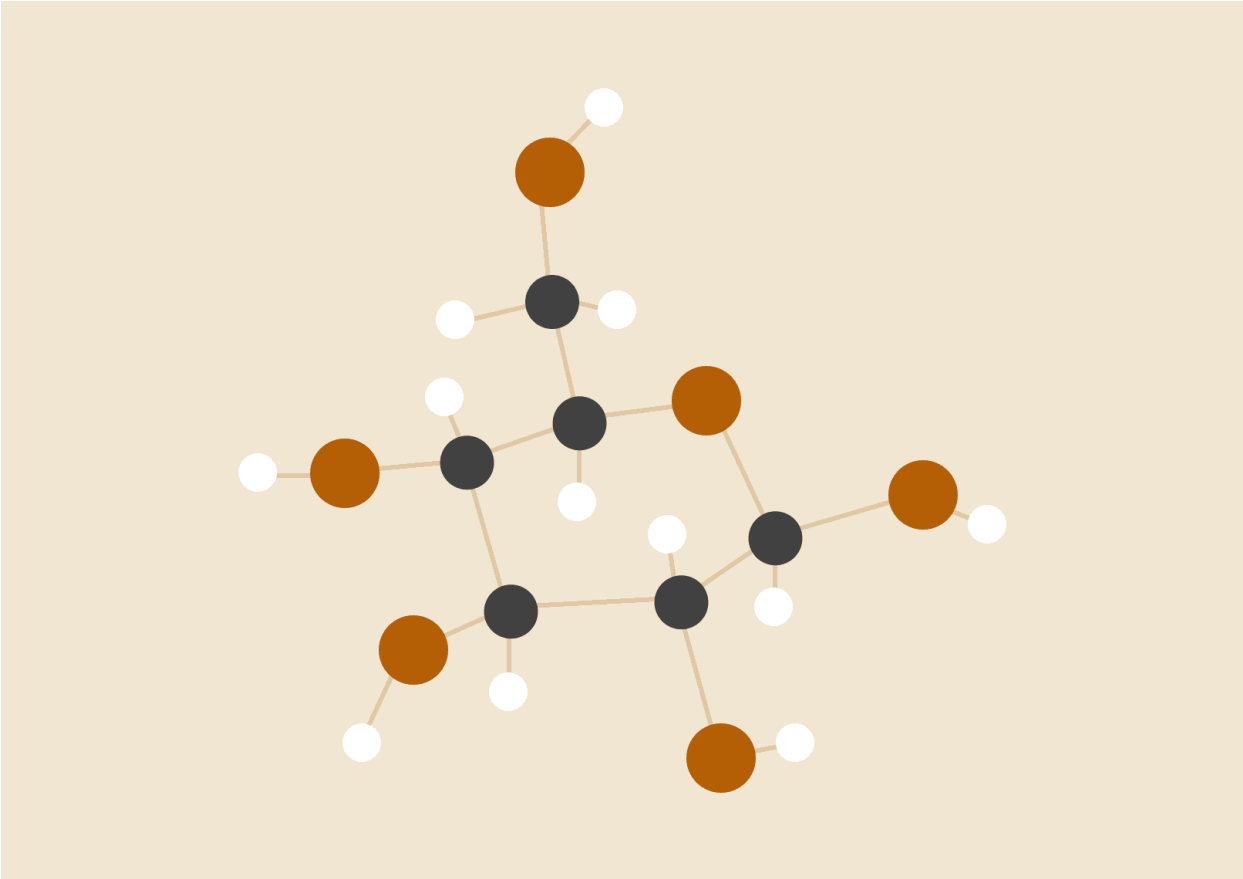
Salters and Smoothers



Jack Stewart

12.10.2024
Probability and Applied Statistics

PSS 2: Octave

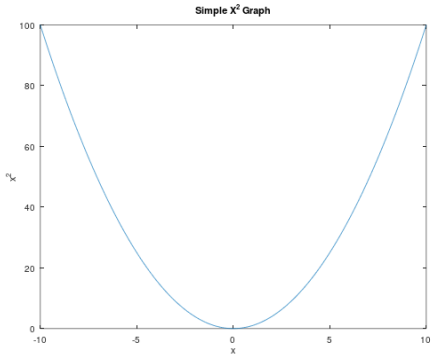
The Setup

When Installed and first started trying to work in Octave, I felt really lost but with a bit of time it ended up feeling decently familiar. I started out this process by just trying to write y=x^2 in Octave and plot it, as I figured that would be a good starting point.

Graphing y = x^2

The first step in this process was to program the function y=x^2, as this is going to be our base for our salters and smoothers. To graph this program I looked through the section of Octave's documentation on graphing and found the basic setup to graph a function in 2 variables in octave, and modified it to suit the purpose of graphing the function y=x^2. That code, along with the graph, is below.

```
x = -10:0.1:10;
y = x.^2
plot (x, y);
xlabel ("x");
ylabel ("x^2");
title ("Simple X^2 Graph");
```

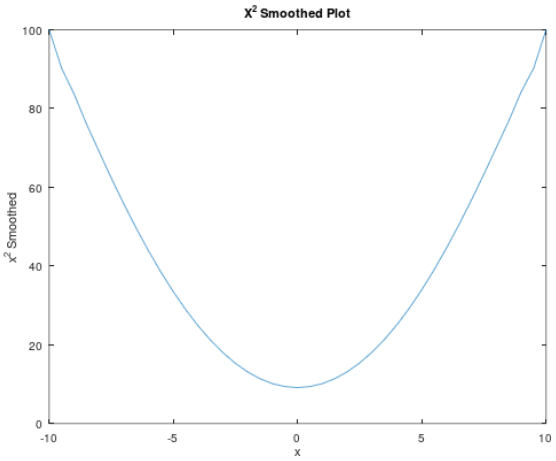


This code creates two matrices, the first one defined to be going from -10 to 10 via increments of .1, making a 1x201 matrix which is used as our x values. The second is directly based on the first, taking each value and squaring it for the respective value.

Making the Smoother

Upon beginning work on the smoother, I immediately struggled to understand the syntax and language of Octave. It felt foreign to me, and I couldn't seem to understand how to do even as simple a task as y(x) = y(x+1). After a lot of experimentation I found a way to loop over the matrix that made the change I wanted which is shown below along with its resulting graph. I ran this 10 times over because simply one didn't have a sufficiently noticeable effect on the graph.

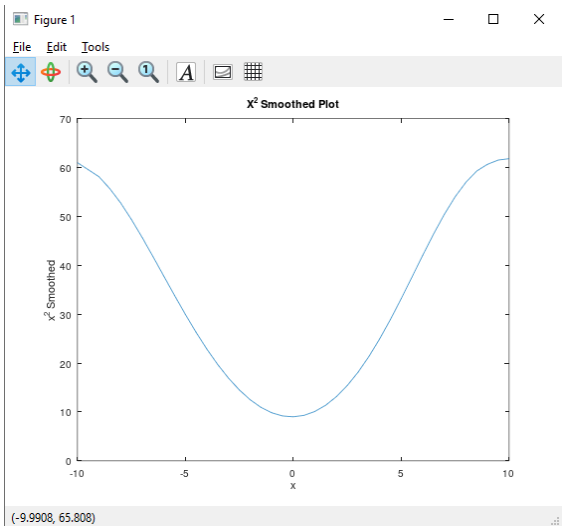
```
x = -10:0.5:10;
y = x.^2

for flattenCount = 0:10
  for index = 3:39
    y(:,index) = mean(y(:,index-2:index+2))
  endfor
endfor

plot (x, y);
xlabel ("x");
ylabel ("x^2 Smoothed");
title ("X^2 Smoothed Plot");
```



For this plot I increased the increment between values to .5 to speed up computation time, but it still didn't work as well as I wished due to the ends not being accessible, or I would get an outOfIndex Error. This resulted in the strange end behavior of the graphs where it suddenly skews

To fix this I added 2 more loops to handle each end specifically, which resulted in a much better result, which really surprised me with it's shape. The code and image are below.

```
x = -10:0.5:10;
y = x.^2

for flattenCount = 0:10
  for index = 1:2
    y(:,index) = mean(y(:,1:index+2))
  endfor

  for index = 3:39
    y(:,index) = mean(y(:,index-2:index+2))
  endfor

  for index = 39:41
    y(:,index) = mean(y(:,index-2:41))
  endfor
endfor

plot (x, y);
xlabel ("x");
ylabel ("x^2 Smoothed");
title ("X^2 Smoothed Plot");
```

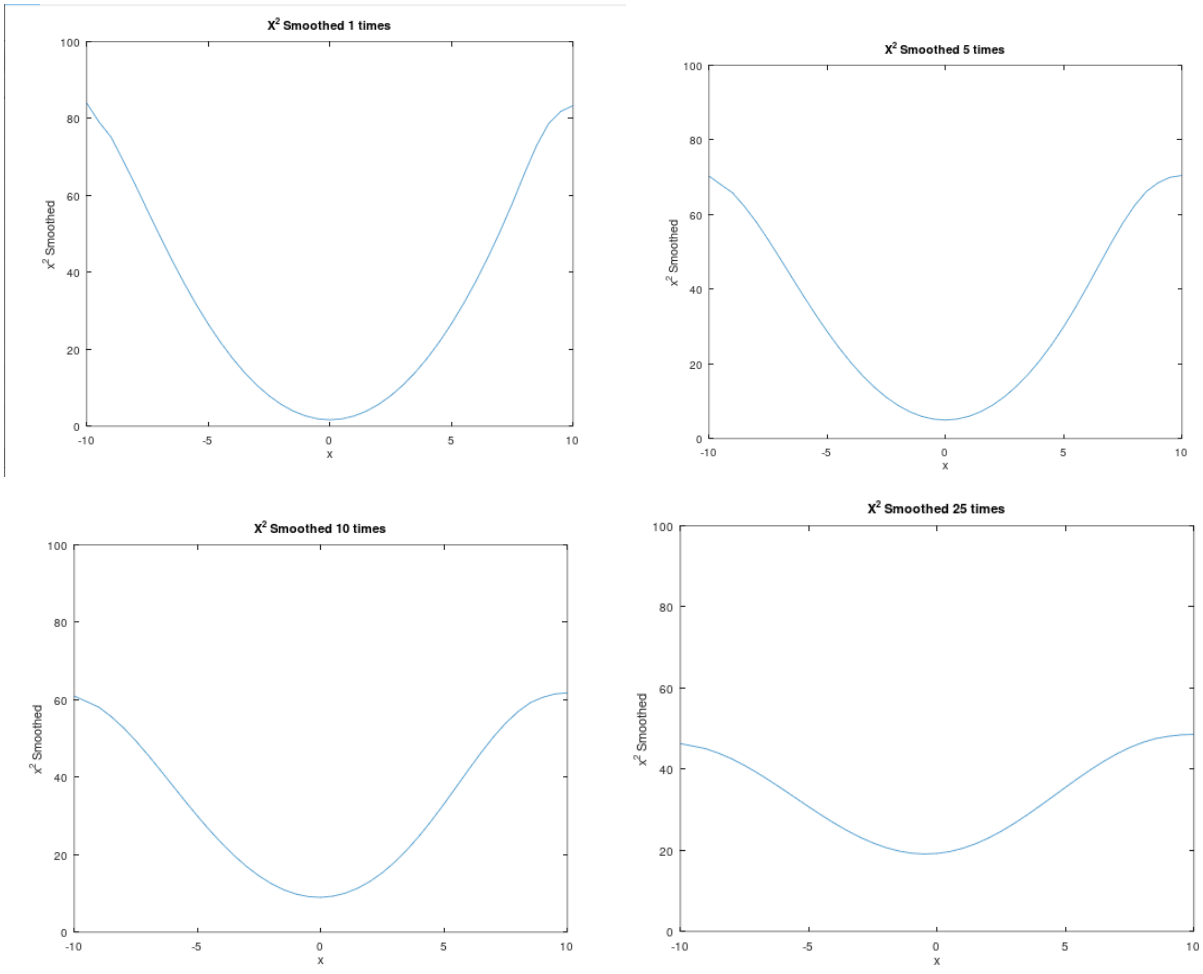


After some analysis of the shape I deemed that this is right, and the reason the ends are more harshly affected that the other regions is because at the extreme endpoints, say the first index, the only available values are far lower than the first index itself whereas the

second index has the first index to help balance out the change. This code ended up being my final code, aside from some slight modifications to allow for easier control of how many times the graph was smoothed.

Changing The Conditions

Due to the fact that I had to run the smoothing function multiple times for a visibly different result, I wanted to compare the graphs of the same function but smoothed differing numbers of times. I changed the code up slightly to make it simpler to allow for the changing of the number of times the smoothing function was run. In the process of doing this I also had to learn how to manually set the values of the axis, which was tricker than I expected. The graphs below are for 1 (top left), 5 (top right), 10 (bottom left), and 25 (bottom right) smoothings.



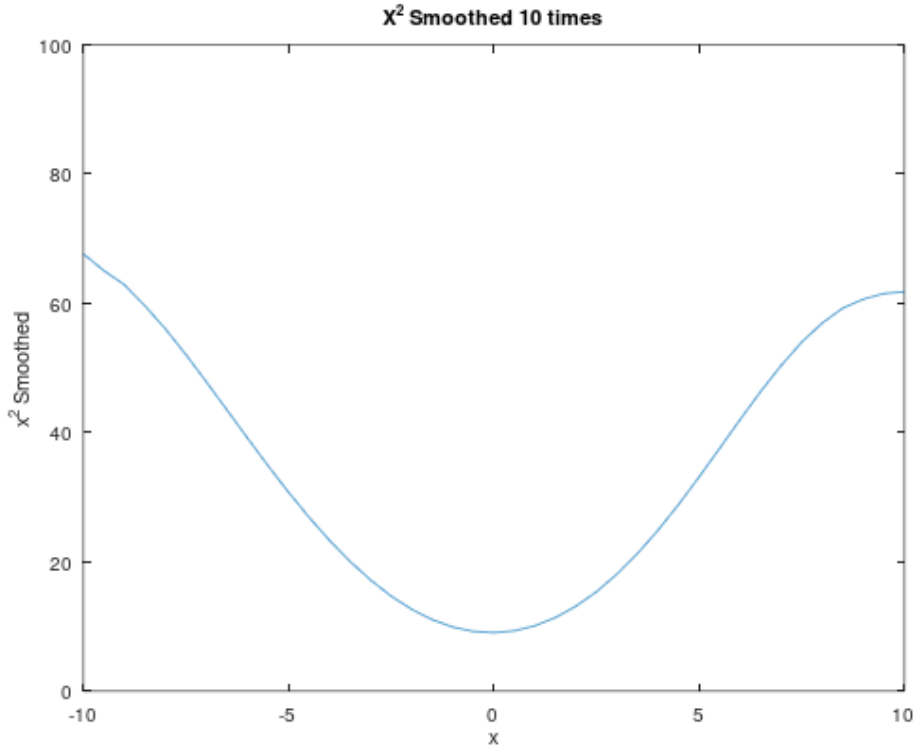
Conclusions about the Smoother

While it's not terribly surprising, the smoother seems to be approximately approaching a limit of a horizontal line, given the function was smoother enough at that line appears to be somewhere around y=35. The effects on the edges are drastically more pronounced, and the least affected areas were around the center because the slope was the lowest so the change in value was the least pronounced on each iteration. I think that the issues faced near the edges could be very problematic for the use of this type of function as an actual smoother, because after being strongly smoothed the function itself is nearly unrecognizable, due to the massively over smoothed edges, as the function more closely resembles the graph y=-cos(x) than y=x^2.

Potential Solution to the Problems with the Smoother (Extra Credit?)

One potential way to fix this would be to average the values near the edges with less overall values, or add the value of the point itself into the summation once for every missing value. Both of these would cause it to be pulled down less at the extreme ends. For example the second strategy would make the flattening equation for the first index = 3 * 10^2 + 9.5 ^2 + 9.8^2, which would overall reduce the averaging effect. Below I modified the code to take this approach on the left side of the graph specifically for a value of 10 smoothings and the difference is pretty apparent.

```
for smoothCounter = 0:smoothCount
  for index = 1:2
    y(:,index) = mean([y(:,1:index+2) y(:,index) * ones(1, 3-index)])
  endfor

  for index = 3:39
    y(:,index) = mean(y(:,index-2:index+2))
  endfor

  for index = 39:41
    y(:,index) = mean(y(:,index-2:41))
  endfor
endfor
```

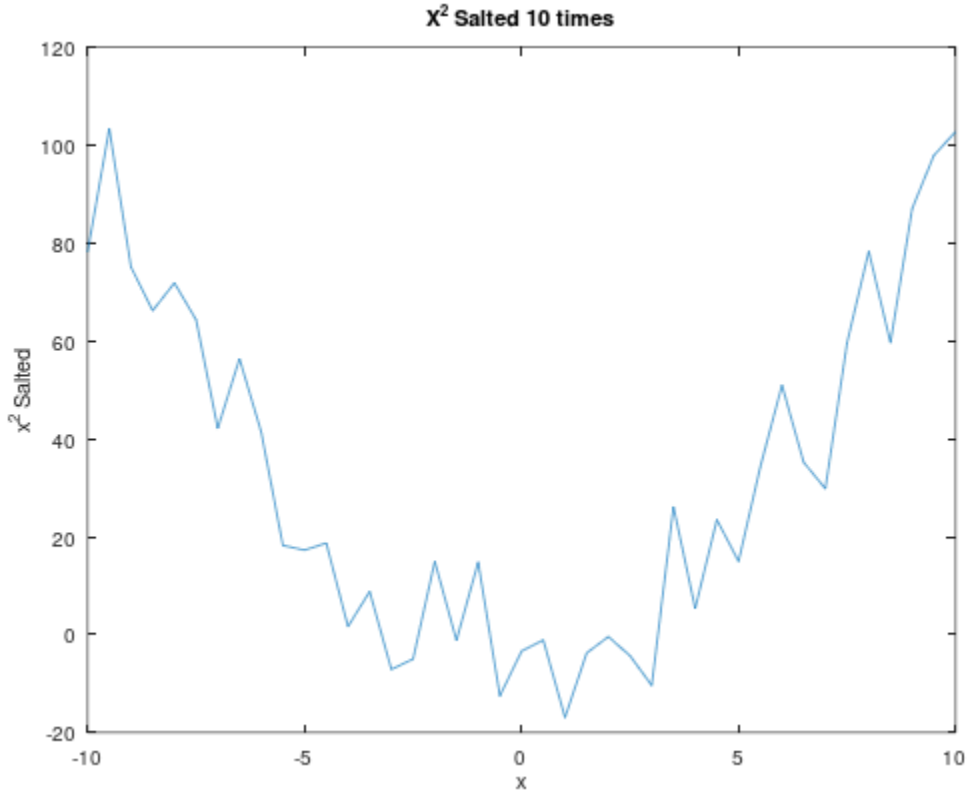


I think the smoothing function on the left, which is the modified algorithm, is a more effective way of smoothing because it smooths the function while still retaining more of the original idea of the function, and not creating as drastic of a negative curvature as the smoothing function on the right did.

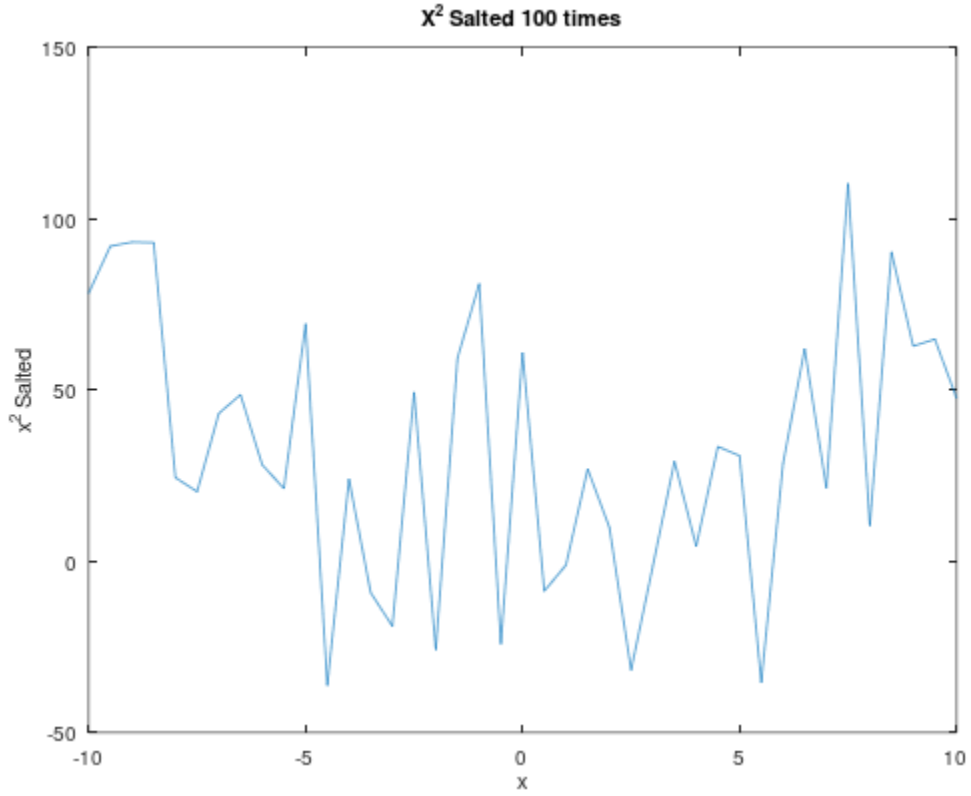
Salting y=x^2

The process of creating the salter was massively more simple than making the smoother. I began with the same code as the smoother I just used, and modified the smoothing function to add a random variable. I choose to use a uniform random variable from -5 to 5. I did this by taking the random function and multiplying it by ten (making it have a range of [0, 10]) then subtracting 5. This made a uniform random number generator from -5 to 5. I added this to each index in the function and this was the result along with the corresponding code.

```
x = -10:0.5:10;
y = x.^2
saltCount = 10
for smoothCounter = 0:saltCount
  for index = 1:41
    y(:,index) = y(:,index) + 10 * rand() - 5
  endfor
endfor

plot (x, y);
xlabel ("x");
ylabel ("x^2 Salted");
title ("X^2 Salted 10 times");
```



Once again I allowed the ability of the user to salt their program multiple times, but because of the way that the random number generator worked, being centered on zero, adding more salts increased the variance but not the average values of any of the data points. I thought this was kind of interesting, as I could salt the graph 100 times and still, in all likelihood, end up with a graph that largely resembled the original which I did here.



The average shape of the curve can still be seen, even though it has been so thoroughly salted repeatedly. The salter can also be adjusted based on how drastic the salting should be, by reducing the multiplier applied to the rand function.

Pss 1: Java

Coding the functions via CSV

I started out this task by creating a program to graph the simple function y=x^2. This was pretty easy to do in java via a csv, as I just took an input list that I defined with a for loop for my x values and created the y values off of it. I set it up so that I have a class for each type, a Plotter, Salter, and Smoother. For the smoother I made an adjustable setting for the radius around the value to average together.

Plotter Program:

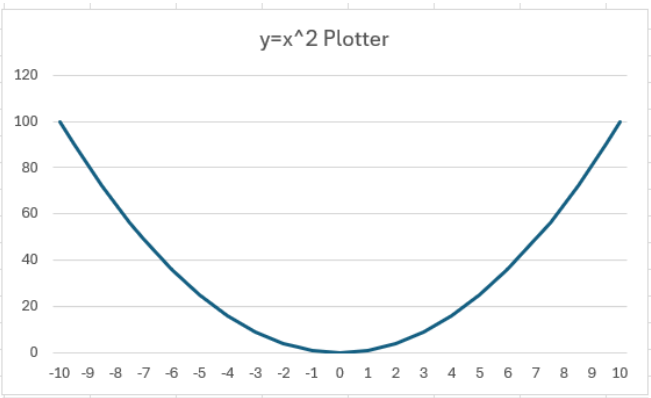
```java
import java.io.FileWriter;
import java.io.IOException;
import java.io.Writer;
import java.util.ArrayList;

public class Smoother
{
    private ArrayList<Double> xValues;
    private ArrayList<Double> yValues;
    private int smoothRadius;

    public Smoother(ArrayList<Double> xInputs, int inputSmoothRadius)
    {
        xValues = xInputs;
        yValues = new ArrayList<Double>();
        smoothRadius = inputSmoothRadius;

        //Handes values that would result in IndexOutOfRange Error on negative side
        for(int index = 0; index < smoothRadius; index++)
        {
            double sum = 0;
            for(int viewingIndex = 0; viewingIndex <= index + smoothRadius; viewingIndex++)
            {
                sum += Math.pow(xValues.get(viewingIndex), b:2);
            }
            yValues.add(sum / (index + 3));
        }

        for(int index = smoothRadius; index < xValues.size() - smoothRadius; index++)
        {
            //Controls the y value based off the x value set inputted
            double sum = 0;
            for(int viewingIndex = index - smoothRadius; viewingIndex <= index + smoothRadius; viewingIndex++)
            {
                sum += Math.pow(xValues.get(viewingIndex), b:2);
            }
            yValues.add(sum / (smoothRadius * 2 + 1));
        }

        for(int index = xValues.size() - smoothRadius; index < xValues.size(); index++)
        {
            //Controls the y value based off the x value set inputted
            double sum = 0;
            for(int viewingIndex = index - smoothRadius; viewingIndex < xValues.size(); viewingIndex++)
            {
                sum += Math.pow(xValues.get(viewingIndex), b:2);
            }
            yValues.add(sum / (xValues.size() - index + 2));
        }
    }

    public void toFile(String filePath) throws IOException
    {
        Writer fileWriter = new FileWriter(filePath);
        for(int index = 0; index < xValues.size(); index++)
        {
            //Writes in the form "x, y\n" for each value pair
            fileWriter.write(xValues.get(index) + ", " + yValues.get(index) + "\n");
        }
        fileWriter.close();
    }
}
```

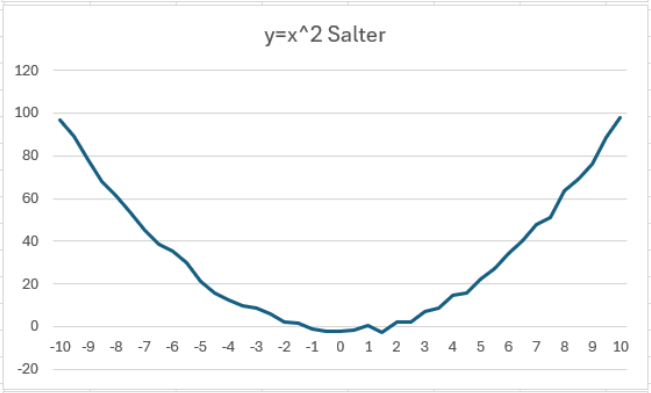
Example CSV for the Plotter:

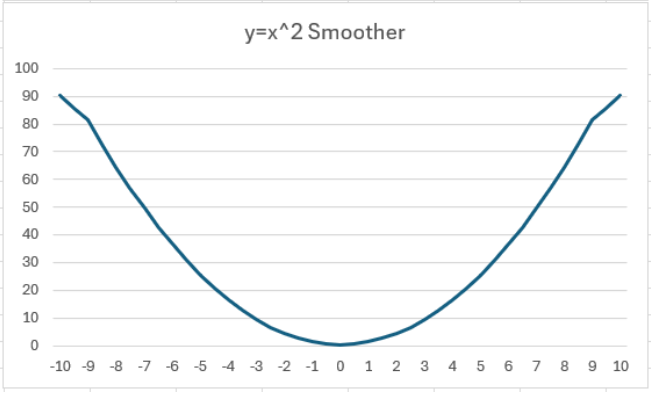
```
-10.0, 90.41666666666667
-9.5, 85.875
-9.0, 81.5
-8.5, 72.75
-8.0, 64.5
-7.5, 56.75
-7.0, 49.5
-6.5, 42.75
-6.0, 36.5
-5.5, 30.75
-5.0, 25.5
-4.5, 20.75
-4.0, 16.5
-3.5, 12.75
-3.0, 9.5
-2.5, 6.75
-2.0, 4.5
-1.5, 2.75
-1.0, 1.5
-0.5, 0.75
0.0, 0.5
0.5, 0.75
1.0, 1.5
1.5, 2.75
2.0, 4.5
2.5, 6.75
3.0, 9.5
3.5, 12.75
4.0, 16.5
4.5, 20.75
5.0, 25.5
5.5, 30.75
6.0, 36.5
6.5, 42.75
7.0, 49.5
7.5, 56.75
8.0, 64.5
8.5, 72.75
9.0, 81.5
9.5, 85.875
10.0, 90.41666666666667
```

I then took this data from the csv and pasted it into excel, which I used to construct the relevant line graphs. From the graphs it can be seen that the Salter slightly roughens the data while the plotter smoother flattens it down slightly, having a more accentuated effect at the tips much like was seen in the previous in Octave.

Plotter X	Plotter Y	Salter X	Salter Y	Smoother	Smoother Y
-10	100	-10	96.8305	-10	90.4167
-9.5	90.25	-9.5	89.2725	-9.5	85.875
-9	81	-9	78.476	-9	81.5
-8.5	72.25	-8.5	67.8672	-8.5	72.75
-8	64	-8	61.6705	-8	64.5
-7.5	56.25	-7.5	53.8206	-7.5	56.75
-7	49	-7	44.8974	-7	49.5
-6.5	42.25	-6.5	38.6415	-6.5	42.75
-6	36	-6	35.3556	-6	36.5
-5.5	30.25	-5.5	29.8613	-5.5	30.75
-5	25	-5	21.2917	-5	25.5
-4.5	20.25	-4.5	15.6366	-4.5	20.75
-4	16	-4	12.5016	-4	16.5
-3.5	12.25	-3.5	9.56359	-3.5	12.75
-3	9	-3	8.49832	-3	9.5
-2.5	6.25	-2.5	6.03371	-2.5	6.75
-2	4	-2	2.28659	-2	4.5
-1.5	2.25	-1.5	1.63321	-1.5	2.75
-1	1	-1	-0.8465	-1	1.5
-0.5	0.25	-0.5	-2.0241	-0.5	0.75
0	0	0	-2.0784	0	0.5
0.5	0.25	0.5	-1.8953	0.5	0.75
1	1	1	0.29721	1	1.5
1.5	2.25	1.5	-2.4342	1.5	2.75
2	4	2	2.22025	2	4.5
2.5	6.25	2.5	2.25615	2.5	6.75
3	9	3	6.98186	3	9.5
3.5	12.25	3.5	8.68223	3.5	12.75
4	16	4	14.6589	4	16.5
4.5	20.25	4.5	15.8154	4.5	20.75
5	25	5	22.5274	5	25.5
5.5	30.25	5.5	26.951	5.5	30.75
6	36	6	34.304	6	36.5
6.5	42.25	6.5	39.9988	6.5	42.75
7	49	7	47.6371	7	49.5
7.5	56.25	7.5	51.2804	7.5	56.75
8	64	8	63.3826	8	64.5
8.5	72.25	8.5	68.9518	8.5	72.75
9	81	9	76.0603	9	81.5
9.5	90.25	9.5	88.6046	9.5	85.875
10	100	10	98.0715	10	90.4167







On the smoother, after experimenting with the available settings I left for myself, I noticed that increasing the range of the smoothing greatly increases the smoothing effect. To achieve the most effective smoothing it would probably be some mix of having a decently large radius and running the smoothing function a few times. This smoother could also be corrected in a similar fashion to how the Octave smoother was corrected at the ends by adding the target value to the average at a higher weight.