Assignment 2
MY 459 / MY 559

Benjamin Lauderdale
Methodology Department
London School of Economics

1 Preliminaries

Before starting, set the working directory in R to be the folder where you have saved the data file. You can do this using the menus in the R application, or via the command line `setwd` command.

The data set for this class assignment is the data used in lecture on attitudes towards homosexuality in the US. The data include all observations between 1972 to 2012 on three variables: the year of the survey (`year`), the age of the respondent (`age`) and the response to the question about homosexuality (`homosex`). The data file is in Stata format, so we need to load the foreign library to access it.

```
library(foreign)
data <- read.dta("GSS7212_R3_Extract.dta")
```

For much of the assignment, it will be easier to work with these data if we remove all the respondents for whom any of the variables are missing. The following command creates a new data frame `data2` with only the observations (rows) which have no missing data in any column:

```
data2 <- data[rowSums(is.na(data)) == 0,]
```

The data do not have the birth year for the respondents, but we can calculate it easily (we may be off by a year due to rounding, but we are not going to worry about that).

```
data2$birthyear <- data2$year - data2$age
```

We are going to recode the variable `homosex` so that it is a binary variable that equals 1 if a respondent gave the *always wrong* or *almost always wrong* responses. Note that this is the reverse of how we defined the variable in lecture.

```
data2$wrong <- as.numeric(data2$homosex == "always wrong" |  data2$homosex == "almst always wrg")
```

Let's check that this worked:

```
print(table(data2$homosex,data2$wrong))
```

```
##
##                          0     1
##    iap                   0     0
##    always wrong          0 21527
##    almst always wrg      0  1575
##    sometimes wrong    2240     0
##    not wrong at all   7265     0
##    other                81     0
##    dk                    0     0
##    na                    0     0
```

Whereas in lecture we focused on all the individuals surveyed in 2010, in this assignment we will focus on individuals born in particular years (birth cohorts). We need to select out the cases we are interested in from the data set. Here, we are going to select the *baby boomer* cohort, which we will define as 1946 to 1955. These were individuals born in the decade immediately after the end of the World War II, which saw a spike in birth rates.

```
data3 <- data2[data2$birthyear >= 1946 & data2$birthyear <= 1955 ,]
```

2 Writing a Kernel Regression Function

We are going to start with a slight improvement on the last version of the kernel regression function that I defined in lecture, which calculates and returns the root mean square error of the kernel regression in addition to everything else.

```
weighted.mean <- function(x, w) sum(x * w)/sum(w)
rmse <- function(fitted,observed) sqrt(mean((observed-fitted)^2))

kreg <- function(x,y,bw,kernel = "uniform"){

    output <- list(x=x,y=y,bw=bw,kernel=kernel)
    class(output) <- 'kreg'

    # specify kernel function
    if (kernel == "uniform") kernelf <-
        function(xdiff) dunif(xdiff,min=-bw,max=bw)

    # define function to calculate a single fitted value
    fittedf <- function(xfitted) weighted.mean(y, kernelf(xfitted - x))

    # apply fitted value function to each observation and calculate RMSE
    output$fitted <- sapply(x,fittedf)
    output$rmse <- rmse(output$fitted,output$y)

    return(output)

}
```

3 Checking your function

Any time you have written a function, you should do some checks to make sure it works. Here are some that make sense for this case, but in general whenever you program a function, you should check that it does what you expect.

3.1 Check the Output Object

To check that the structure of the output is what we intended, you can run the `str()` command on the output to see how the list (or other) object is put together:

```
        kreg5.fit <- kreg(data3$year,data3$wrong,bw=5)
        print(str(kreg5.fit))

## List of 6
##  $ x     : int [1:6820] 1973 1973 1973 1973 1973 1973 1973 1973 1973 1973 ...
##  $ y     : num [1:6820] 0 1 1 1 0 1 0 0 1 1 ...
##  $ bw    : num 5
##  $ kernel: chr "uniform"
##  $ fitted: num [1:6820] 0.626 0.626 0.626 0.626 0.626 ...
##  $ rmse  : num 0.474
##  - attr(*, "class")= chr "kreg"
## NULL
```

3.2 Special Cases

With a uniform kernel, if the bandwidth is greater than the range of the data, then all fitted values will be identical. Let's check that this is true by using 100 years as the bandwidth:

```
        kreg100.fit <- kreg(data3$year,data3$wrong,bw=100)
        print(sd(kreg100.fit$fitted))

## [1] 0
```

The standard deviation being zero means that all the values are the same, that is good. But let's make sure that is not the case if we put in a small bandwidth:

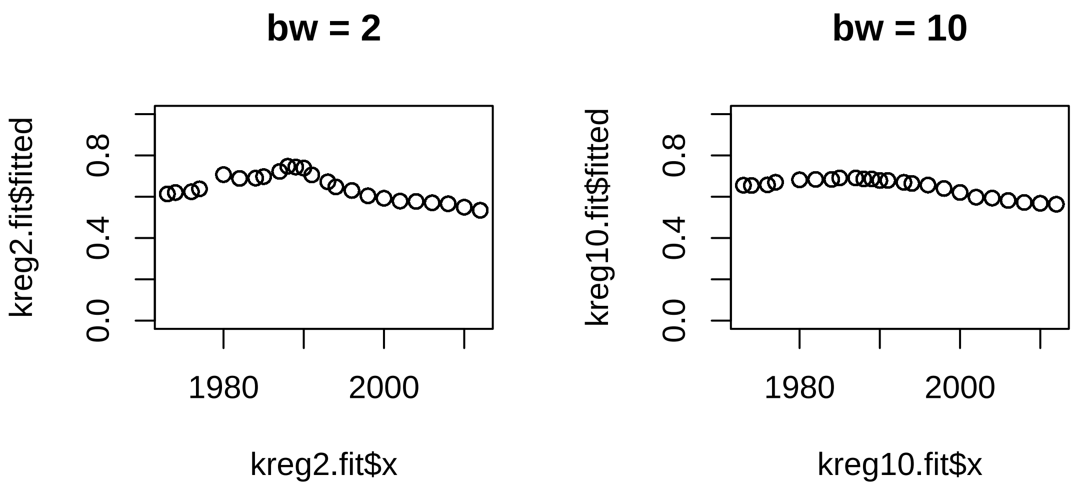
```
        kreg5.fit <- kreg(data3$year,data3$wrong,bw=5)
        print(sd(kreg5.fit$fitted))

## [1] 0.05208482
```

Good! The fitted values are not all the same, this is a sign that the function worked.

3.3 Examine an Example

To see whether the output looks plausible, let's plot the fitted values for two different bandwidths and check that the larger bandwidth looks smoother.

```r
        par(mfrow=c(1,2)) # creates grid of plots with 1 row and 2 columns

        kreg2.fit <- kreg(data3$year,data3$wrong,bw=2)
        plot(kreg2.fit$x,kreg2.fit$fitted,ylim=c(0,1),main="bw = 2")

        kreg10.fit <- kreg(data3$year,data3$wrong,bw=10)
        plot(kreg10.fit$x,kreg10.fit$fitted,ylim=c(0,1),main="bw = 10")
```



The plot with bandwidth 10 is much smoother than with bandwidth 2. These plots are a bit ugly, but we will see how to do better in the next section.

4 Defining a S3 Class

Recall from lecture that we defined a class for the output object of `kreg`:

```r
class(output) <- 'kreg'
```

This line defines the class of the output object to be `kreg` rather than a generic list. We are defining a new class of object called `kreg` that will have the components `x`, `y`, `bw`, etc. Once we have done this, we can write further functions that specifically expect a list with these components. In lecture, we wrote a custom plotting function for `kreg` objects:

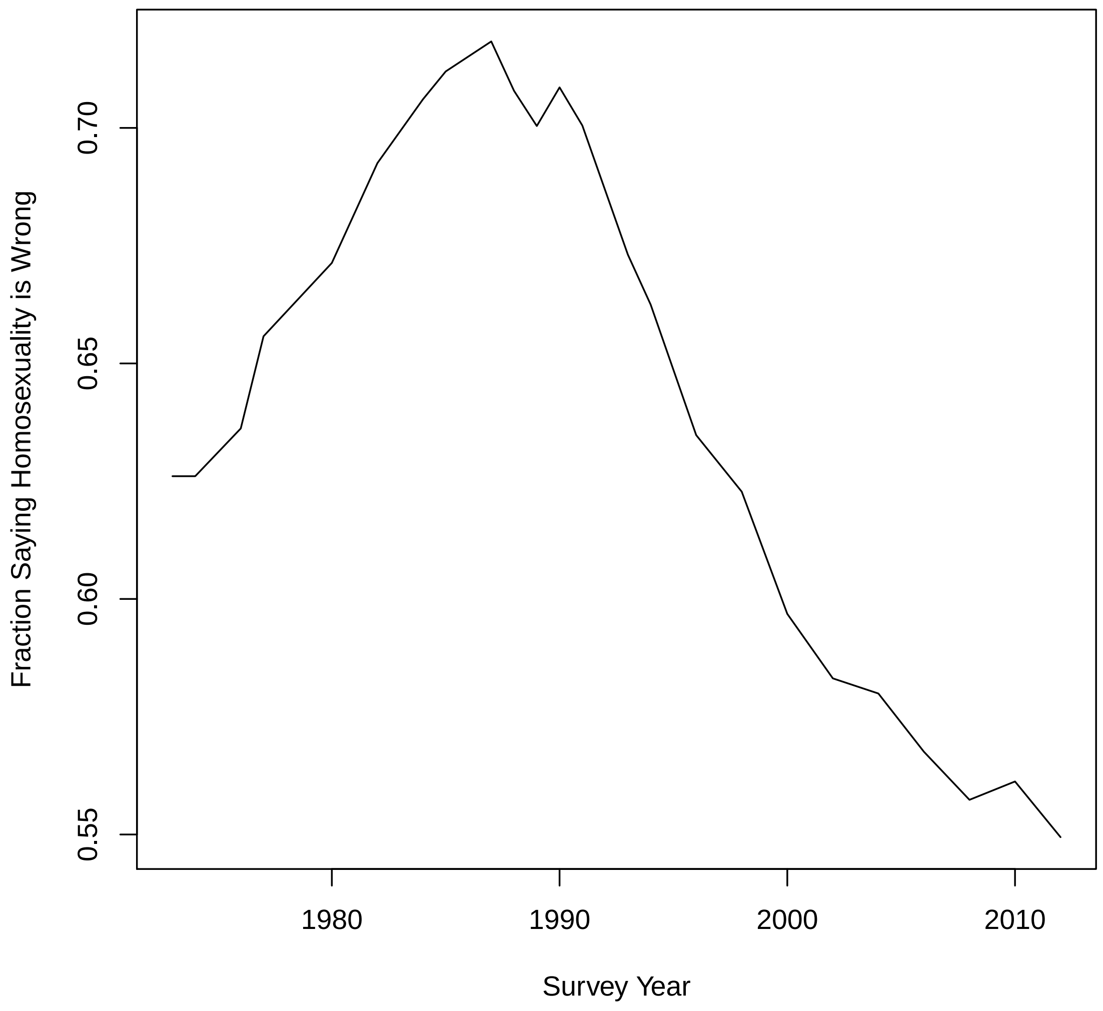
```r
plot.kreg <- function(kreg,...){
    plotorder <- order(kreg$x)
    plot(kreg$x[plotorder],kreg$fitted[plotorder],...)
}
```

Note that the ellipsis '...' allows us to pass through arguments to the inner plot function, so we can modify the labels and other features of the plot in the usual ways. We can now call this function either by running it explicitly:

```
plot.kreg(kreg5.fit)
```

or by just using the generic plot() command (with some examples of additional plotting options):

```
plot(kreg5.fit,
type="l",
xlab="Survey Year",
ylab="Fraction Saying Homosexuality is Wrong"
)
```



When you use the `plot()` command, it will first look at the `class` attribute of the first argument. This is what we set earlier. If there is a corresponding `plot.class()` command, it will use that command to create the plot.

We could similarly create a custom print command for our class if we wanted to do so. This might be something like:

```r
print.kreg <- function(kreg.fit){
        print('Kernel Regression',quote=FALSE)
        print(paste('N:',length(kreg.fit$x)),quote=FALSE)
        print(paste('Bandwidth:',kreg.fit$bw),quote=FALSE)
        print(paste('RMSE:',kreg.fit$rmse),quote=FALSE)
}
```

Once you create this function, if you run the command `print(kreg.fit)`, or just run `kreg.fit`, you will no longer see all the list components, but rather just the output defined by print.kreg() in the function above. If you wanted to remove this function so that you could see the default list printing method, you could run:

```r
rm(print.kreg)
```

The `rm` function can also be used on any other R object, including both functions and data structures.

5 Comparing Bandwidths

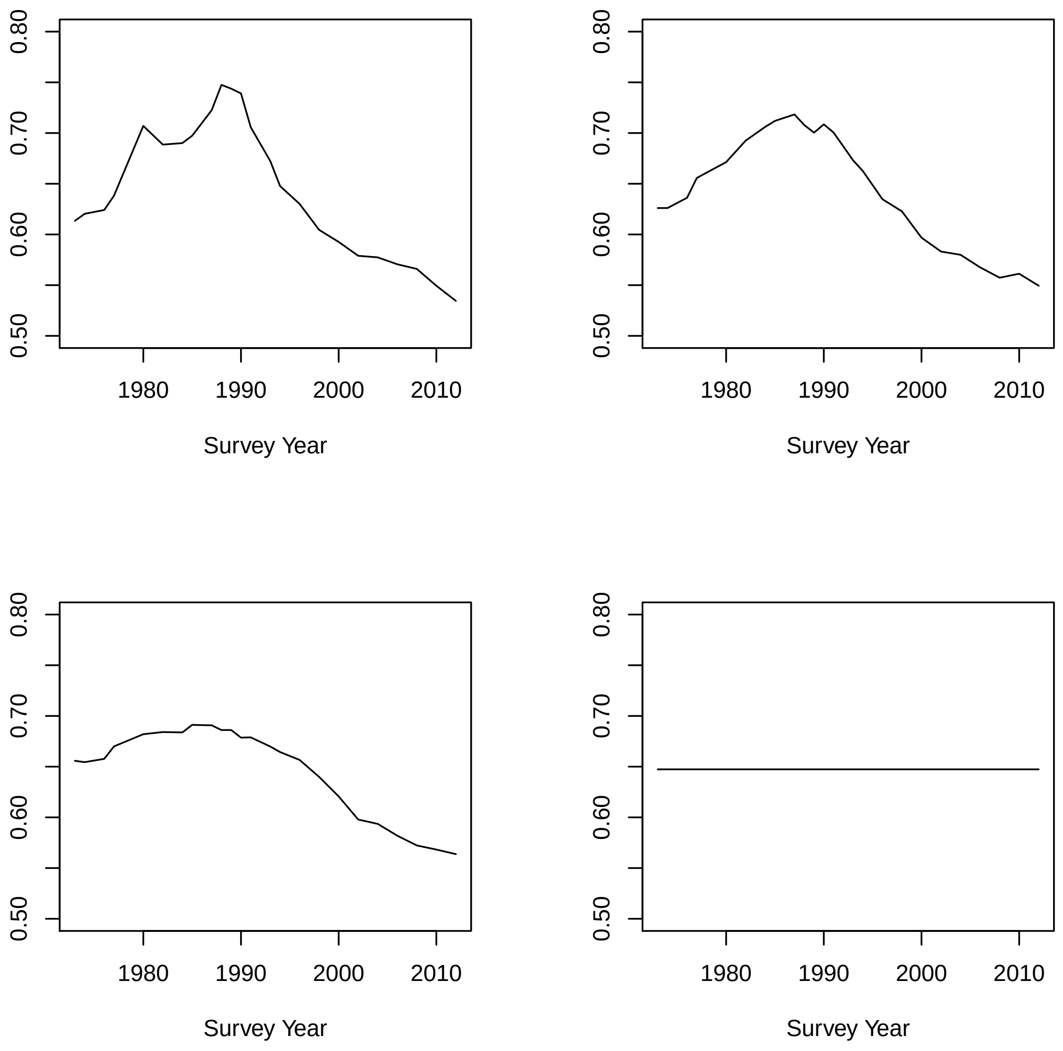
Using the new print method we created above (which you should recreate, if you removed it), we can now compare the RMSEs for the various models we have fit so far.

```r
print(kreg2.fit)

## [1] Kernel Regression
## [1] N: 6820
## [1] Bandwidth: 2
## [1] RMSE: 0.474003859262374

print(kreg5.fit)

## [1] Kernel Regression
## [1] N: 6820
## [1] Bandwidth: 5
## [1] RMSE: 0.47425523407106

print(kreg10.fit)

## [1] Kernel Regression
## [1] N: 6820
## [1] Bandwidth: 10
## [1] RMSE: 0.474994630647171

print(kreg100.fit)
```

```
## [1] Kernel Regression
## [1] N: 6820
## [1] Bandwidth: 100
## [1] RMSE: 0.477791610403467
```

We can now visually compare the kernel regressions with different bandwidths (using the pch="" option to eliminate the plotting symbols for the raw data):

```
par(mfrow=c(2,2))
plot(kreg2.fit,ylim=c(0.5,0.8),type="l",xlab="Survey Year",ylab="")
plot(kreg5.fit,ylim=c(0.5,0.8),type="l",xlab="Survey Year",ylab="")
plot(kreg10.fit,ylim=c(0.5,0.8),type="l",xlab="Survey Year",ylab="")
plot(kreg100.fit,ylim=c(0.5,0.8),type="l",xlab="Survey Year",ylab="")
```



Homework

R Programming for the Kernel Regression

1. Create a `fitted` method and a `residuals` method for the `kreg` class. These are both single line functions.

2. Create a `lines` method for the `kreg` class that works similarly to the `plot` method, but instead of creating a new plot it adds lines to an existing plot. Use the function to create a single plot overlaying all four of the kernel regression fits plotted that I plotted in a 2x2 grid above.

3. Add an option to your kernel regression function to use a normal kernel rather than a uniform kernel.

   - In order to get the kernel weights for the normal kernel, you should use the `dnorm()` function. Look at the documentation for the function by running the command `?dnorm` in the R console. You might also look at the sequences of values output by the commands `dunif(seq(-4,4,0.1),-1,1)` and `dnorm(seq(-4,4,0.1),0,1)`, and then see what happens when you change the parameters of those commands.
   - We have set up the uniform kernel to have interquartile range equal to the bandwidth (full range equal to two bandwidths). For comparability purposes, set the normal kernel's interquartile range to be equal to the bandwidth, which means that the standard deviation of the normal kernel should be `bw/(2*qnorm(0.75))`.

   Create a plot comparing a kernel regression with uniform kernel and bandwidth of 5 years to one with a normal kernel and bandwidth of 5 years.

4. Modify the kernel regression function so that it still works if some of the elements of `y` are missing. There are several ways to do this, but the important thing is that you should preserve the input ordering of observations. You cannot simply throw away the missing values, you must maintain them through the calculation so that the output fitted values have the same length and order of observation as the inputs.

   - Functions like `sum` and `mean` in R usually have a `,na.rm=TRUE` option that omits missing values.
   - To test your function, I suggest replacing some of the elements of the *y* vector using the following series of commands:

```r
# create a new data frame to mess with
data4 <- data3

# decimate the outcome variable with missing data
data4$wrong[seq(1,nrow(data4),10)] <- NA
```

Applying Kernel Regression

5. If you calculated a uniform kernel regression with a bandwidth of less than one year on these day, what would the estimates correspond to?

6. Explain why the RMSEs get smaller with decreasing bandwidth, and why this means that we cannot use RMSE for choosing the bandwidth.

7. Generate a plot of a kernel regression for the cohort of individuals born between 1956 and 1964, with normal bandwidth equal to 5 years (use the uniform if you could not get the normal working). Describe similarities and differences versus the 1946 to 1955 cohort.

8. Choosing among the kernels that you have programmed (uniform and normal) and any bandwidths that you might try, which one looks to you like the best description of the trends in attitudes among the 1946 to 1955 cohort? (There are many plausible answers to this question!)