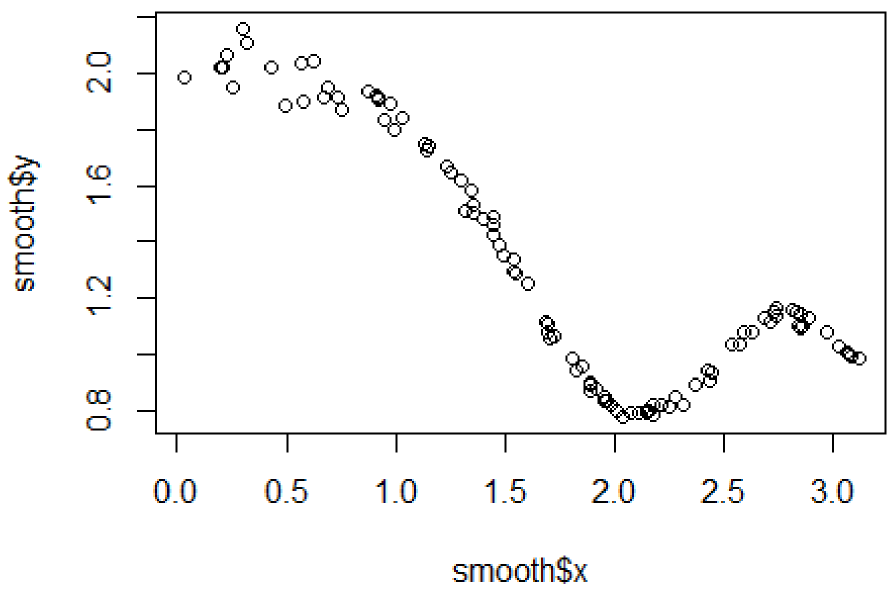
# Yichen Dong Module 12 HW

Yichen Dong

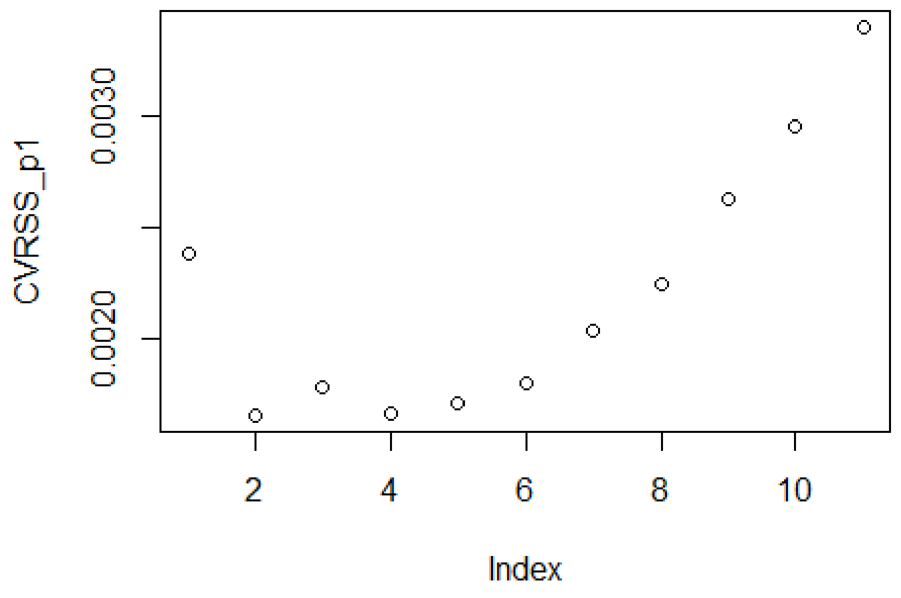
November 29, 2018

## Problem 1

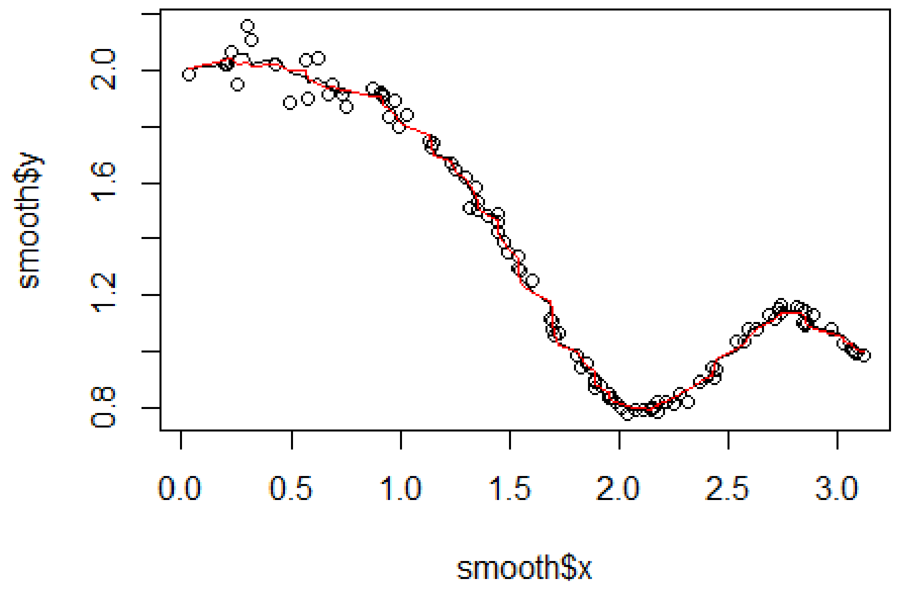
```
library(dplyr)

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

x = scan('smooth_x.txt')
y = scan('smooth_y.txt')
smooth = as.data.frame(cbind(x,y))%>%
  arrange(x)
plot(smooth$x,smooth$y)
```



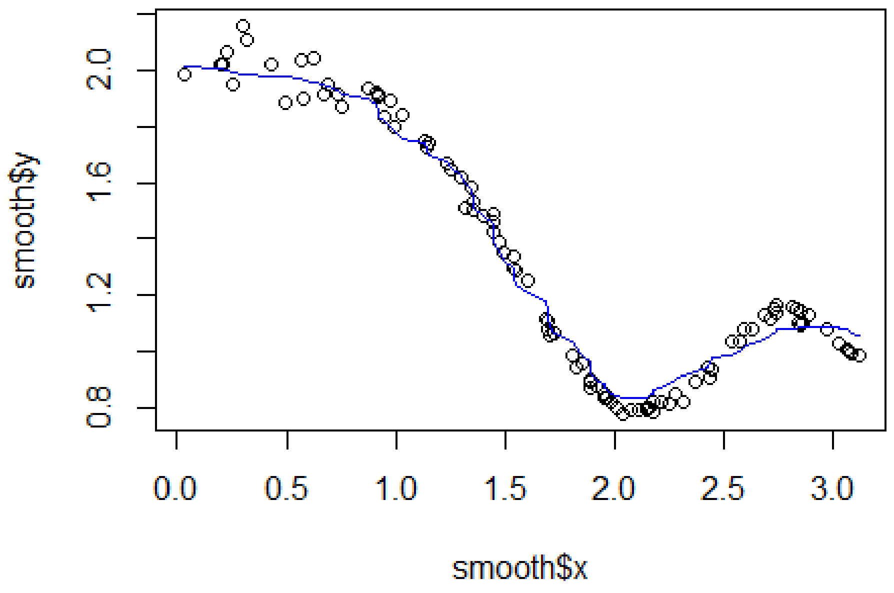
```r
n = length(smooth$x)
k_end = 11
S_k = NULL

CVRSS_p1 = vector("numeric",length = k_end)

for(k in 1:k_end){
  S = matrix(data = 0, nrow = n,ncol = n)
  span = 2*k+1
  b = (span-1)/2
  for(i in 1:n){

    if(i-b <= 0){
      left = i-1
    } else{
      left = b
    }

    if(i+b>n){
      right = 100-i
    } else{
      right = b
    }
    truncated_k = left+right+1
    S[i,(i-left):(i+right)] = 1/truncated_k
    S_k[i] = sum(smooth$y * S[i,])
  }
  smooth[paste("k_",k, sep = '')] = S_k
  for(i in 1:n){
    CVRSS_p1[k] = CVRSS_p1[k] + 1/n *((smooth$y[i]-S_k[i])/(1-S[i,i]))^2
  }
}
plot(CVRSS_p1)
```



```
plot(smooth$x,smooth$y)
lines(smooth$x,smooth$k_2,type = 'l',col='black')
lines(smooth$x,smooth$k_4,type = 'l',col='red')
```



```r
plot(smooth$x,smooth$y)
lines(smooth$x,smooth$k_11,type = 'l',col='blue')
```



It looks like 2 and 4 have the lowest CVRSS. It seems that both 2 and 4 perform really well in the middle part of the data, but they are both not as good in the sparser early portion of the data. However, 2 has a slightly lower cVRSS, so that's the one that I will select.

Also, just for interest, I tried to see what the highest CVRSS, a k of 11, would give us. We can see that is significantly undershoots the dips and rises when x>2.

## Problem 2

### Part a.

I'm not exactly sure how to say this using a mathematical formula, but the idea is this:

- For a span k, calculate b = (k-1)/2
- For each x_i, repeat a process similar to problem, where if i - b <=0, the length of the left side is i-1, else it is b. If i+b > n, then the right length is n-i, else it's b
- retrieve y_i for x[(i-left):(i+right)]
- order the y_is by value, then take the (left+right+1)/2th value. This is the median. In case the previous value is not an integer, take the average of the two values around it.
- Set that value to be s_k for x_i.

- Repeat for each x_i

## Part b

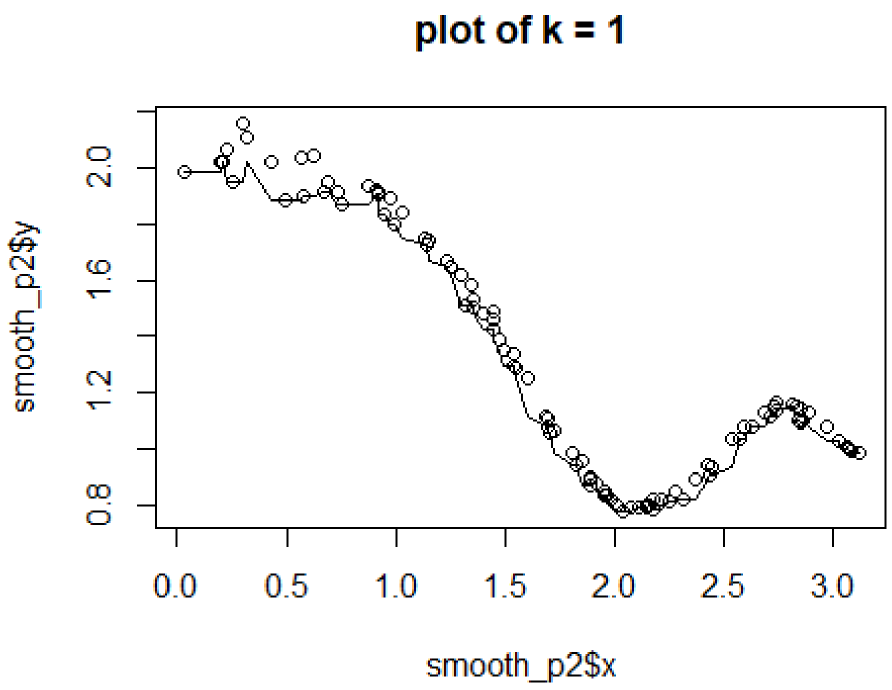
This is not a linear smoother, although an array S can be created for a specific dataset. The reason that it's not a linear smoother despite being able to apply an nxn matrix S to the y values is because the matrix S depends on the values of Y. For example, in the constant span running mean, if we change y_i to y_i*100, the matrix itself would not change. However, if we did that for the constant span running median, it might shift the median if y_i was on the lower half of the span.

If we were to create a matrix S, it would have all 0s in each row except for the one value that's a median, which would have a 1 in that cell.

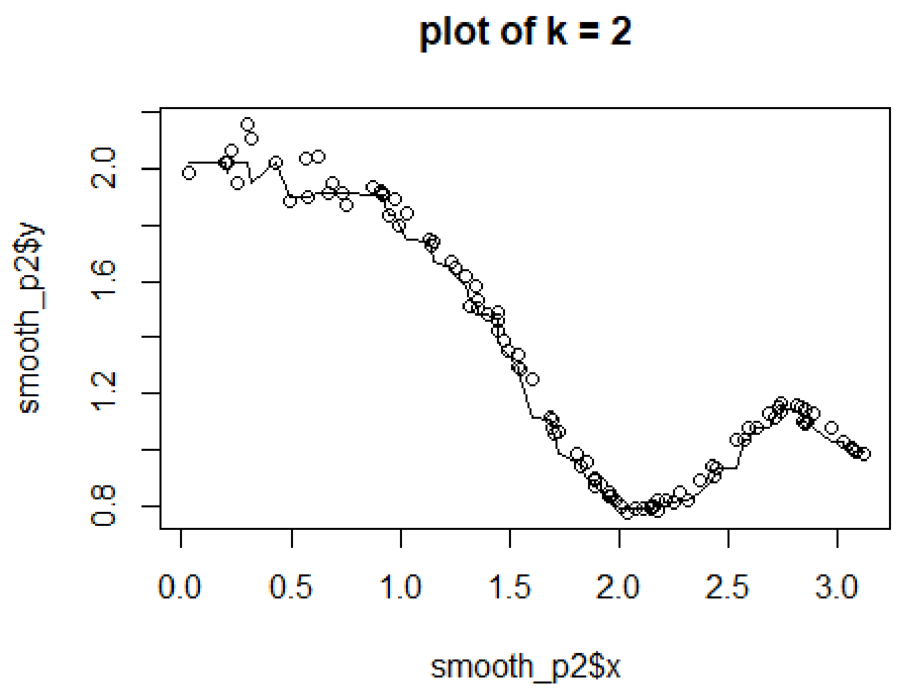
## Part c

```r
smooth_p2 = as.data.frame(cbind(x,y))%>%
  arrange(x)
n = length(smooth_p2$x)
k_end = 11
S_k_med = NULL

for(k in 1:k_end){
  span = 2*k+1
  b = (span-1)/2
  for(i in 1:n){

    if(i-b <= 0){
      left = i-1
    } else{
      left = b
    }

    if(i+b>n){
      right = 100-i
    } else{
      right = b
    }
    cs_med = smooth_p2$y[(i-left):(i+right)]
    cs_med = sort(cs_med)
    if ((span-1)/2%%1 == 0){#Since R does not have a function to check for an
integer, this checks to see if a decimal is returned
      S_k_med[i] = cs_med[(span-1)/2]
    }else{
      S_k_med[i] = (cs_med[floor((span-1)/2)] + cs_med[ceiling((span-1)/2)])/
2
    }
  }
  smooth_p2[paste("k_",k, sep = '')] = S_k_med
}
```

```
plot(smooth_p2$x,smooth_p2$y)
lines(smooth_p2$x,smooth_p2$k_1)
title(main = "plot of k = 1")
```

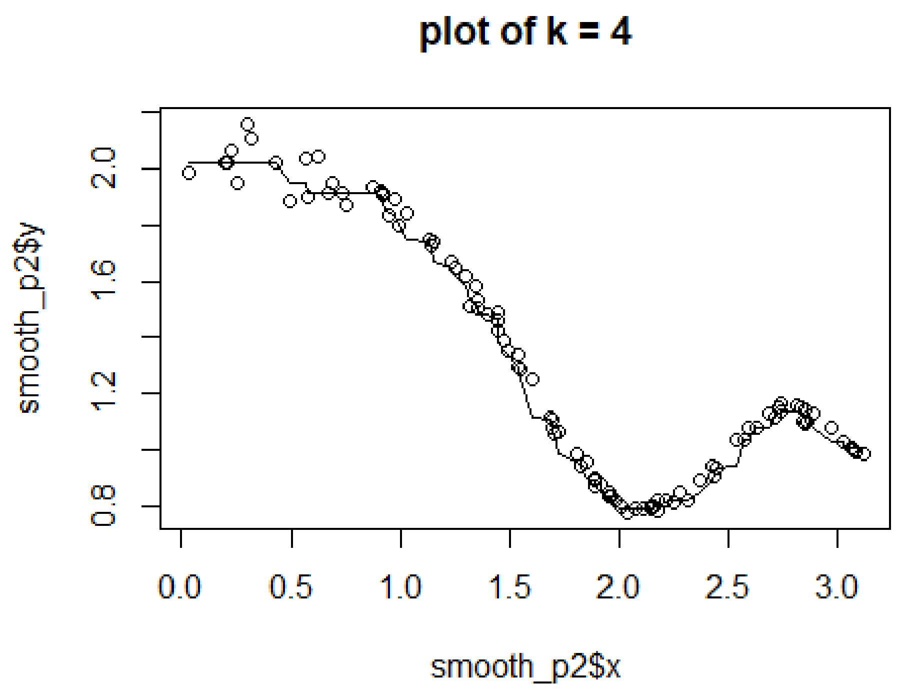


```
plot(smooth_p2$x,smooth_p2$y)
lines(smooth_p2$x,smooth_p2$k_2)
title(main = "plot of k = 2")
```

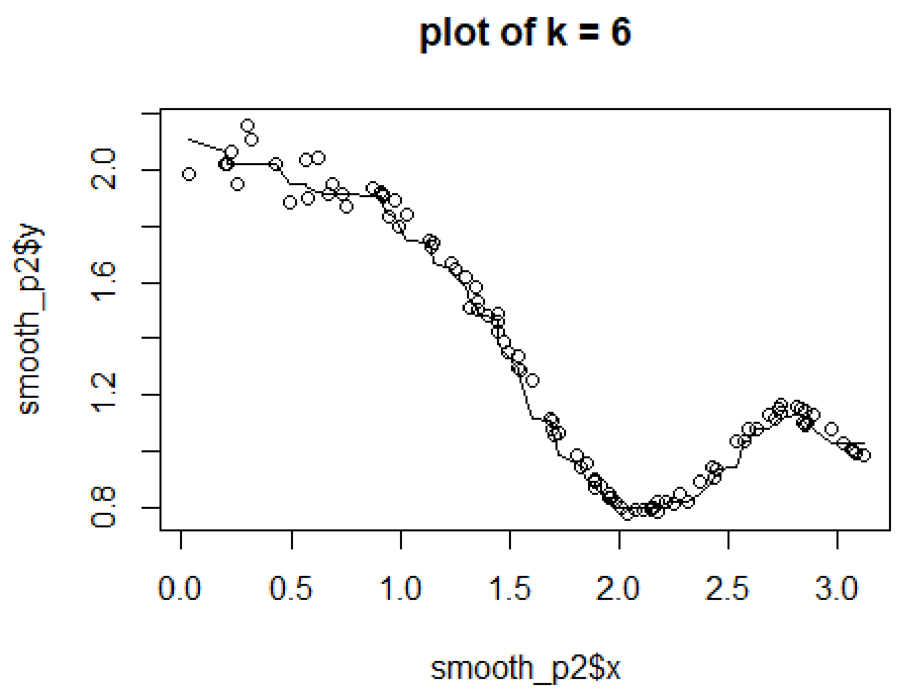


We can see that the plots for k=1 and 2 appear to be too wiggly, and does not track that well in the earlier sections. It seems that the optimal span is not the same.

```
plot(smooth_p2$x,smooth_p2$y)
lines(smooth_p2$x,smooth_p2$k_4)
title(main = "plot of k = 4")
```

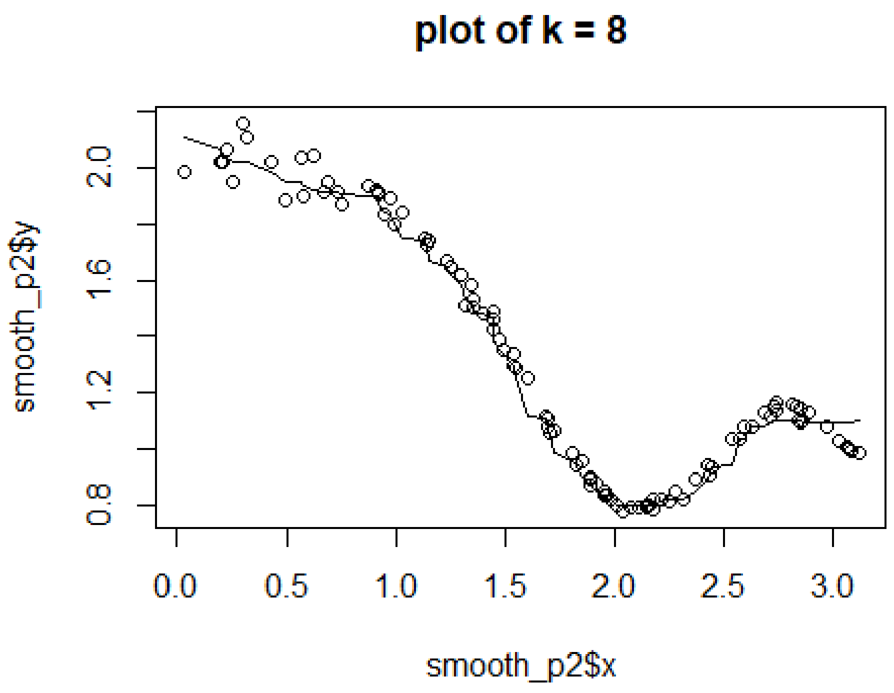


```
plot(smooth_p2$x,smooth_p2$y)
lines(smooth_p2$x,smooth_p2$k_6)
title(main = "plot of k = 6")
```

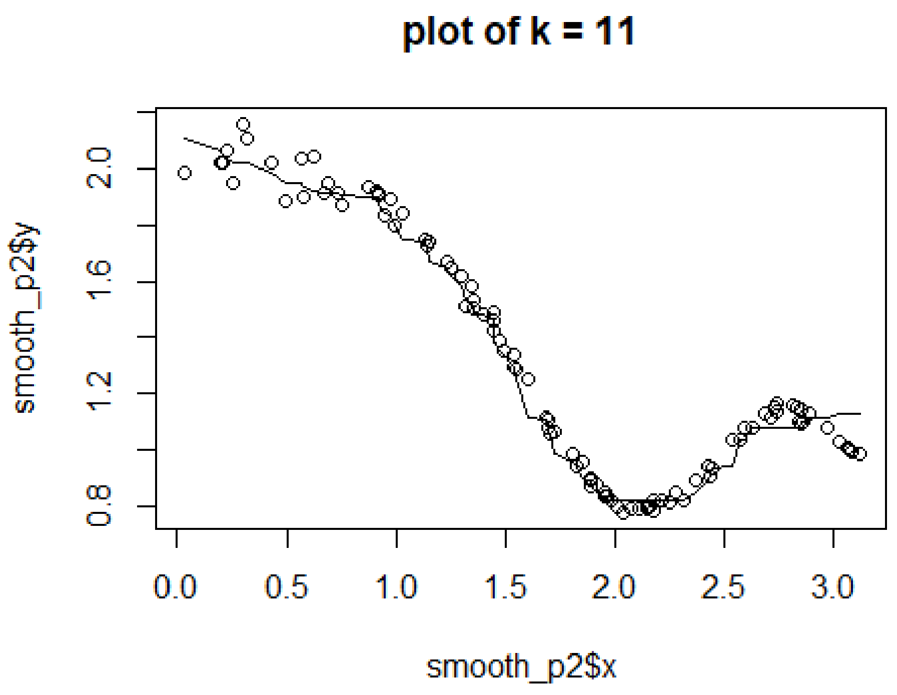


These seem to be the best choices.It tracks well towards the middle and the end of the graph, and for 6, seems to be able to pick up a little of the upward trend towards the beginning of the data. Neither is perfect, as 6 has the rise at the end, while 4 has significant plateaus towards the beginning

```
plot(smooth_p2$x,smooth_p2$y)
lines(smooth_p2$x,smooth_p2$k_8)
title(main = "plot of k = 8")
```



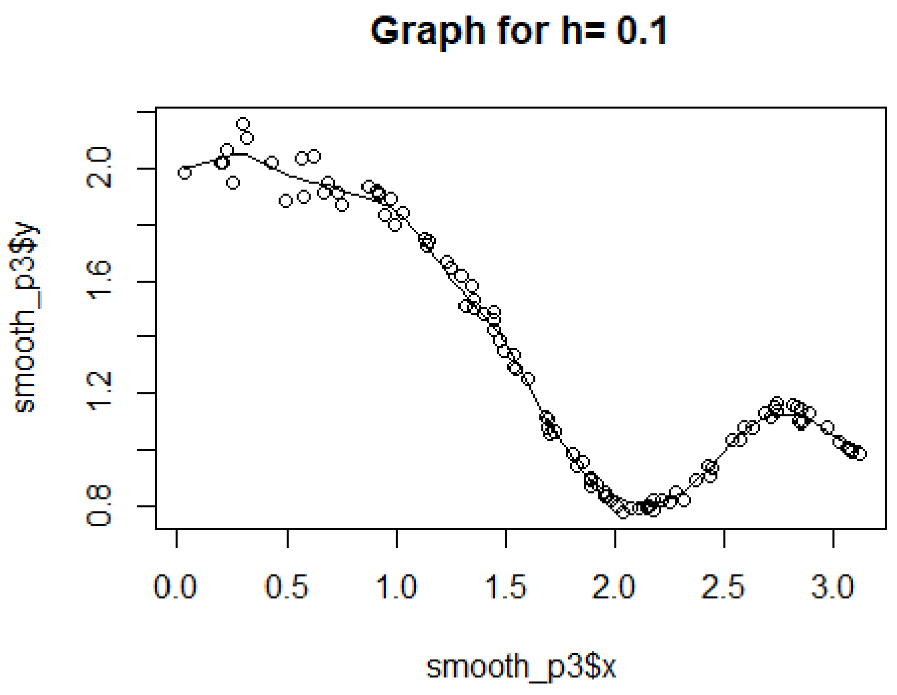
```
plot(smooth_p2$x,smooth_p2$y)
lines(smooth_p2$x,smooth_p2$k_11)
title(main = "plot of k = 11")
```

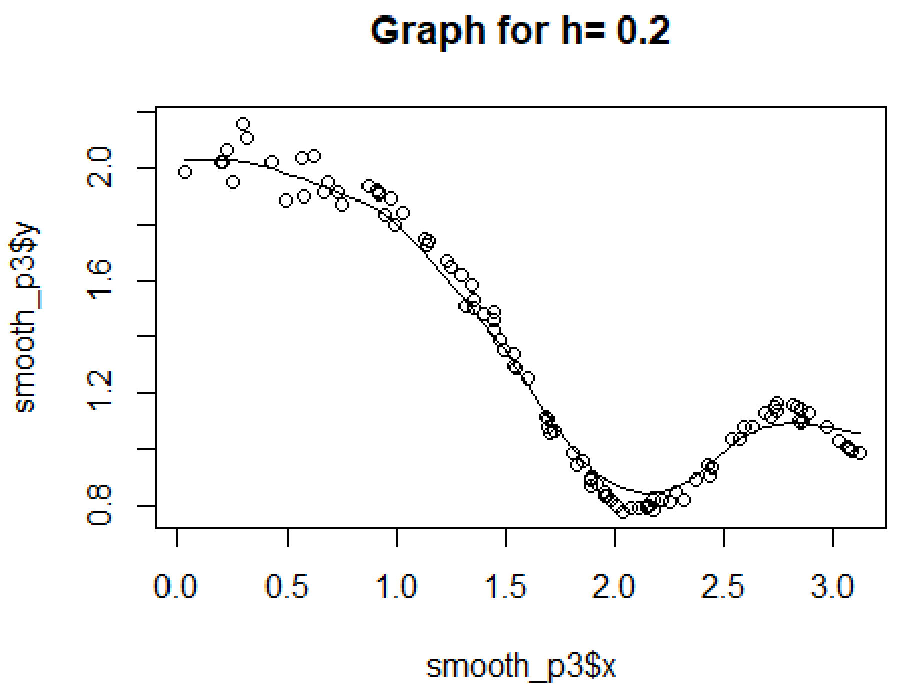


These two seem to be doing much better in the beginning of the graph, but missed the dip at the end entirely. I would say a k of 6 is probably the best choice, or maybe the median is just not a good way to represent this data. Or take a larger k at the firest half of the graph and a smaller k for the second.

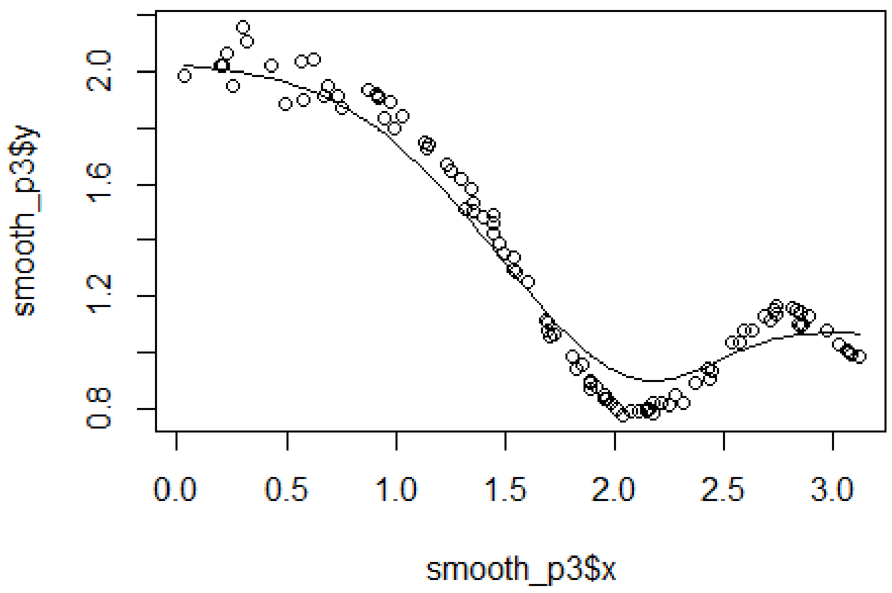
## Problem 3

```
smooth_p3 = as.data.frame(cbind(x,y))%>%
  arrange(x)
h_array = seq(.1,1.1, by =.1)
n = length(smooth_p3$x)

K_z = function(z){
  1/sqrt(2*pi)*exp(-z^2/2)
}

for(h in h_array){
  S = matrix(data = 0, nrow = n,ncol = n)
  s_k_norm = NULL
  for(i in 1:n){
    for(j in 1:n){
      S[i,j] = K_z((smooth_p3$x[i] - smooth_p3$x[j])/h)
    }
    sum_i = sum(S[i,])
    s_k_norm[i] = 1/sum_i * sum(smooth_p3$y * S[i,])
  }
  smooth_p3[paste("h_",h, sep = '')] = s_k_norm
  plot(smooth_p3$x,smooth_p3$y)
  lines(smooth_p3$x, s_k_norm)
  title(main = paste("Graph for h=" , h))
}
```

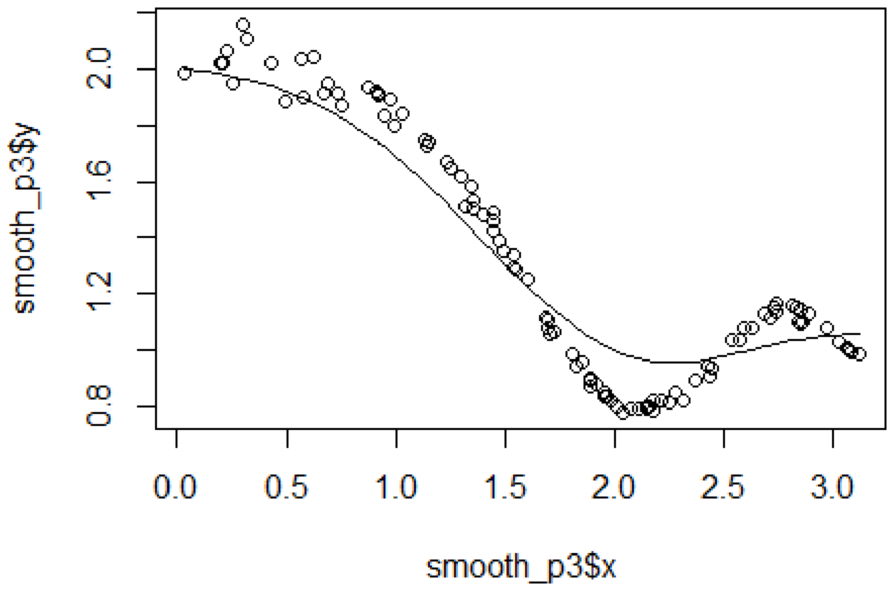


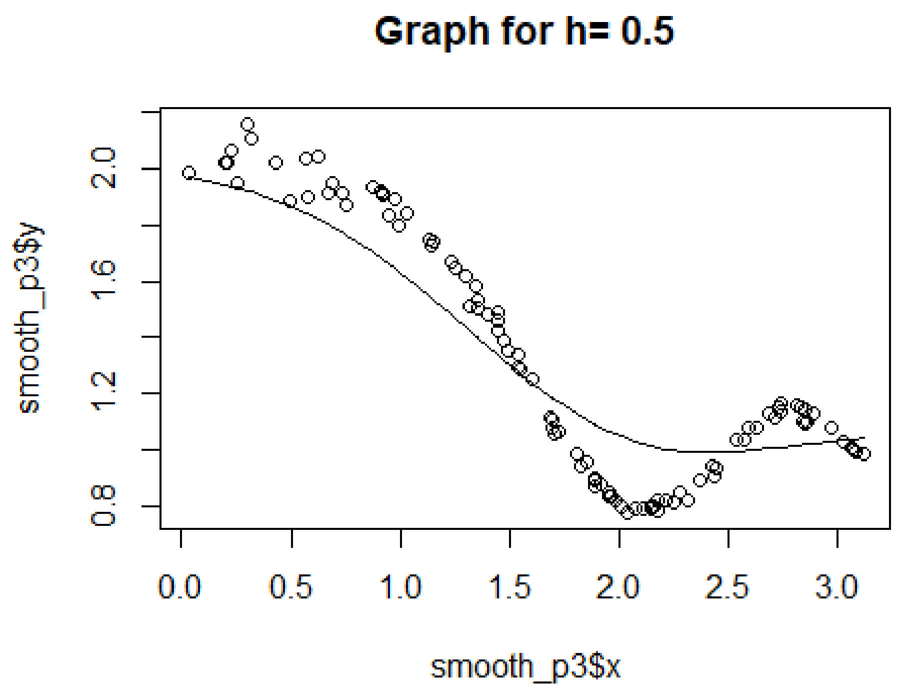


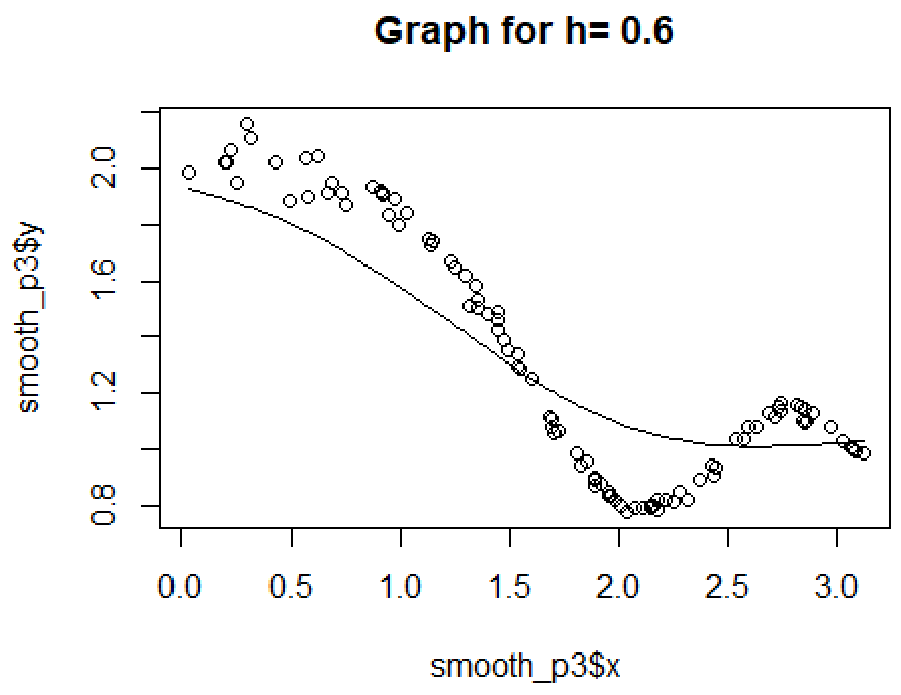
## Graph for h= 0.3

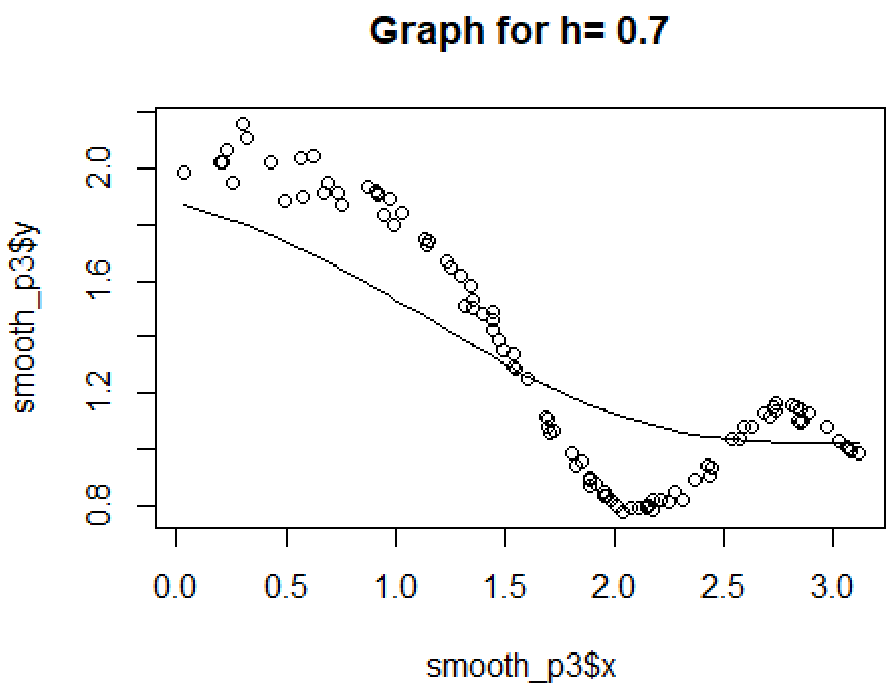


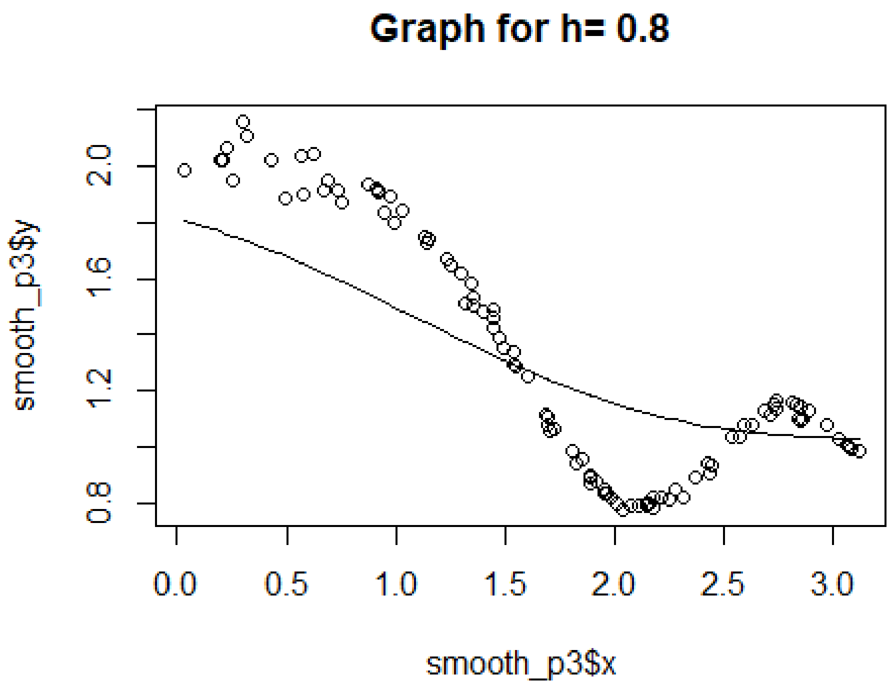
## Graph for h= 0.4



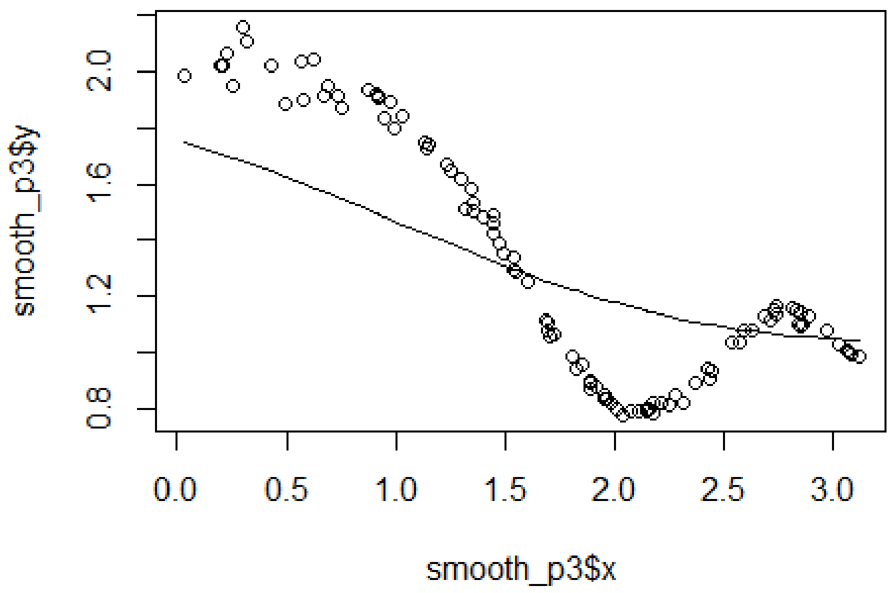

Graph for h= 0.5
smooth_p3$y
smooth_p3$x
0.8
1.2
1.6
2.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0


Graph for h= 0.6
smooth_p3$y
smooth_p3$x
0.8
1.2
1.6
2.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0

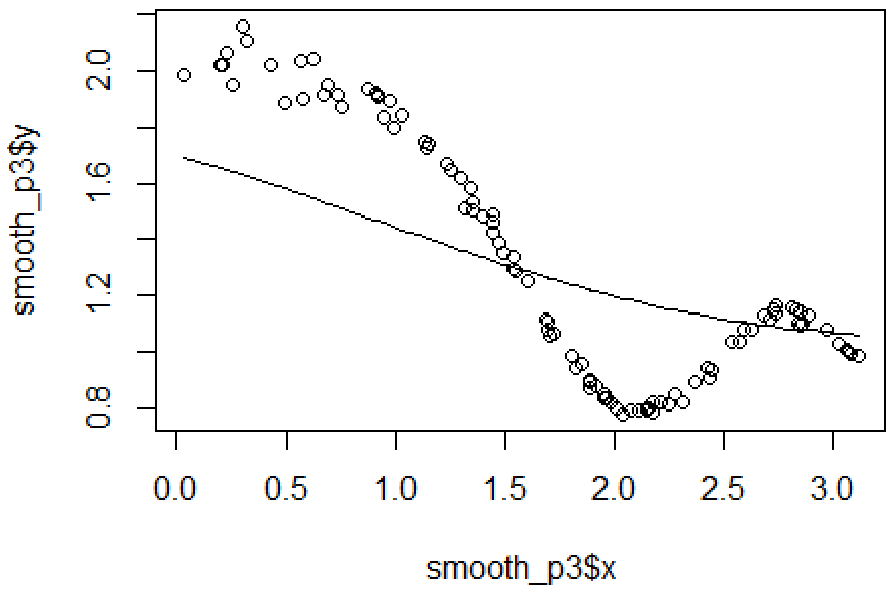

Graph for h= 0.7
smooth_p3$y
smooth_p3$x
0.8
1.2
1.6
2.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0

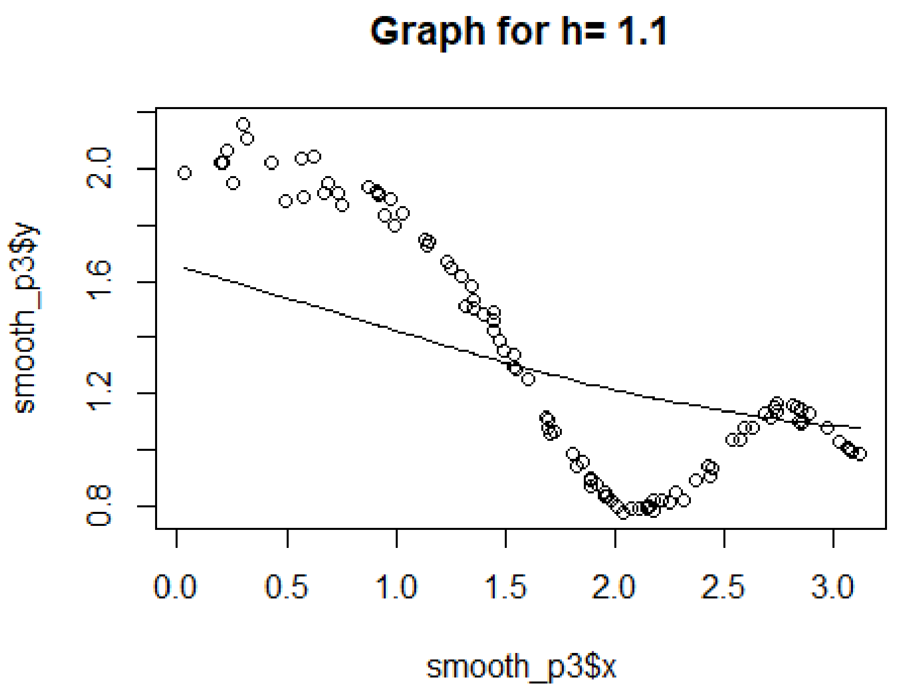

Graph for h= 0.8
smooth_p3$y
smooth_p3$x
0.8
1.2
1.6
2.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0

## Graph for h= 0.9



## Graph for h= 1





It seems that h=0.1 or h=0.2 performs the best. I would say .1 would be my choice, since it tracks the movement of the last part of the graph really well, and does decently in the beginning even if it's a little wiggly. h=.2 doesn't seem to track the valley and peak as well in the second half. Anything larger than that seems to perform increasingly worse, as it seems to miss the entire shape of the curve and become a straight line towards h=1.1

```
for(h in c(.1)){
  S = matrix(data = 0, nrow = n,ncol = n)
  s_k_norm = NULL
  for(i in 1:n){
    for(j in 1:n){
      S[i,j] = K_z((smooth_p3$x[i] - smooth_p3$x[j])/h)
    }
    sum_i = sum(S[i,])
    s_k_norm[i] = 1/sum_i * sum(smooth_p3$y * S[i,])
  }
}
S[30,]
```

```
##   [1] 6.089978e-36 2.420153e-27 1.118474e-26 1.041138e-25 1.365124e-24
##   [6] 9.968226e-23 6.995307e-22 2.869417e-17 4.343236e-15 1.048899e-12
##  [11] 1.940744e-12 8.732879e-11 1.296266e-09 4.252016e-09 5.791164e-08
##  [16] 2.151064e-07 6.182368e-05 2.066486e-04 3.209223e-04 3.945111e-04
##  [21] 8.492025e-04 2.357325e-03 4.767903e-03 1.274223e-02 1.056404e-01
##  [26] 1.219996e-01 1.414213e-01 3.222912e-01 3.663155e-01 3.989423e-01
##  [31] 3.873171e-01 3.534147e-01 3.385822e-01 3.277421e-01 2.261078e-01
##  [36] 1.288212e-01 1.263725e-01 1.239807e-01 8.066967e-02 5.640596e-02
##  [41] 2.056610e-02 1.972100e-02 1.457970e-02 3.452453e-03 1.708792e-04
##  [46] 1.531471e-04 1.128287e-04 1.080682e-04 4.124088e-05 1.055117e-06
##  [51] 3.950430e-07 8.112348e-08 9.222444e-09 9.135885e-09 6.666849e-09
##  [56] 1.818752e-09 1.942763e-10 1.626459e-10 9.704965e-11 2.684098e-11
##  [61] 3.442765e-12 5.161253e-13 3.689643e-14 1.773316e-15 6.205577e-17
##  [66] 5.062835e-17 2.851634e-17 8.346289e-18 4.150497e-18 1.745417e-19
##  [71] 4.670231e-21 5.932449e-22 1.040067e-23 2.375739e-26 1.015096e-28
##  [76] 1.964045e-29 1.153339e-29 9.354779e-35 8.093189e-37 1.259329e-37
##  [81] 1.661557e-39 8.879457e-43 1.785895e-44 6.952579e-46 2.656785e-46
##  [86] 1.367717e-46 4.085682e-51 1.790237e-52 5.053878e-53 1.605219e-53
##  [91] 1.544072e-53 5.467628e-54 3.897170e-56 4.023701e-62 4.856506e-66
##  [96] 2.191186e-66 5.266350e-69 1.684909e-69 2.007399e-70 3.572916e-73
```

Not sure which row you wanted, so I just picked the thirtieth one.