# HW08 Registration

Drew Remmenga

2022-12-03

- You are encouraged to use web resources and class materials
- You can work together but hand in your own work.
- Be spare in what you include and you will lose credit if you include too much extraneous output or information. All parts questions count for an equal number of points.
- Any subproblems marked GRAD are required for 500 level students but will serve as an extra credit question for the 400 level students.
- Please send me email if you have questions or any concerns. `nychka@mines.edu`
  - Hand in your work in pdf format in Gradescope. You can keep the questions as part of what you hand in but you should begin your *answer* on a separate page. You can use `\newpage` to create a page break in your work.

## Reformatting text and code that are not your answers

It is harder to grade homework when all of the introduction and question text are included. Either comment out the answers without just deleting them or put your answers in a different color.

To use the html commenting format:

```
<!--
This text is now commented out and will not be part of the rendered output.
-->
```

To change the color you need to have Latex working:

```
\textcolor{magenta}{This is a magenta block of text. }
```

And the rendered version in pdf:

This is a magenta block of text.

## Points

All subsections of the problems count equally for 10 points:

- 400 level 1(a), 1(b) 1(c) 2(a) 2(b) EXTRA total 40
- 500 level 1(a), 1(b) 1(c) 2(a) 2(b) total 50

### Setup

```
setwd("~/School/MATH498/HW8/")
suppressMessages(library(fields))
source("estimationScriptTest.R")
```

## Problem 0 – In Class Example

Here is a reproduction of the inclass example to use as a guide to work on a different data set. First include this handy function to use in this work

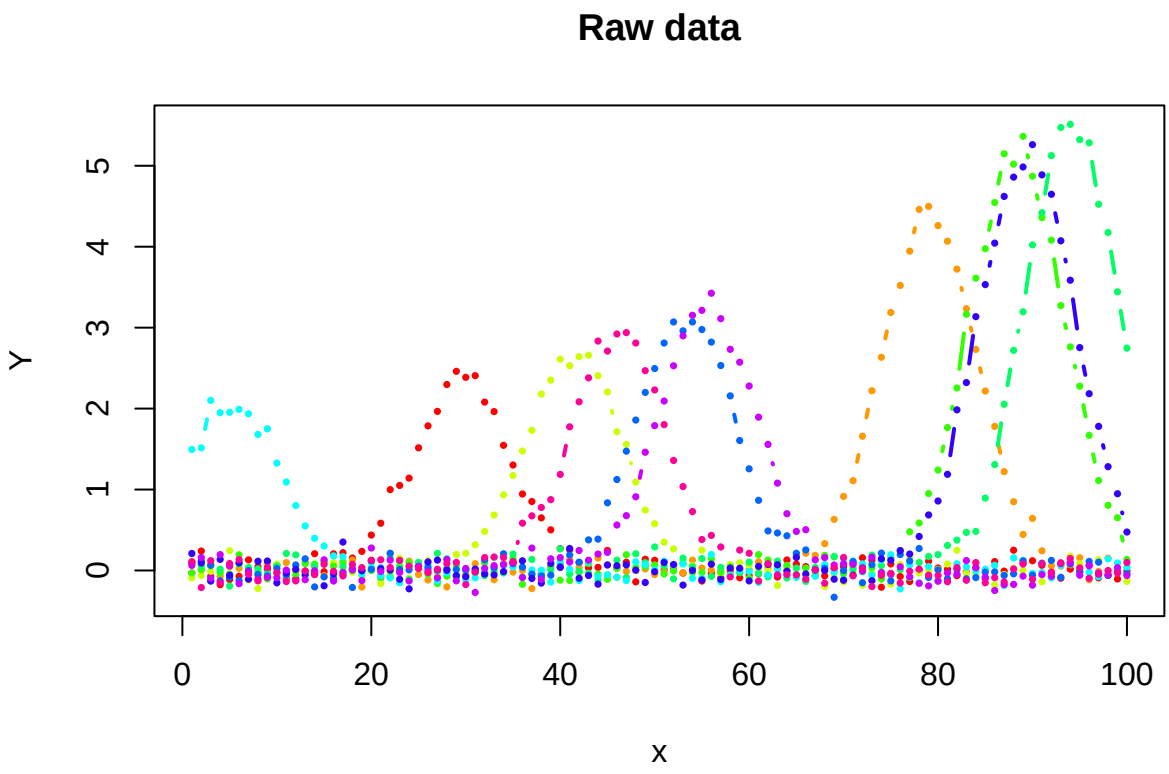
```
shiftTemplate <- function(x, center, template) {
  splint(template$x + center, template$y, x)
}
```

```
#
#
set.seed(123)
x<- 1:100
n<- length(x)
L<- 10
trueCenter<- runif(L,1,100)
trueScale<- 4*(trueCenter/100)^2 + 2
Y0<- matrix(NA, n,L)
constant<- dnorm(0, sd=5) # maximum of N(0,1)
for( k in 1:L){
  Y0[,k]<- dnorm(x, mean=trueCenter[k], sd= 5)*trueScale[k]/constant
}

set.seed(112)
Y<- Y0 + .1*matrix( rnorm(L*n),n,L)
```

### Plot of raw data

```
matplot( x,  Y , type = "b",  col = rainbow(L), pch=16,
         lwd = 2, lty = 1 , cex=.5)
title("Raw data")
```




### create starting values

Need to specify the beginning template and the center and alpha and also the amount of smoothing.

```
templateGrid <- seq(-50,50, length.out = 500)
# starting shape for the template
templateStart <- list(x = templateGrid,
                      y = exp(-abs(templateGrid)/10)  )

ind <- apply( Y, 2, which.max)
parCenterOLD<- x[ind]
parAlphaOLD<- rep( 1,ncol(Y))
template<- templateStart

DFBump<-15
```

## Estimation code

The code below (see the file estimationScript ) defines the function **register**. This is a convenient to rerun this code in several places without having to cut and paste. Results are returned as a list with components **parCenter**, **parAlpha** and **template**.

Note that this only works when the objects `x,Y, parAlphaOLD, parCenterOLD, template, DFBump` are defined correctly for your problem. Change these to run on another problem.

```
look<- register( x, Y, parAlphaOLD, parCenterOLD, template, DFBump )
```

```
## [1] "running script with  x, Y, parAlphaOLD, parCenterOLD, and template"
## [1] "make sure these are correct!"
## iteration 1
## 1  2  3  4  5  6  7  8  9  10  1 center  0.7828501 template  0.1064527
## iteration 2
## 1  2  3  4  5  6  7  8  9  10  2 center  0.40028 template  0.003927239
## iteration 3
## 1  2  3  4  5  6  7  8  9  10  3 center  0.1017341 template  0.0009243365
## iteration 4
## 1  2  3  4  5  6  7  8  9  10  4 center  0.0194249 template  0.0002278862
## iteration 5
## 1  2  3  4  5  6  7  8  9  10  5 center  0.003663404 template  0.0001536404
## [1] " All done!"
```

```
####source("estimationScript.R")
```

### Examine results

The converged template is quite different than the (poor!) starting guess. Fit to the original data looks good. Here to understand the model scrutinize how `look$parAlpha[k] * shiftTemplate(x, look$parCenter[k], look$template )` should work for the curve `k` .

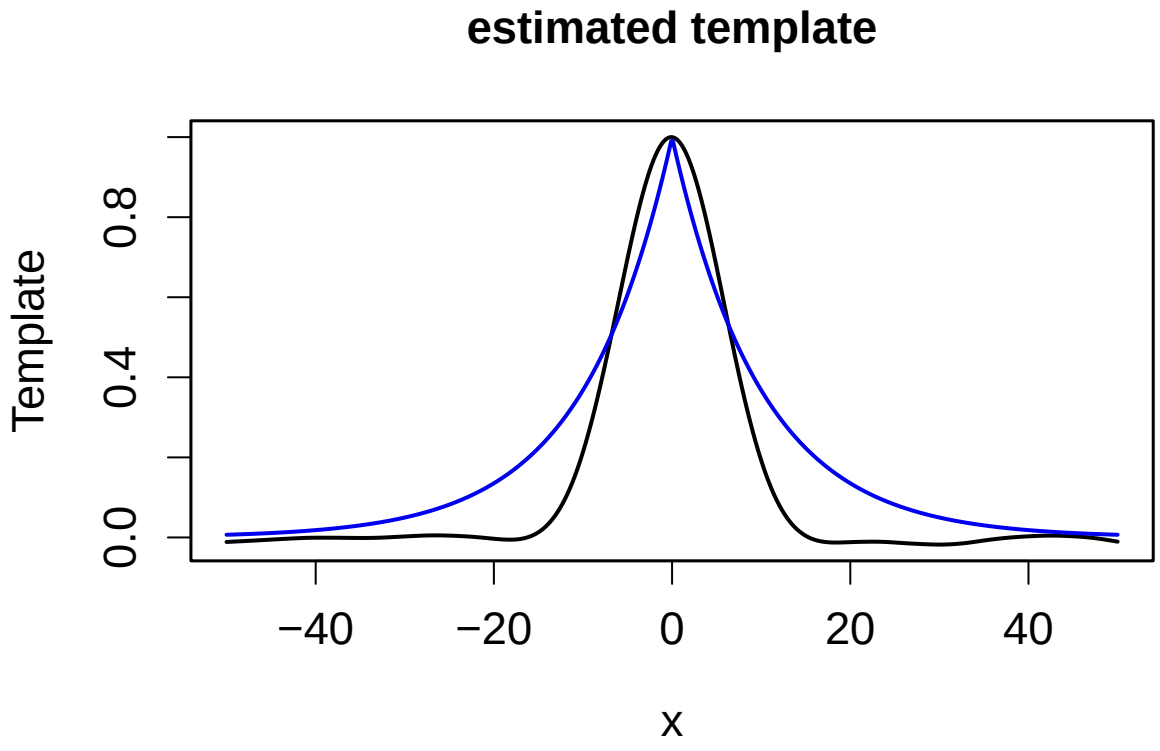
In notation, if $T$ is the template function we have

$$\hat{f}_k(x) = \alpha_k T(x - \text{center}_k)$$

the only complication is we keep track of $T$ as a discrete set of x and y points.

```
fields.style()
plot( look$template, type="l", xlab="x", ylab="Template", lwd=2 )
lines( templateStart, col="3", lwd=2)

title("estimated template")
```

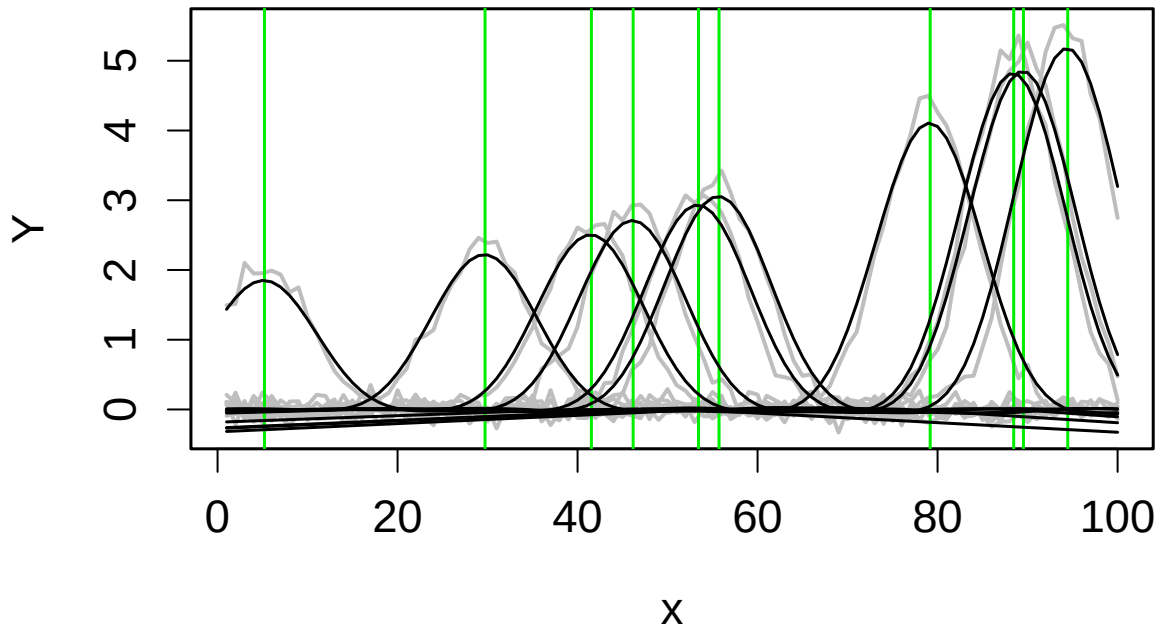



```
# data with superimposed shifted and scaled template


matplot(
  x, Y , type = "l", pch = 16, col = "grey",
  lwd = 2, lty = 1 )

xline( look$parCenter, col="green2")
```

```
for( k in 1: ncol(Y)){
  lines(x, look$parAlpha[k] * shiftTemplate(x,  look$parCenter[k], look$template ) )
}
```



# Problem 1

The goal is to try to register the weight functions for the ice core data in the same way as the example. Here to reduce computation I cut down the size of the weights.

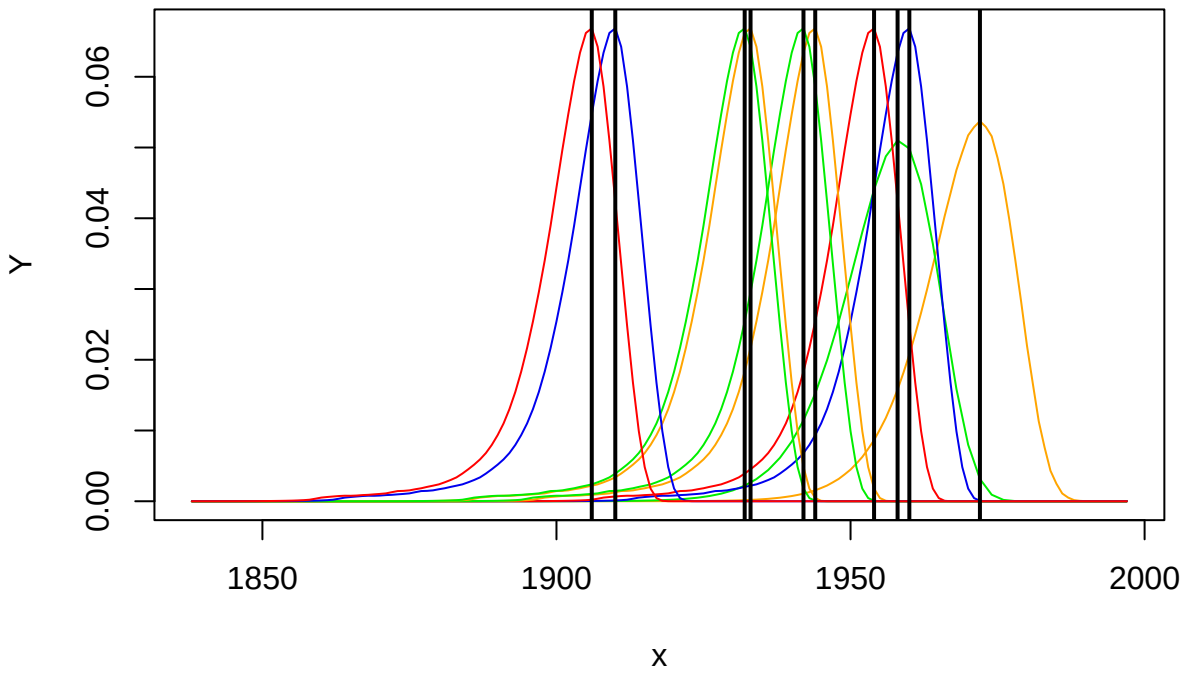
```
load("~/School/MATH498/HW8/CO2IceCore.rda")
Y<- t(W)
x<- uGrid
ind<-(1:160)+ (500-160)
Y0<- Y[ ind, (1:10)+3  ]
x<- x[ind]

Y<- Y0
# Y now is what you will work on

matplot( x, Y, type="l", lty=1)

ind <- apply( Y, 2, which.max)
parCenterOLD<- x[ind]
parAlphaOLD<- rep( 1,ncol(Y))
templateGrid <- seq(-50,50, length.out = 500)
# starting shape for the template
templateStart <- list(x = templateGrid,
                      y = exp(-abs(templateGrid)/8)  )
template<- templateStart


xline( parCenterOLD, col="black", lwd=2)
```




Below we run the registration function on these data with degrees of reedom 10.

```
DFBump<-10
look1<- register( x, Y, parAlphaOLD, parCenterOLD, template, DFBump )
```

```
## [1] "running script with  x, Y, parAlphaOLD, parCenterOLD, and template"
## [1] "make sure these are correct!"
## iteration 1
## 1  2  3  4  5  6  7  8  9  10  1 center  1.412007 template  0.1075181
## iteration 2
## 1  2  3  4  5  6  7  8  9  10  2 center  0.1683461 template  0.0005381972
## iteration 3
## 1  2  3  4  5  6  7  8  9  10  3 center  0.0140741 template  0.000483925
## iteration 4
## 1  2  3  4  5  6  7  8  9  10  4 center  0.01409024 template  0.0004846817
## iteration 5
## 1  2  3  4  5  6  7  8  9  10  5 center  0.0140781 template  0.0004842319
## [1] " All done!"
```

```
#####source("estimationScript.R")
```

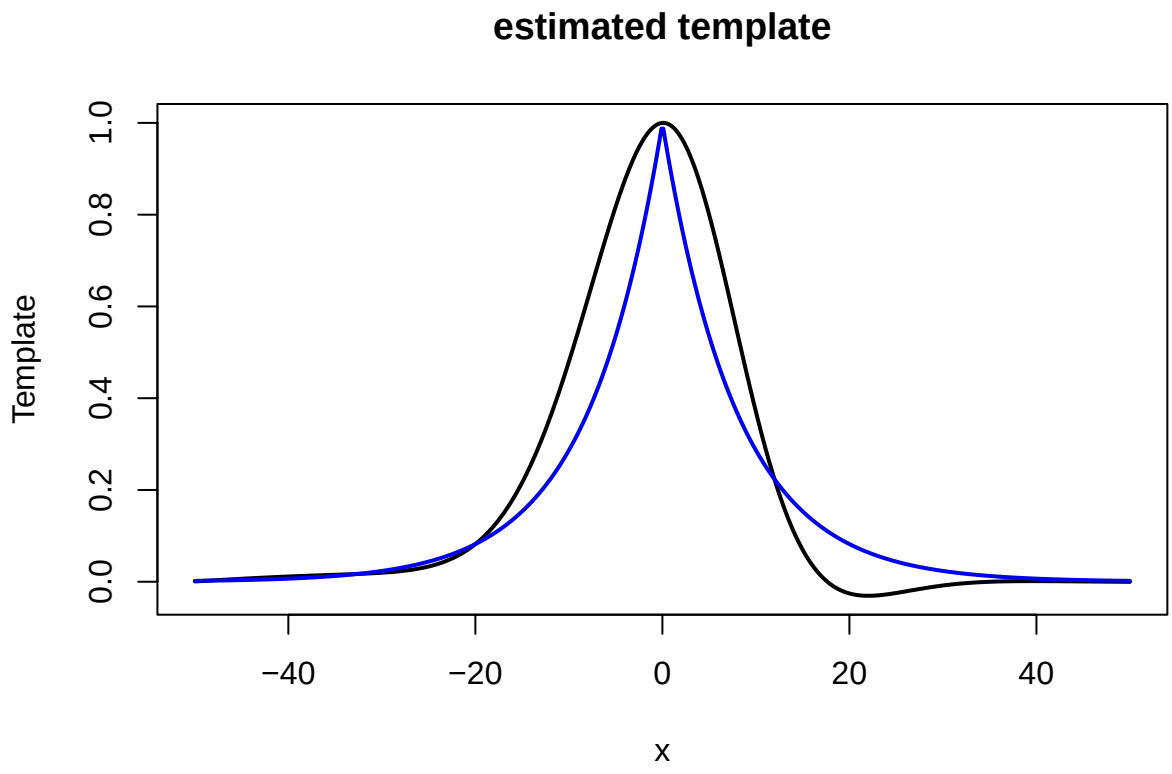
## 1(a)

Plot the fitted results with the raw data just as in the class example above. Just use the same code because the outout objects from the script are all named the same. (but note the output list is `look1` instead of `look`.)

Does the fit seem good? If not what are some problems?

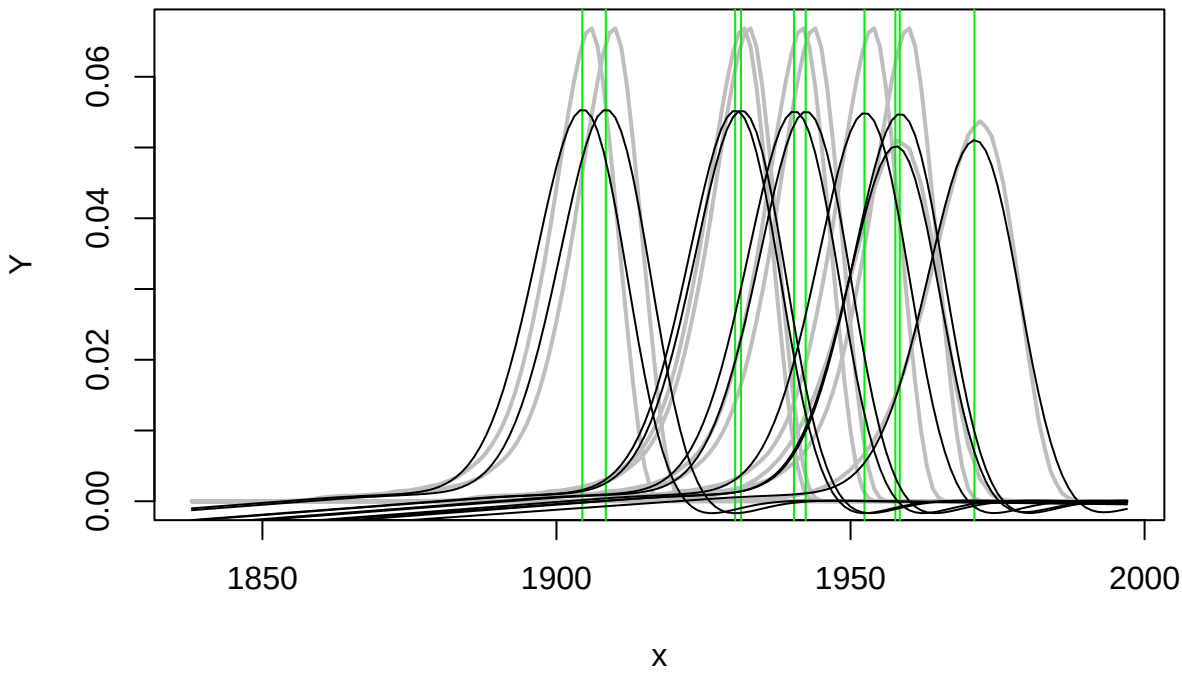
```
plot( look1$template, type="l", xlab="x", ylab="Template", lwd=2 )
lines( templateStart, col="3", lwd=2)

title("estimated template")
```




```
matplot(
  x, Y , type = "l", pch = 16, col = "grey",
  lwd = 2, lty = 1 )

xline( look1$parCenter, col="green2")
for( k in 1: ncol(Y)){
  lines(x, look1$parAlpha[k] * shiftTemplate(x,  look1$parCenter[k], look1$template ) )
}
```




Some problems are the clustering of the slopes. ## 1(b) For your fit in 1(a) does it seem that the algorithm is converging? Yes. ## 1(c) Does your fit change with 25 degrees of freedom? No. Does your fit improve with 50 degrees of freedom? Yes.

# Problem 2

## 2(a)

How would you change the code in `estimationScriptTest.R` in particular the lines

```
out <- nls(Y[, j] ~
               alpha*shiftTemplate(x,  center, template ),
             start = (list(center = parCenterOLD[j], alpha=parAlphaOLD[j])
             )
   )
```

and

```
  registerData[ind, j] <- splint(xCenter,
                                  Y[, j]/ parAlpha[j], templateGrid[ind])
```

if you just wanted to use a template had the shape of a normal density function with a standard deviation of 5.

(You don't need to implement this just say how you would do it. ) We would use rnorm(0,5) in place of the generation of template Grid.

## 2(b) GRAD

How would you changeto this algorithm to add a parameter, $\sigma$, that scales the width of the bump? I.e. a model following the formula

$$\hat{f}_k(x) = \alpha_k T\left(\frac{(x - \text{center}_k)}{\sigma_k}\right)$$

Here $k$ is the $k^{th}$ curve.