lower = .00001, upper =

#The questionable nature of how to split is why I believe the above step could be improved

  BFPriorFree <- GaussKernLik(Optimumbandwidthreg$maximum,X1 = X1, X2 = X3) - GaussKernLik(Optimumaugba

  LBF[D] <- BFPriorFree
}

plot(LBF)
```




How pleasant, this seems to work quite nicely as well.

It seems to be far faster then the other tests.

There is still some work to do. The mean is clearly positive, but I think some fine-tuning can be done.

For instance I'm not using the Full datasets and there is some discussion to be had.

This is far from optimized but I think the idea is nice!