Generating data

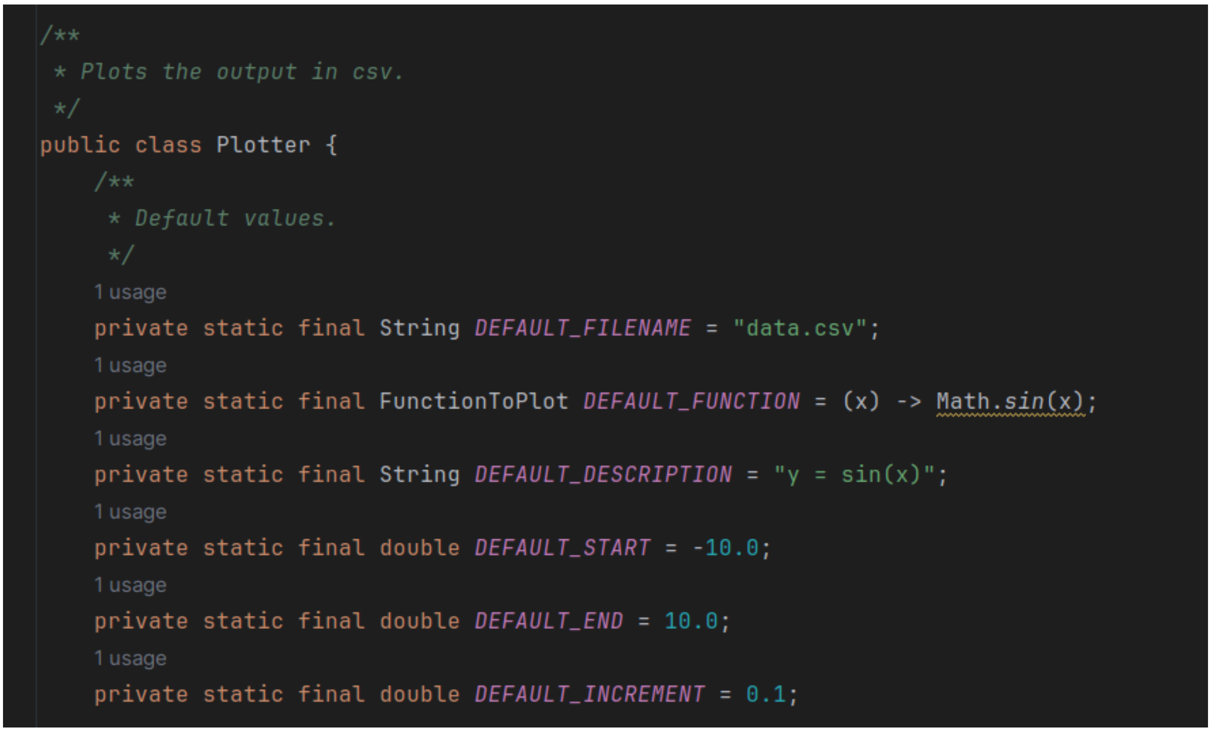
Functions used

My code uses three functions to generate three types of data files:

- y = 3x + cos(2x) to generate **test.csv**
- y = 5x^2 + 2x + 20 to generate **extra.csv**
- y = sin(x) to generate **data.csv**

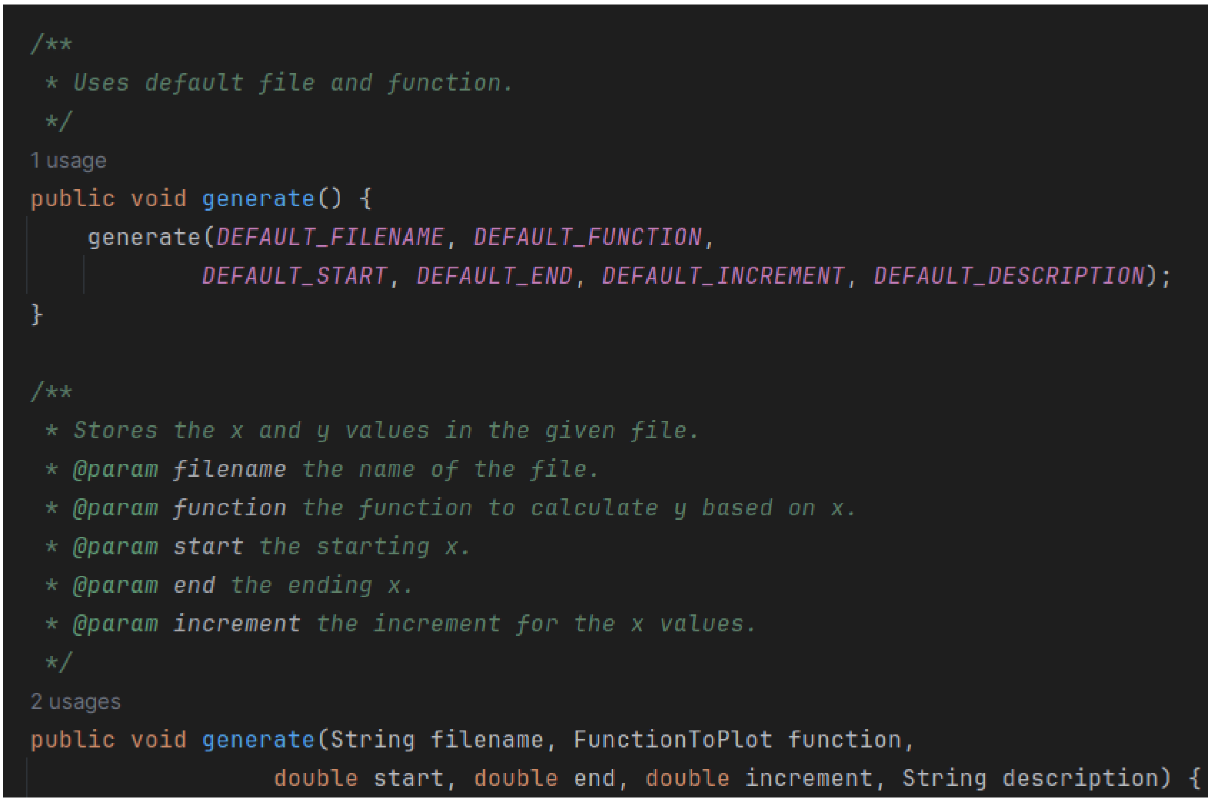
Code

The **Plotter** is responsible for generating a **csv** file. At the start of the class I declare constants that define the default values of filename, function, description, start and end x values and the increment value:



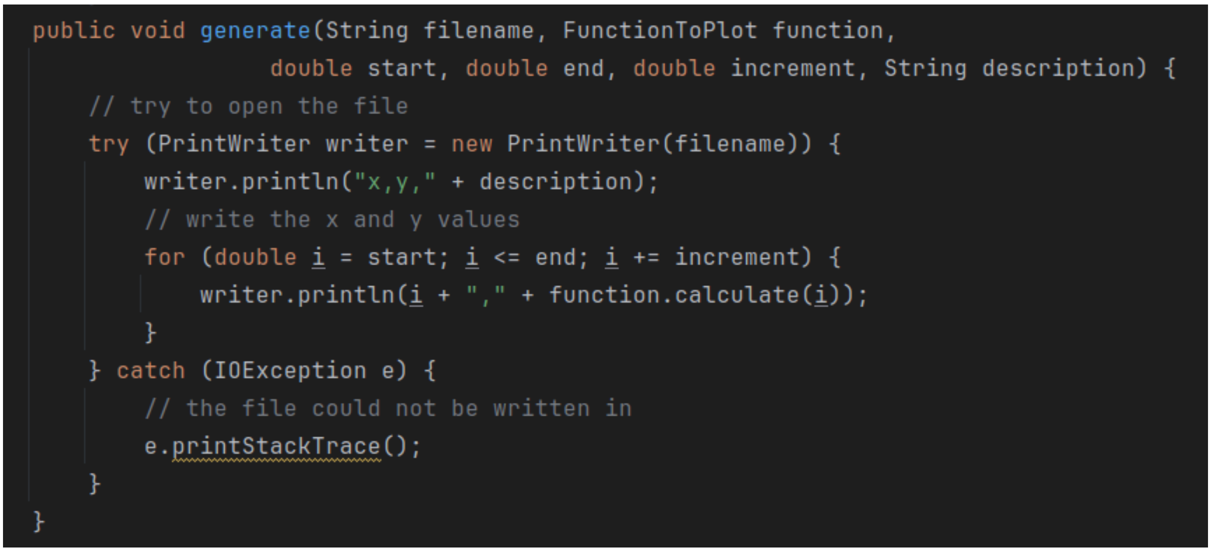
```java
/**
 * Plots the output in csv.
 */
public class Plotter {
    /**
     * Default values.
     */
    1 usage
    private static final String DEFAULT_FILENAME = "data.csv";
    1 usage
    private static final FunctionToPlot DEFAULT_FUNCTION = (x) -> Math.sin(x);
    1 usage
    private static final String DEFAULT_DESCRIPTION = "y = sin(x)";
    1 usage
    private static final double DEFAULT_START = -10.0;
    1 usage
    private static final double DEFAULT_END = 10.0;
    1 usage
    private static final double DEFAULT_INCREMENT = 0.1;
```

The **generate** method is responsible for generating the **csv** file. There are two overloaded versions of the method: the first version accepts no parameter and delegates to the second version, passing the default values as parameters:



```
/**
 * Uses default file and function.
 */
1 usage
public void generate() {
    generate(DEFAULT_FILENAME, DEFAULT_FUNCTION,
            DEFAULT_START, DEFAULT_END, DEFAULT_INCREMENT, DEFAULT_DESCRIPTION);
}

/**
 * Stores the x and y values in the given file.
 * @param filename the name of the file.
 * @param function the function to calculate y based on x.
 * @param start the starting x.
 * @param end the ending x.
 * @param increment the increment for the x values.
 */
2 usages
public void generate(String filename, FunctionToPlot function,
                     double start, double end, double increment, String description) {
```

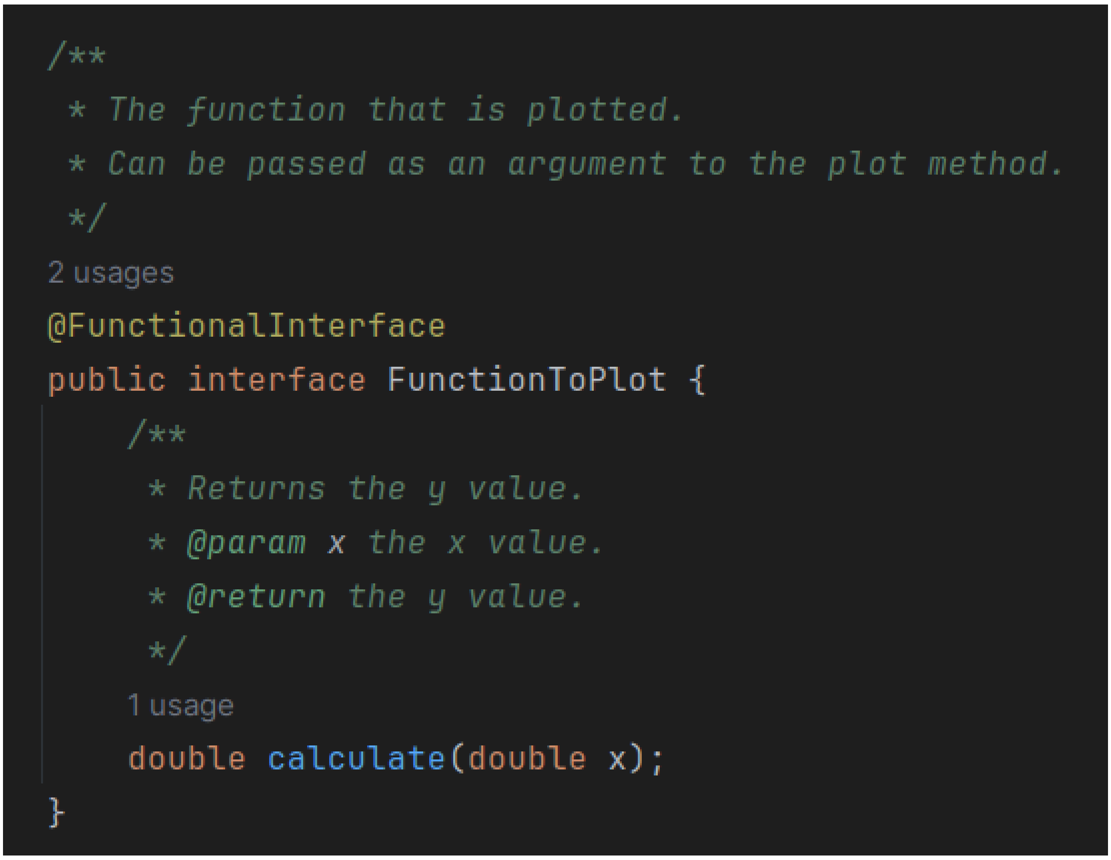
The second version uses the provided parameters to generate the file. The **PrintWriter** class is used to write new lines into the text file, and the **function** parameter is used to generate **y** values for the provided **x** values:



```
public void generate(String filename, FunctionToPlot function,
                     double start, double end, double increment, String description) {
    // try to open the file
    try (PrintWriter writer = new PrintWriter(filename)) {
        writer.println("x,y," + description);
        // write the x and y values
        for (double i = start; i <= end; i += increment) {
            writer.println(i + "," + function.calculate(i));
        }
    } catch (IOException e) {
        // the file could not be written in
        e.printStackTrace();
    }
}
```

The first line in the generated file is the header: it marks the **x** and **y** columns and provides a short description of the function used.

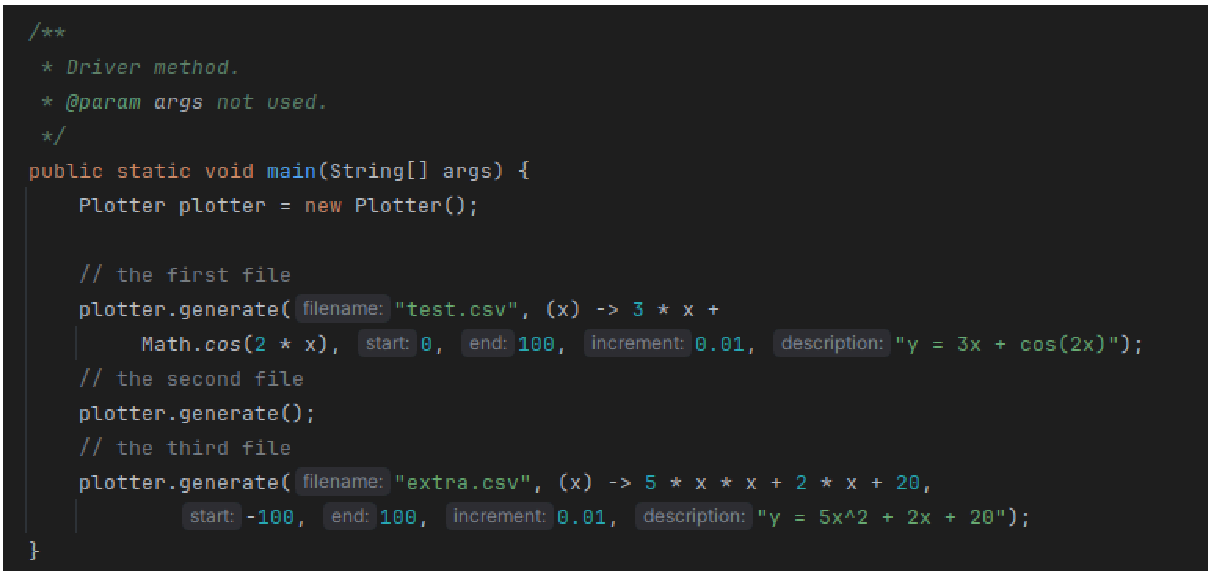
The **FunctionToPlot** is an interface:



```
/**
 * The function that is plotted.
 * Can be passed as an argument to the plot method.
 */
2 usages
@FunctionalInterface
public interface FunctionToPlot {
    /**
     * Returns the y value.
     * @param x the x value.
     * @return the y value.
     */
    1 usage
    double calculate(double x);
}
```

The **FunctionalInterface** annotation marks this interface as one that could be represented by a lambda expression.

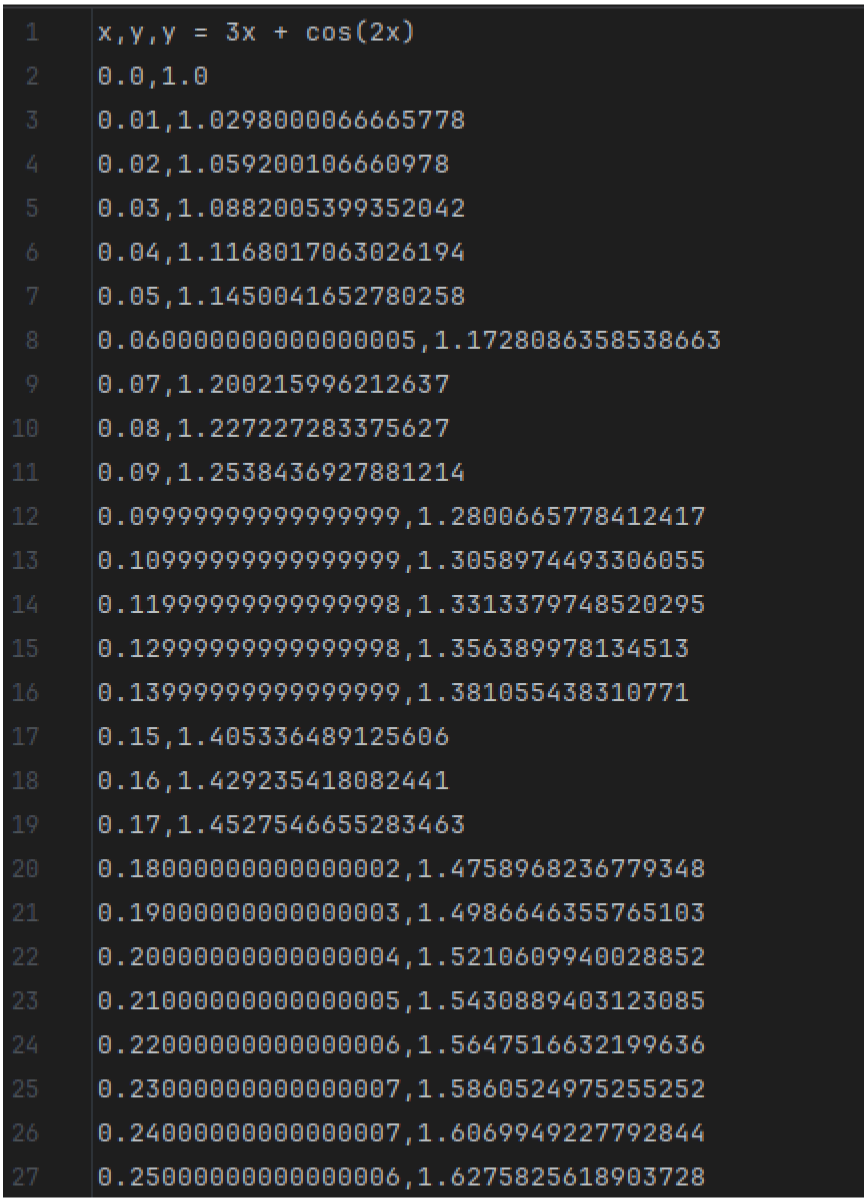
The **main** method is used to instantiate the **Plotter** and generate two **csv** files:



```
/**
 * Driver method.
 * @param args not used.
 */
public static void main(String[] args) {
    Plotter plotter = new Plotter();

    // the first file
    plotter.generate( filename: "test.csv", (x) -> 3 * x +
         Math.cos(2 * x),  start: 0,  end: 100,  increment: 0.01,  description: "y = 3x + cos(2x)");
    // the second file
    plotter.generate();
    // the third file
    plotter.generate( filename: "extra.csv", (x) -> 5 * x * x + 2 * x + 20,
            start: -100,  end: 100,  increment: 0.01,  description: "y = 5x^2 + 2x + 20");
}
```

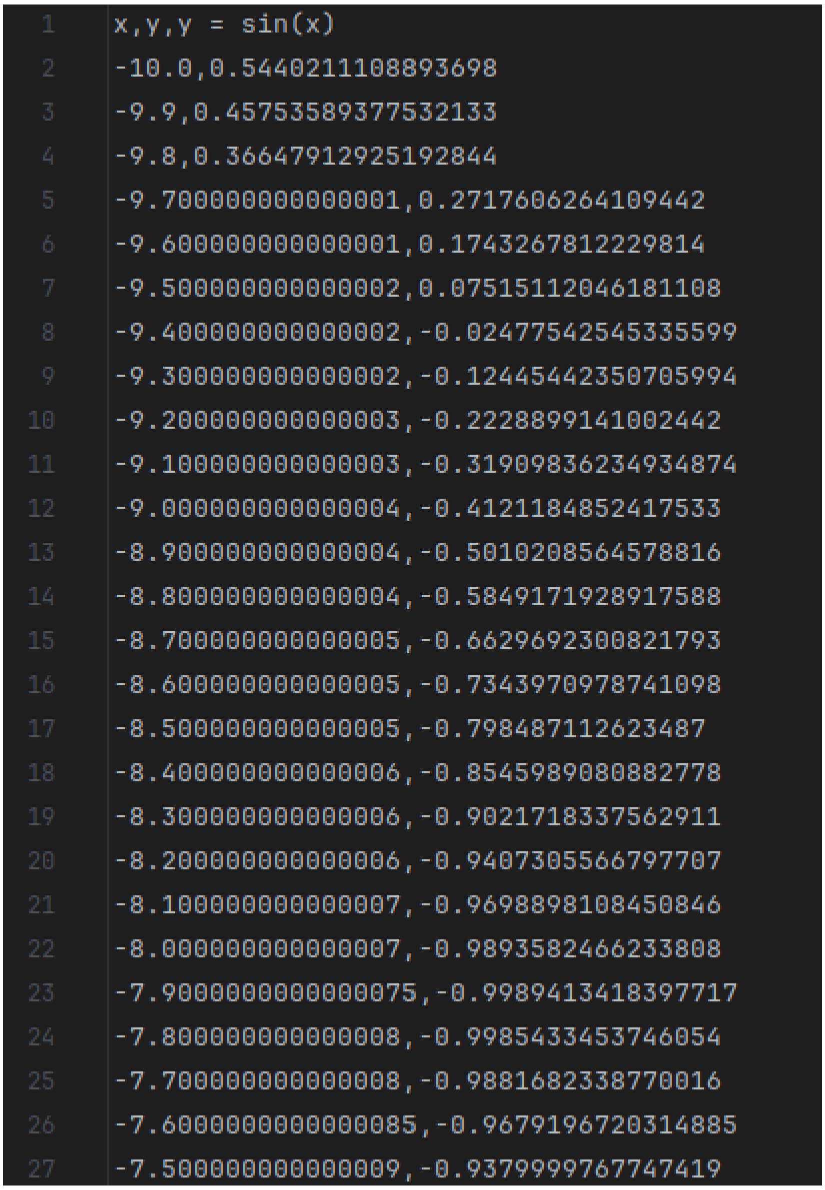
Running the **Plotter**’s **main** method results in the following files being generated:

test.csv



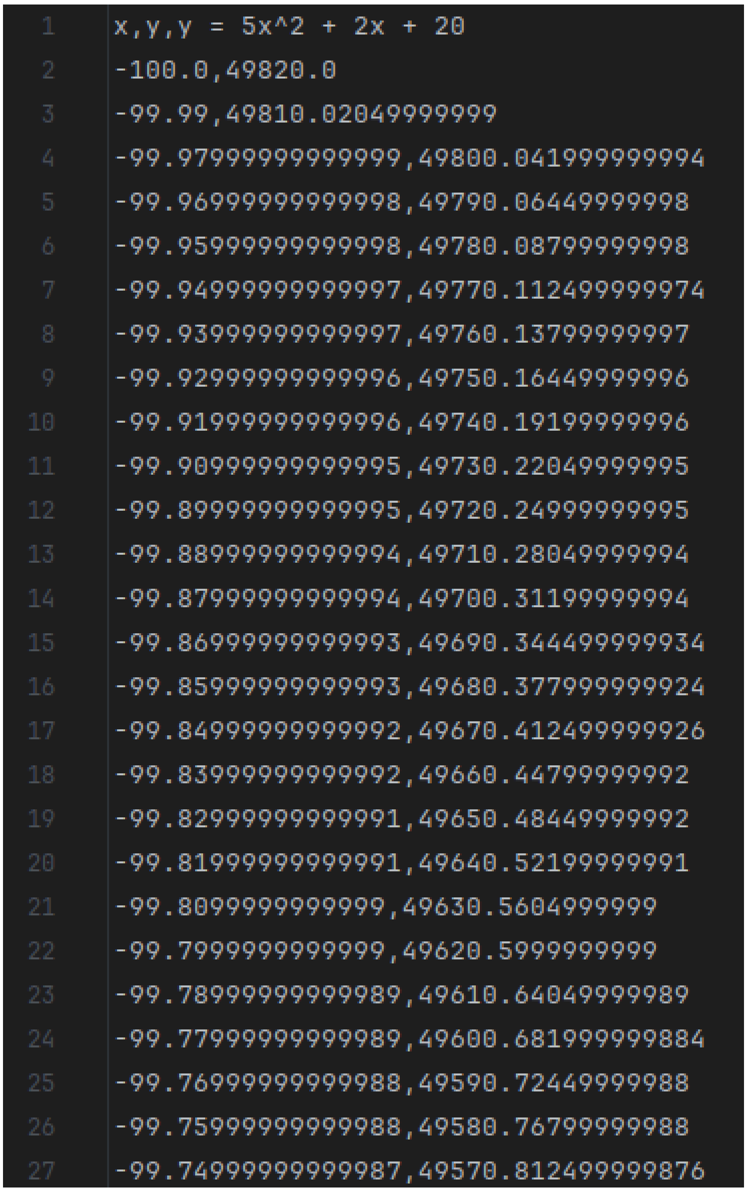
```
x,y,y = 3x + cos(2x)
0.0,1.0
0.01,1.0298000066665778
0.02,1.059200106660978
0.03,1.0882005399352042
0.04,1.1168017063026194
0.05,1.1450041652780258
0.060000000000000005,1.1728086358538663
0.07,1.200215996212637
0.08,1.227227283375627
0.09,1.2538436927881214
0.09999999999999999,1.2800665778412417
0.10999999999999999,1.3058974493306055
0.11999999999999998,1.3313379748520295
0.12999999999999998,1.356389978134513
0.13999999999999999,1.381055438310771
0.15,1.405336489125606
0.16,1.429235418082441
0.17,1.4527546655283463
0.18000000000000002,1.4758968236779348
0.19000000000000003,1.4986646355765103
0.20000000000000004,1.5210609940028852
0.21000000000000005,1.5430889403123085
0.22000000000000006,1.5647516632199636
0.23000000000000007,1.5860524975255252
0.24000000000000007,1.6069949227792844
0.25000000000000006,1.6275825618903728
```

data.csv



```
x,y,y = sin(x)
-10.0,0.5440211108893698
-9.9,0.45753589377532133
-9.8,0.36647912925192844
-9.700000000000001,0.2717606264109442
-9.600000000000001,0.1743267812229814
-9.500000000000002,0.07515112046181108
-9.400000000000002,-0.02477542545335599
-9.300000000000002,-0.12445442350705994
-9.200000000000003,-0.2228899141002442
-9.100000000000003,-0.31909836234934874
-9.000000000000004,-0.4121184852417533
-8.900000000000004,-0.5010208564578816
-8.800000000000004,-0.5849171928917588
-8.700000000000005,-0.6629692300821793
-8.600000000000005,-0.7343970978741098
-8.500000000000005,-0.798487112623487
-8.400000000000006,-0.8545989080882778
-8.300000000000006,-0.9021718337562911
-8.200000000000006,-0.9407305566797707
-8.100000000000007,-0.9698898108450846
-8.000000000000007,-0.9893582466233808
-7.9000000000000075,-0.9989413418397717
-7.800000000000008,-0.9985433453746054
-7.700000000000008,-0.9881682338770016
-7.6000000000000085,-0.9679196720314885
-7.500000000000009,-0.9379999767747419
```

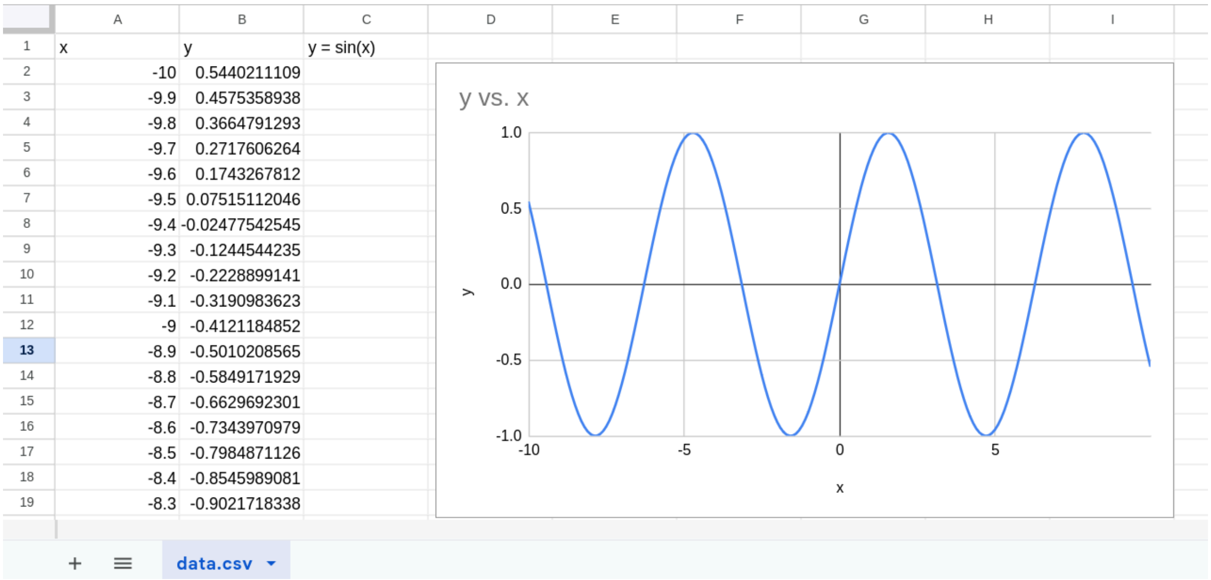
extra.csv



```
x,y,y = 5x^2 + 2x + 20
-100.0,49820.0
-99.99,49810.02049999999
-99.97999999999999,49800.041999999994
-99.96999999999998,49790.06449999998
-99.95999999999998,49780.08799999998
-99.94999999999997,49770.112499999974
-99.93999999999997,49760.13799999997
-99.92999999999996,49750.16449999996
-99.91999999999996,49740.19199999996
-99.90999999999995,49730.22049999995
-99.89999999999995,49720.24999999995
-99.88999999999994,49710.28049999994
-99.87999999999994,49700.31199999994
-99.86999999999993,49690.344499999934
-99.85999999999993,49680.377999999924
-99.84999999999992,49670.412499999926
-99.83999999999992,49660.44799999992
-99.82999999999991,49650.48449999992
-99.81999999999991,49640.52199999991
-99.8099999999999,49630.5604999999
-99.7999999999999,49620.5999999999
-99.78999999999989,49610.64049999989
-99.77999999999989,49600.681999999884
-99.76999999999988,49590.72449999988
-99.75999999999988,49580.76799999988
-99.74999999999987,49570.812499999876
```

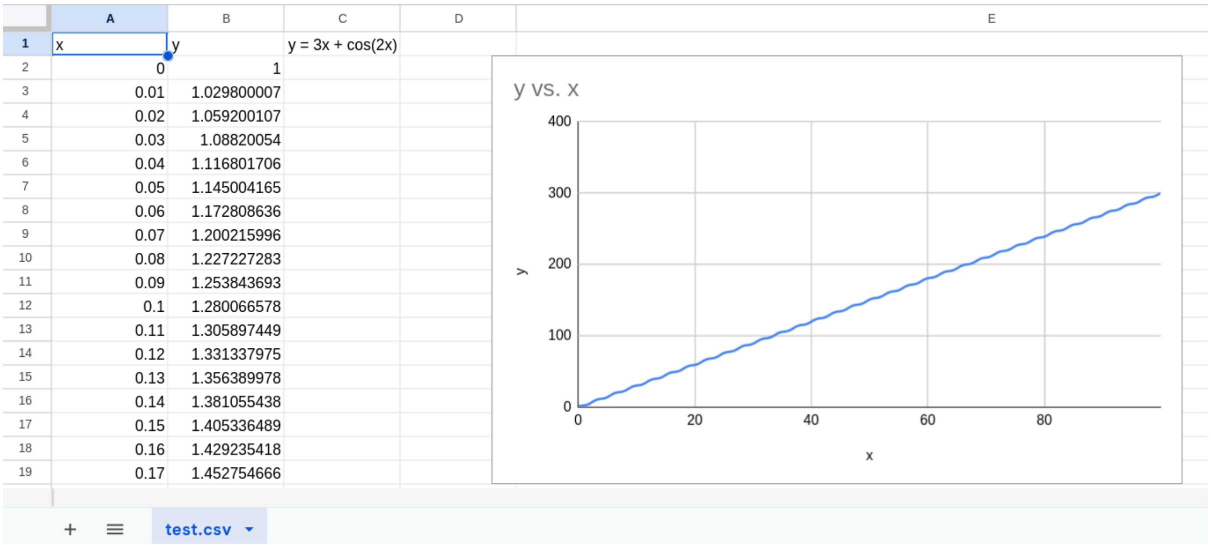
I use Google Sheets to plot the data in the **csv** files:

data.csv



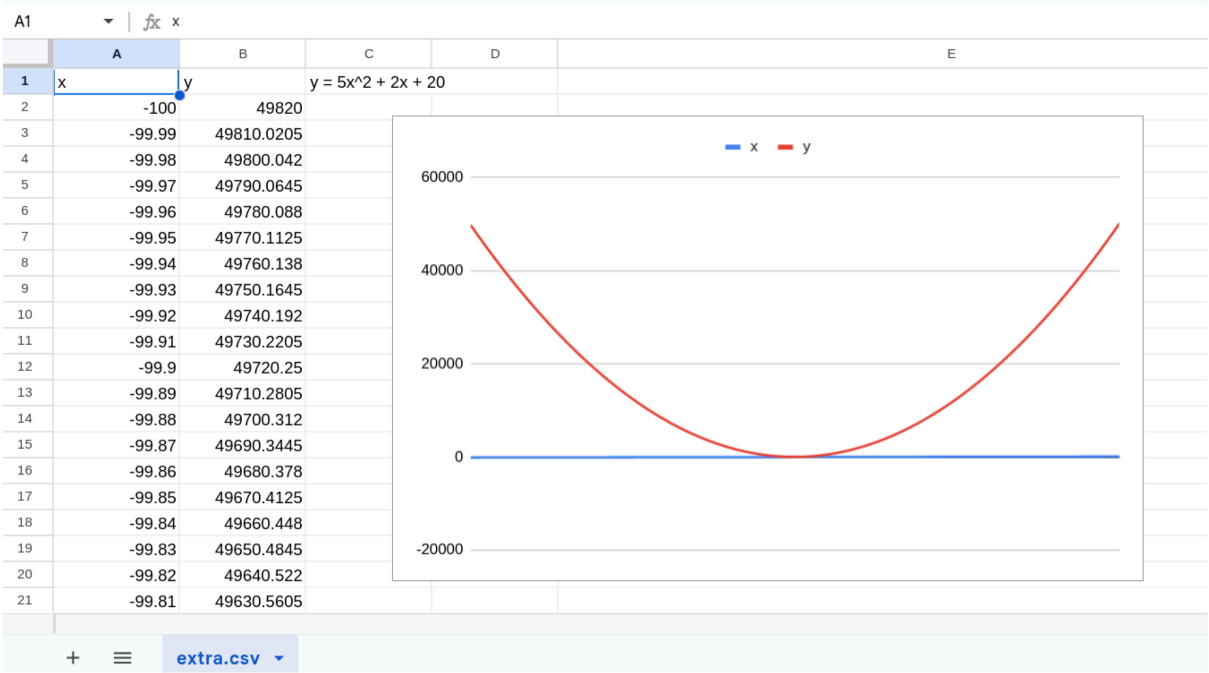
	A	B	C
1	x	y	y = sin(x)
2	-10	0.5440211109	
3	-9.9	0.4575358938	
4	-9.8	0.3664791293	
5	-9.7	0.2717606264	
6	-9.6	0.1743267812	
7	-9.5	0.07515112046	
8	-9.4	-0.02477542545	
9	-9.3	-0.1244544235	
10	-9.2	-0.2228899141	
11	-9.1	-0.3190983623	
12	-9	-0.4121184852	
13	-8.9	-0.5010208565	
14	-8.8	-0.5849171929	
15	-8.7	-0.6629692301	
16	-8.6	-0.7343970979	
17	-8.5	-0.7984871126	
18	-8.4	-0.8545989081	
19	-8.3	-0.9021718338	

test.csv



	A	B	C
1	x	y	y = 3x + cos(2x)
2	0	1	
3	0.01	1.029800007	
4	0.02	1.059200107	
5	0.03	1.08820054	
6	0.04	1.116801706	
7	0.05	1.145004165	
8	0.06	1.172808636	
9	0.07	1.200215996	
10	0.08	1.227227283	
11	0.09	1.253843693	
12	0.1	1.280066578	
13	0.11	1.305897449	
14	0.12	1.331337975	
15	0.13	1.356389978	
16	0.14	1.381055438	
17	0.15	1.405336489	
18	0.16	1.429235418	
19	0.17	1.452754666	

extra.csv

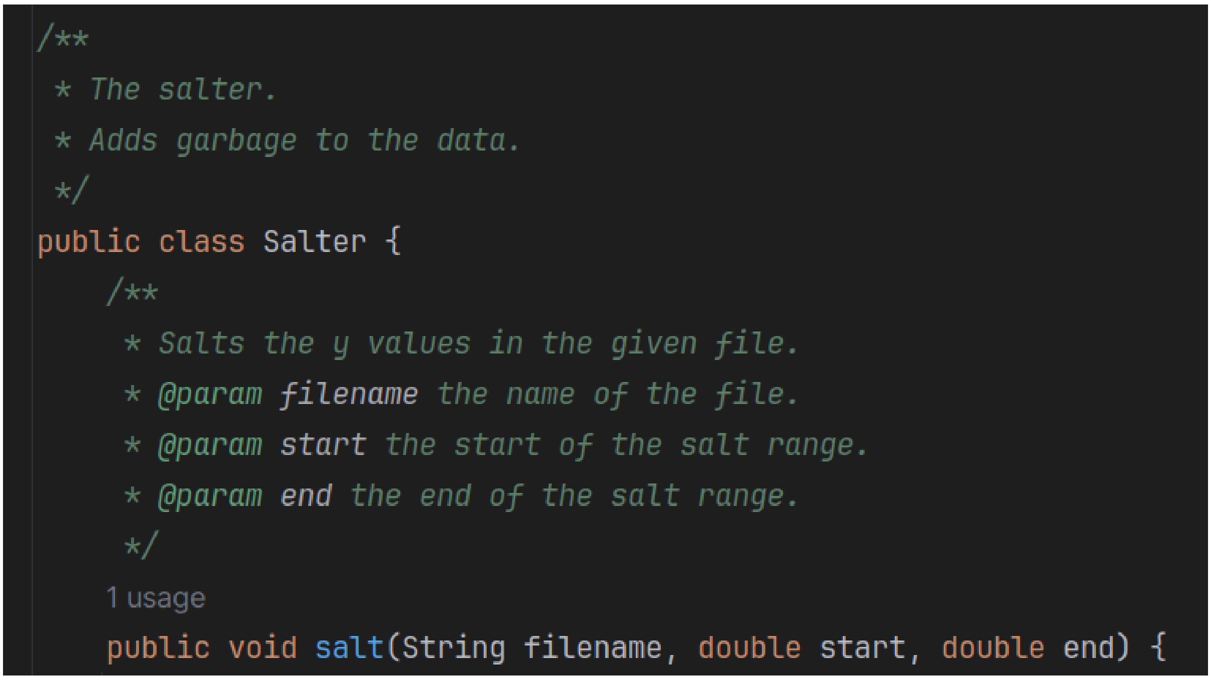


	A	B	C	D	E
1	x	y	y = 5x^2 + 2x + 20		
2	-100	49820			
3	-99.99	49810.0205			
4	-99.98	49800.042			
5	-99.97	49790.0645			
6	-99.96	49780.088			
7	-99.95	49770.1125			
8	-99.94	49760.138			
9	-99.93	49750.1645			
10	-99.92	49740.192			
11	-99.91	49730.2205			
12	-99.9	49720.25			
13	-99.89	49710.2805			
14	-99.88	49700.312			
15	-99.87	49690.3445			
16	-99.86	49680.378			
17	-99.85	49670.4125			
18	-99.84	49660.448			
19	-99.83	49650.4845			
20	-99.82	49640.522			
21	-99.81	49630.5605			

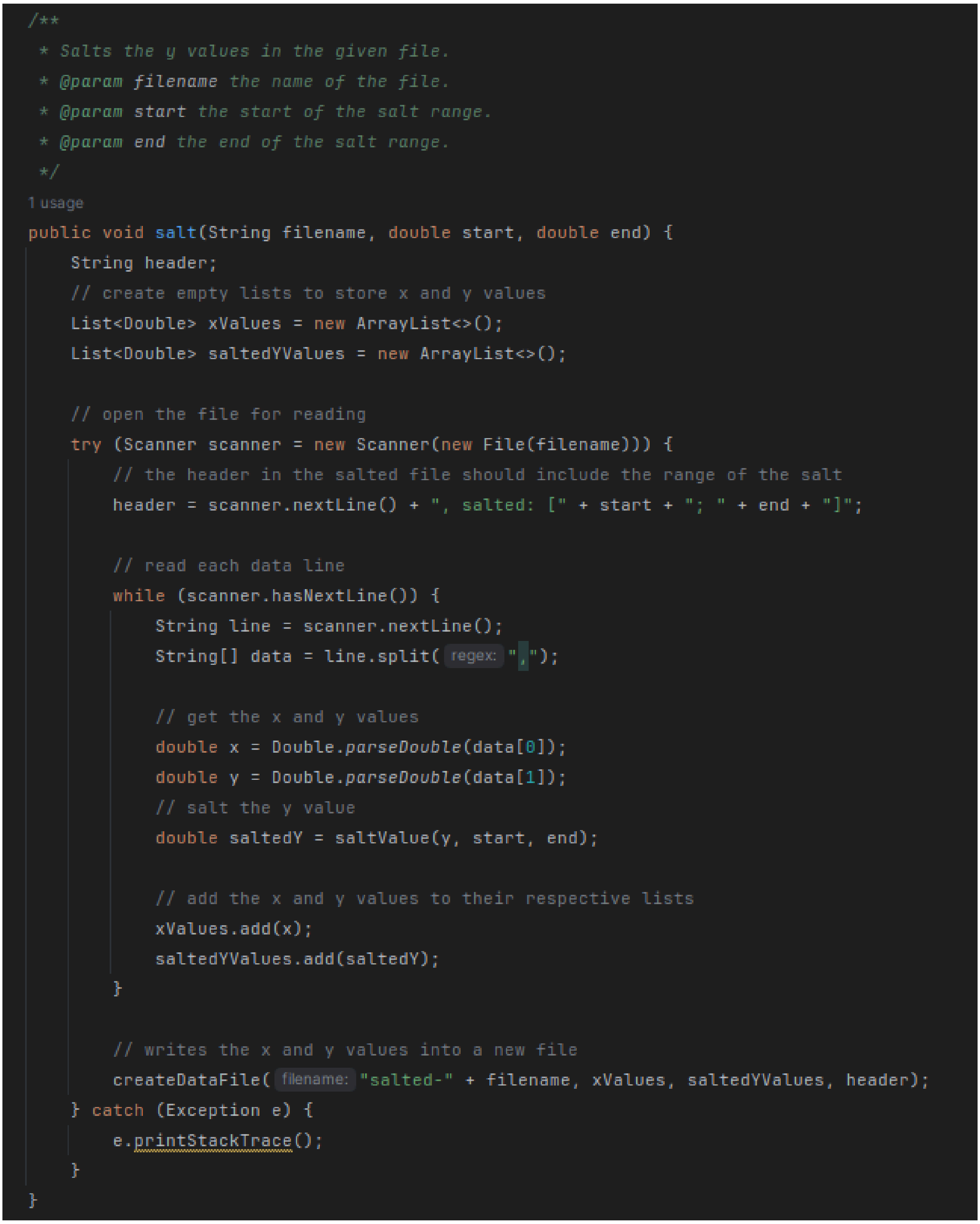
There are 10,000 values in **test.csv**, 201 values in **data.csv**, and 20,000 values in **extra.csv**.

Salting

The **Salter** class is responsible for salting y values. The **salt** method reads the given file, and salts the y values, and writes the x and y values into a new file:

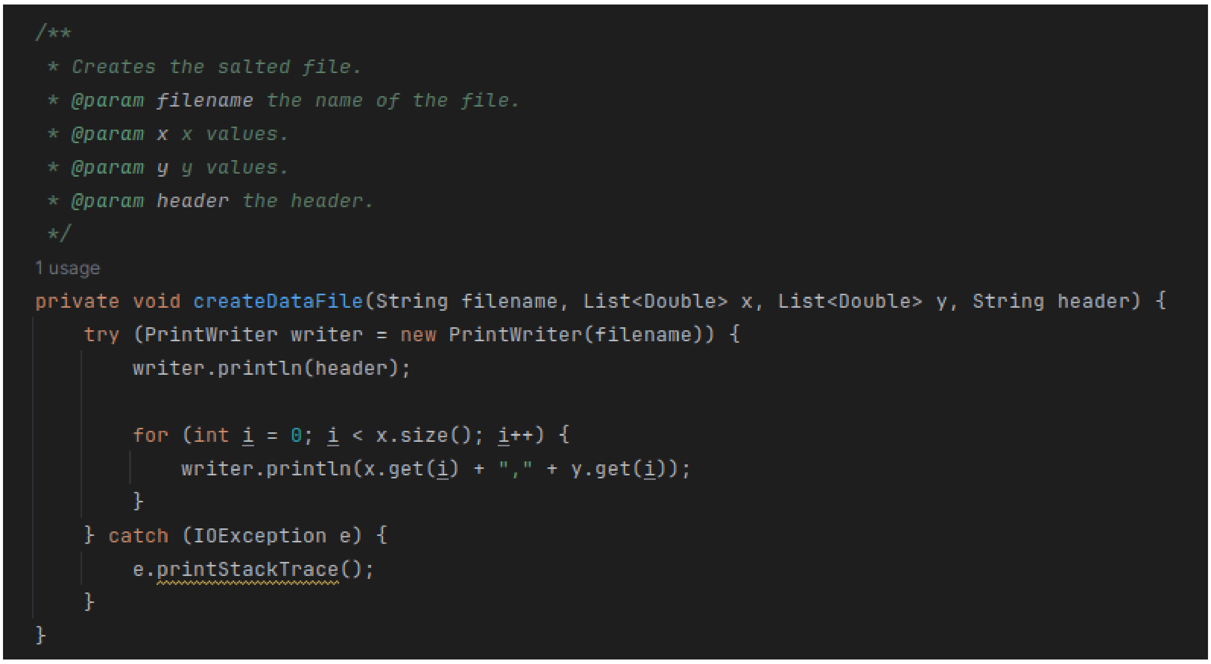


```
/**
 * The salter.
 * Adds garbage to the data.
 */
public class Salter {
    /**
     * Salts the y values in the given file.
     * @param filename the name of the file.
     * @param start the start of the salt range.
     * @param end the end of the salt range.
     */
    1 usage
    public void salt(String filename, double start, double end) {
```



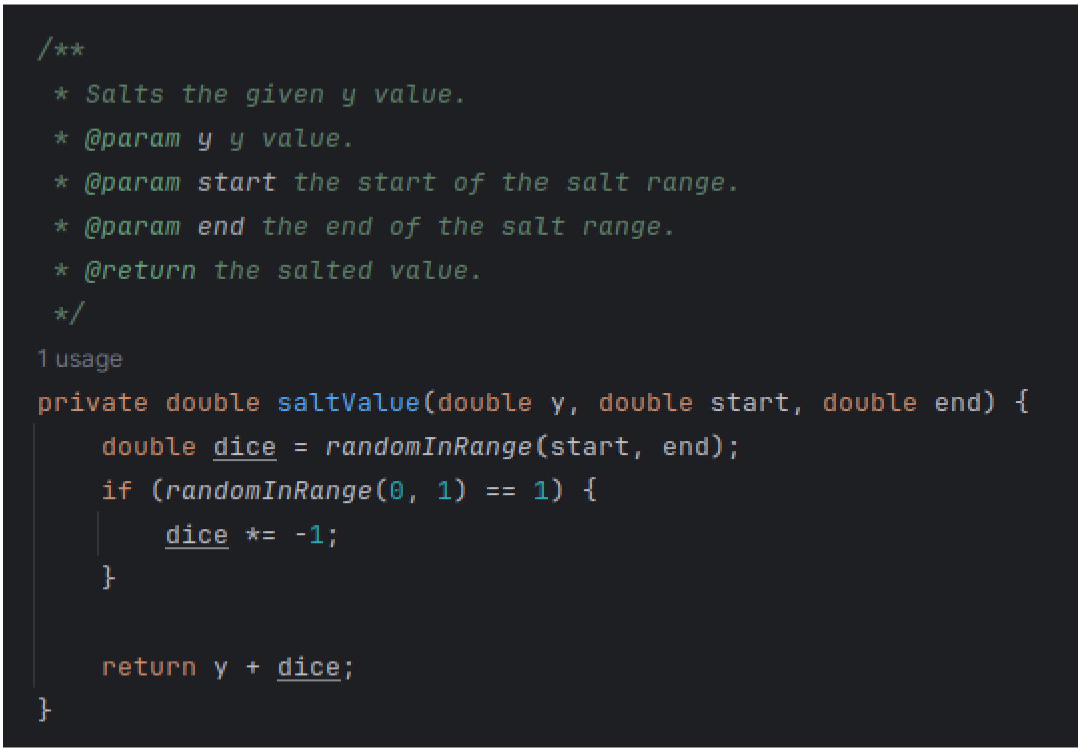
```
/**
 * Salts the y values in the given file.
 * @param filename the name of the file.
 * @param start the start of the salt range.
 * @param end the end of the salt range.
 */
1 usage
public void salt(String filename, double start, double end) {
    String header;
    // create empty lists to store x and y values
    List<Double> xValues = new ArrayList<>();
    List<Double> saltedYValues = new ArrayList<>();

    // open the file for reading
    try (Scanner scanner = new Scanner(new File(filename))) {
        // the header in the salted file should include the range of the salt
        header = scanner.nextLine() + ", salted: [" + start + "; " + end + "]";

        // read each data line
        while (scanner.hasNextLine()) {
            String line = scanner.nextLine();
            String[] data = line.split( regex: ",");

            // get the x and y values
            double x = Double.parseDouble(data[0]);
            double y = Double.parseDouble(data[1]);
            // salt the y value
            double saltedY = saltValue(y, start, end);

            // add the x and y values to their respective lists
            xValues.add(x);
            saltedYValues.add(saltedY);
        }

        // writes the x and y values into a new file
        createDataFile( filename: "salted-" + filename, xValues, saltedYValues, header);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

The **createDataFile** method writes the data in a new file:



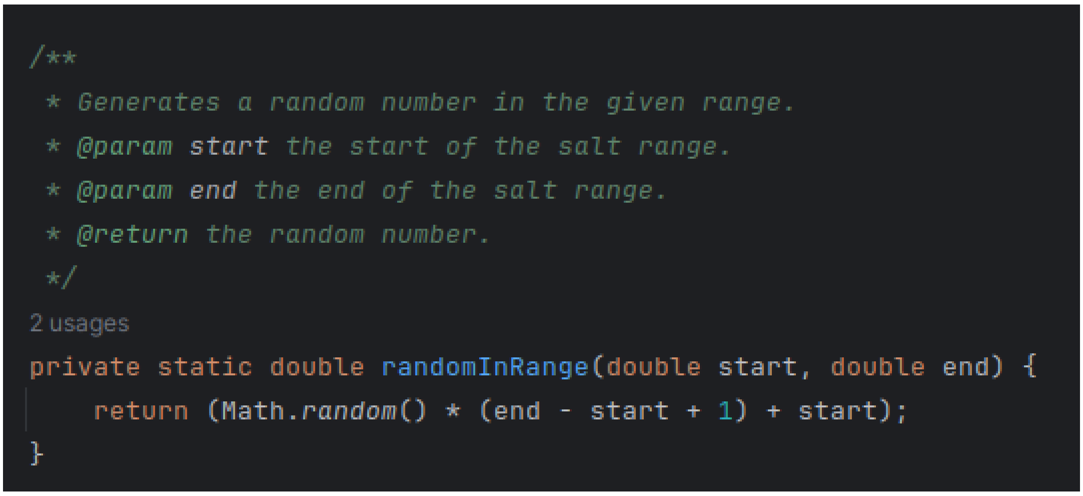
```
/**
 * Creates the salted file.
 * @param filename the name of the file.
 * @param x x values.
 * @param y y values.
 * @param header the header.
 */
1 usage
private void createDataFile(String filename, List<Double> x, List<Double> y, String header) {
    try (PrintWriter writer = new PrintWriter(filename)) {
        writer.println(header);

        for (int i = 0; i < x.size(); i++) {
            writer.println(x.get(i) + "," + y.get(i));
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

The **saltValue** method returns a salted y value:



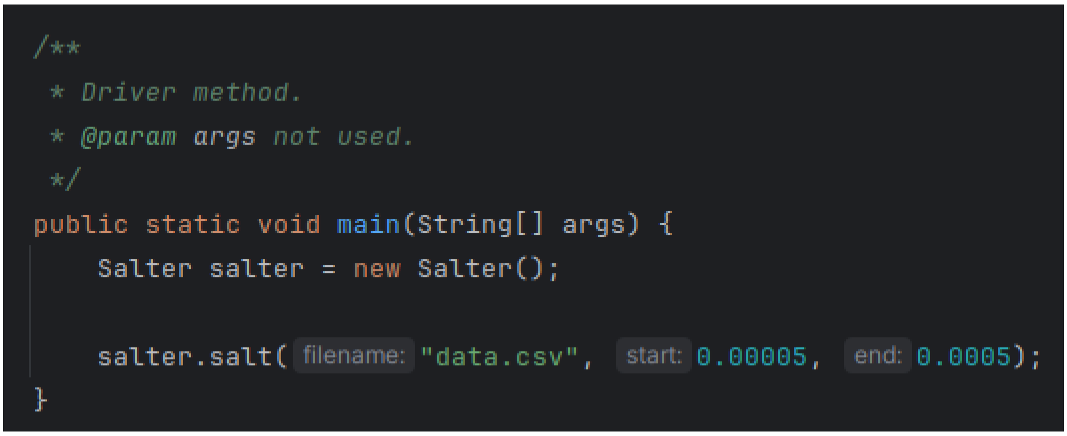
```
/**
 * Salts the given y value.
 * @param y y value.
 * @param start the start of the salt range.
 * @param end the end of the salt range.
 * @return the salted value.
 */
1 usage
private double saltValue(double y, double start, double end) {
    double dice = randomInRange(start, end);
    if (randomInRange(0, 1) == 1) {
        dice *= -1;
    }

    return y + dice;
}
```

The **randomInRange** method generates a random number in the given range:



```
/**
 * Generates a random number in the given range.
 * @param start the start of the salt range.
 * @param end the end of the salt range.
 * @return the random number.
 */
2 usages
private static double randomInRange(double start, double end) {
    return (Math.random() * (end - start + 1) + start);
}
```

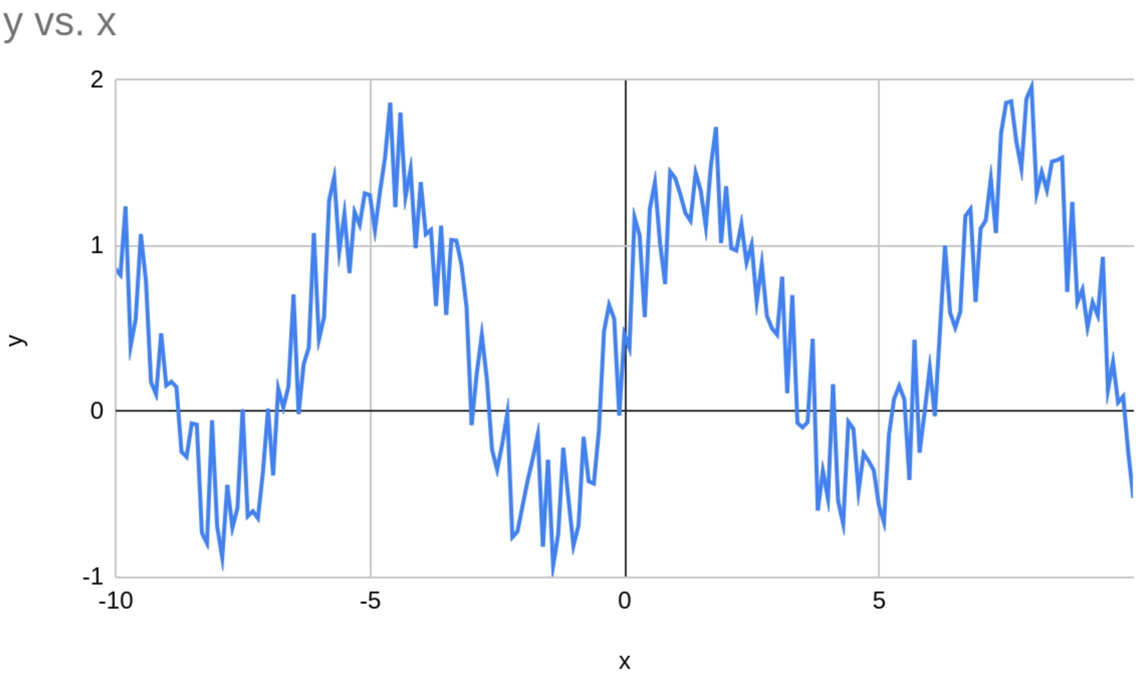
The **main** method is used to generate salted **csv** files:



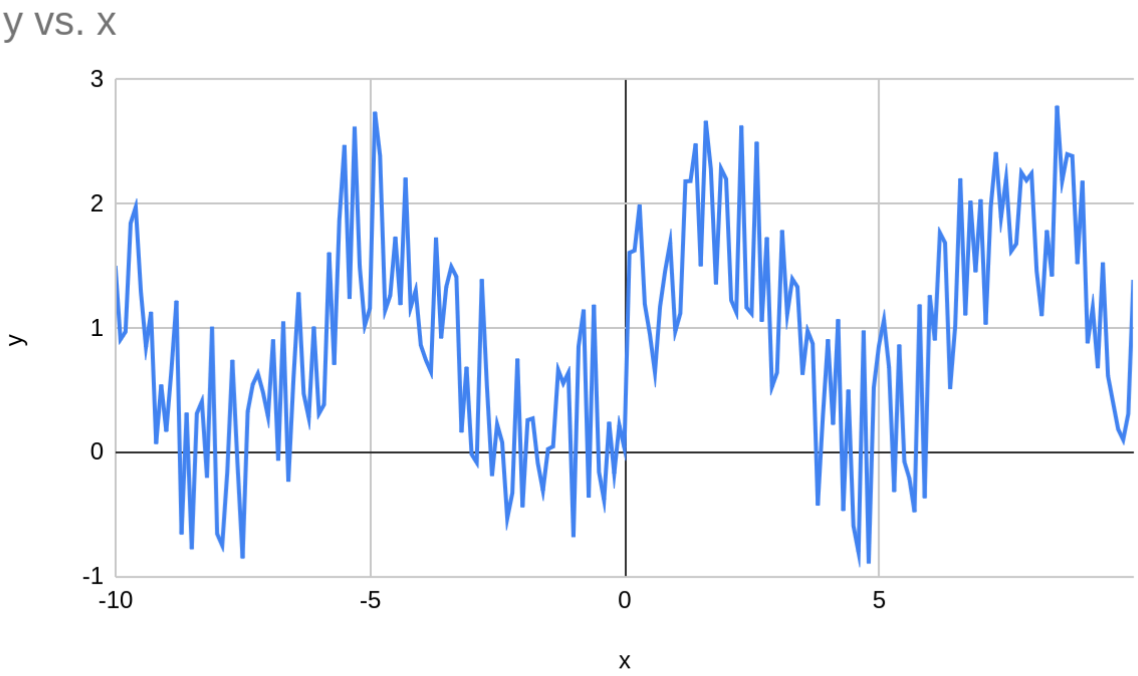
```
/**
 * Driver method.
 * @param args not used.
 */
public static void main(String[] args) {
    Salter salter = new Salter();

    salter.salt( filename: "data.csv",  start: 0.00005,  end: 0.0005);
}
```

Salting **data.csv**

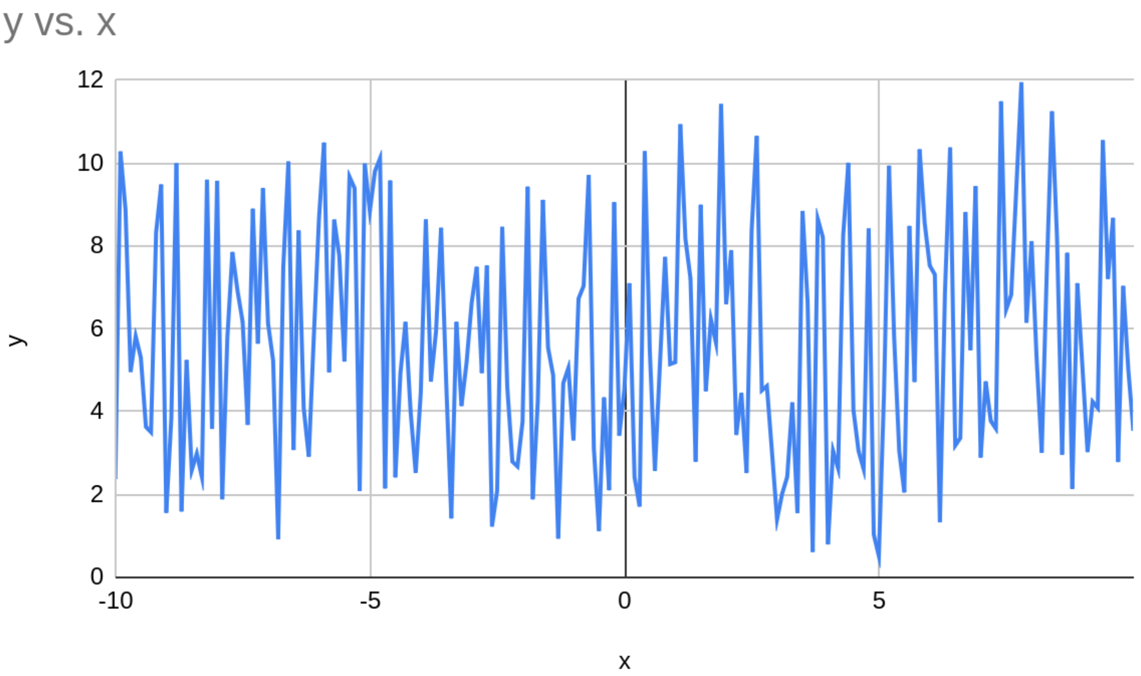
Salt range: [0.00005;0.0005]



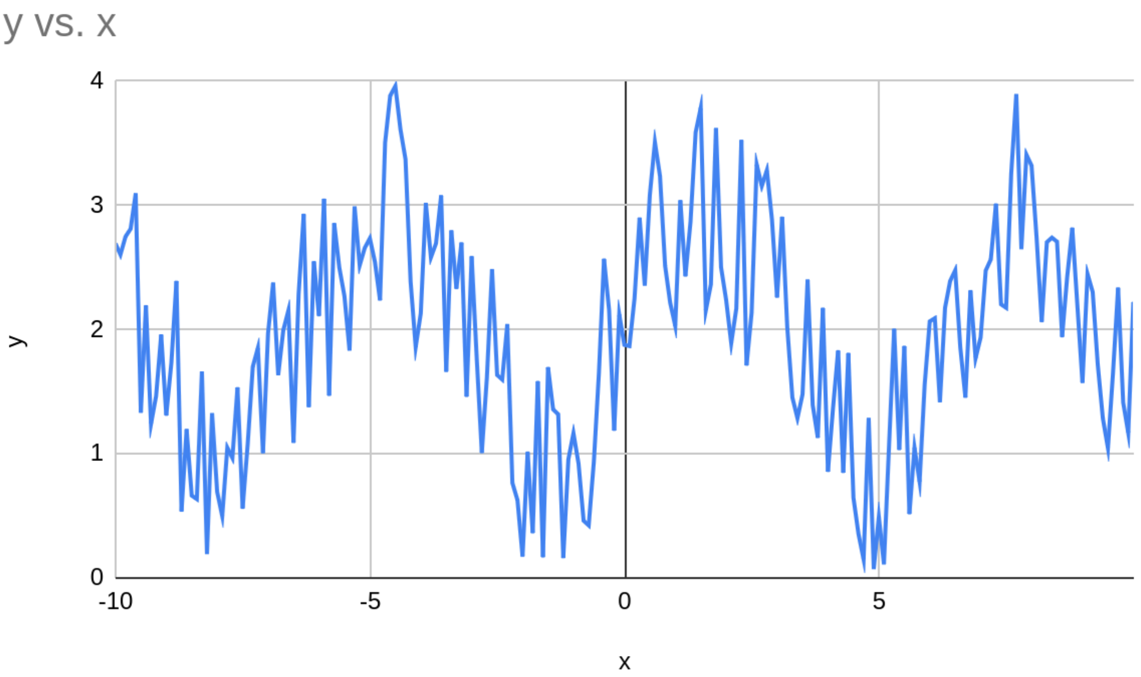
Salt range: [0.00005;1]



Salt range: [1;10]



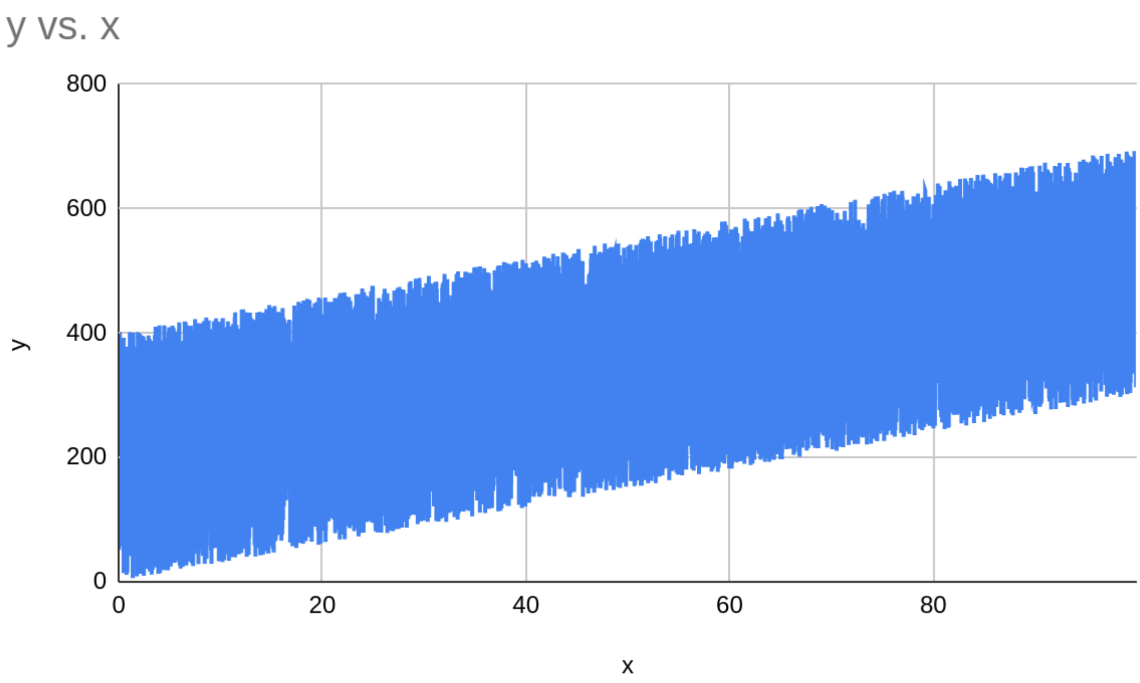
Salt range: [1;2]



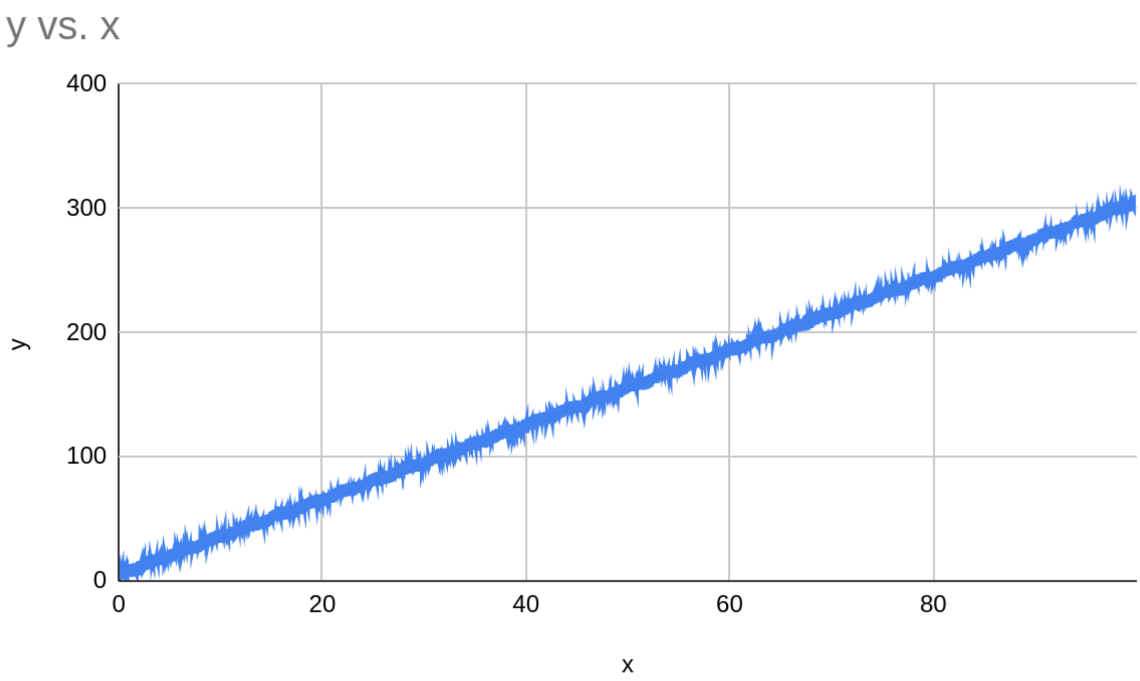
As can be seen from the charts, the higher the difference between the start and end of the salt range, the more unrecognizable the original graph is. It’s also important to select a reasonable salt range: if the **y** values in the original graph are all within [0, 1] range, salt values in [1, 10] range rather than adding some volatility to the original data, would destroy all meaningful patterns the data used to represent.

Salting **test.csv**

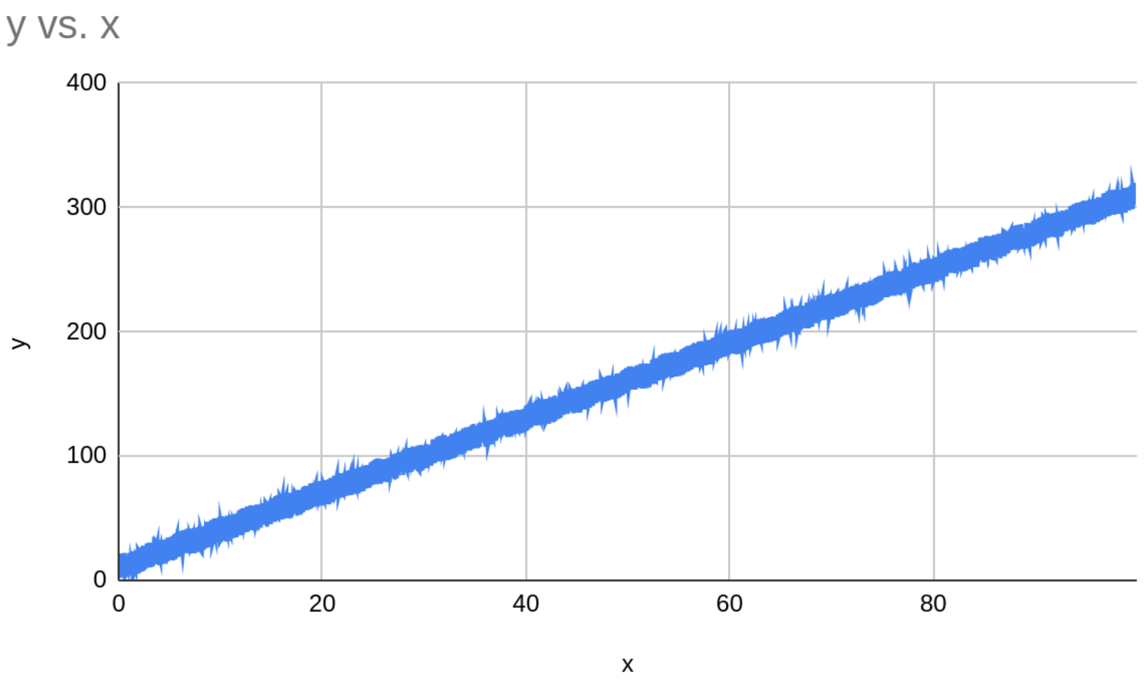
salt range: [0;400]



salt range: [0;10]



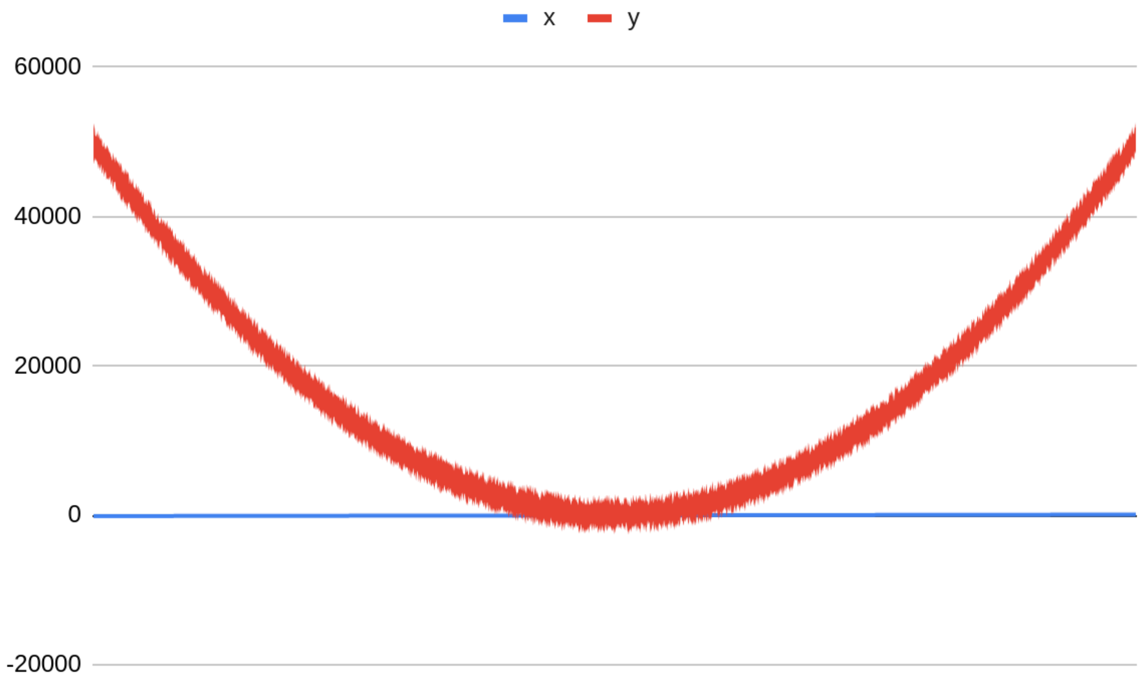
salt range: [0;20]



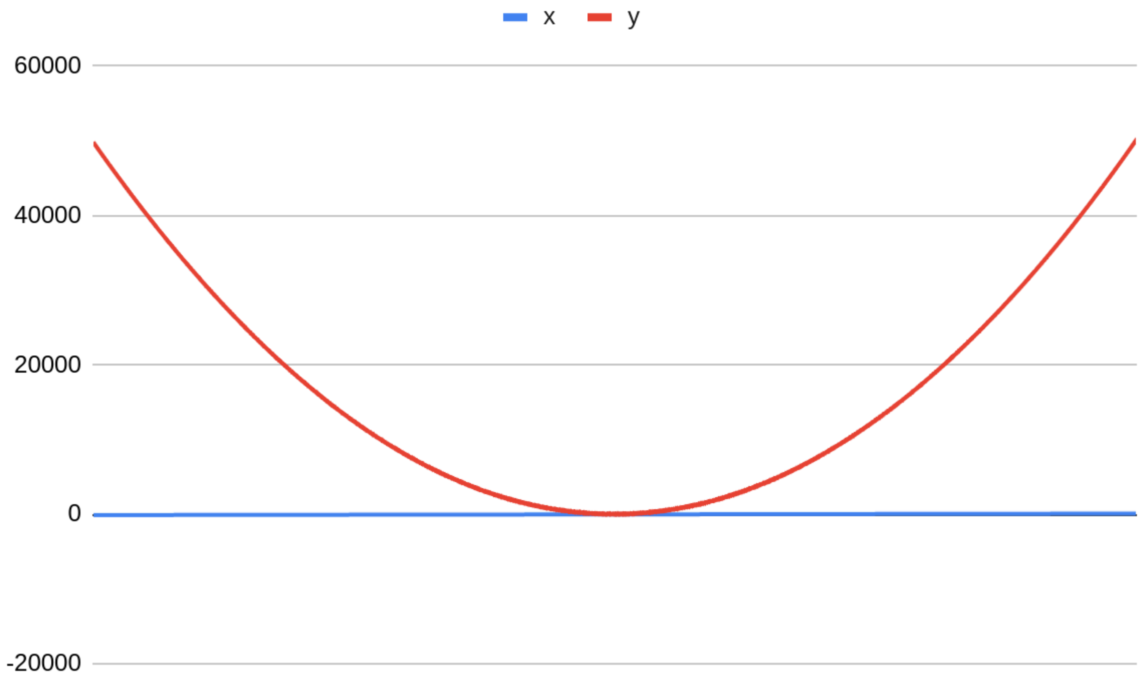
The original chart is mostly linear. The salt range shouldn't be too broad else the data becomes just a collection of random **y** values.

Salting **extra.csv**

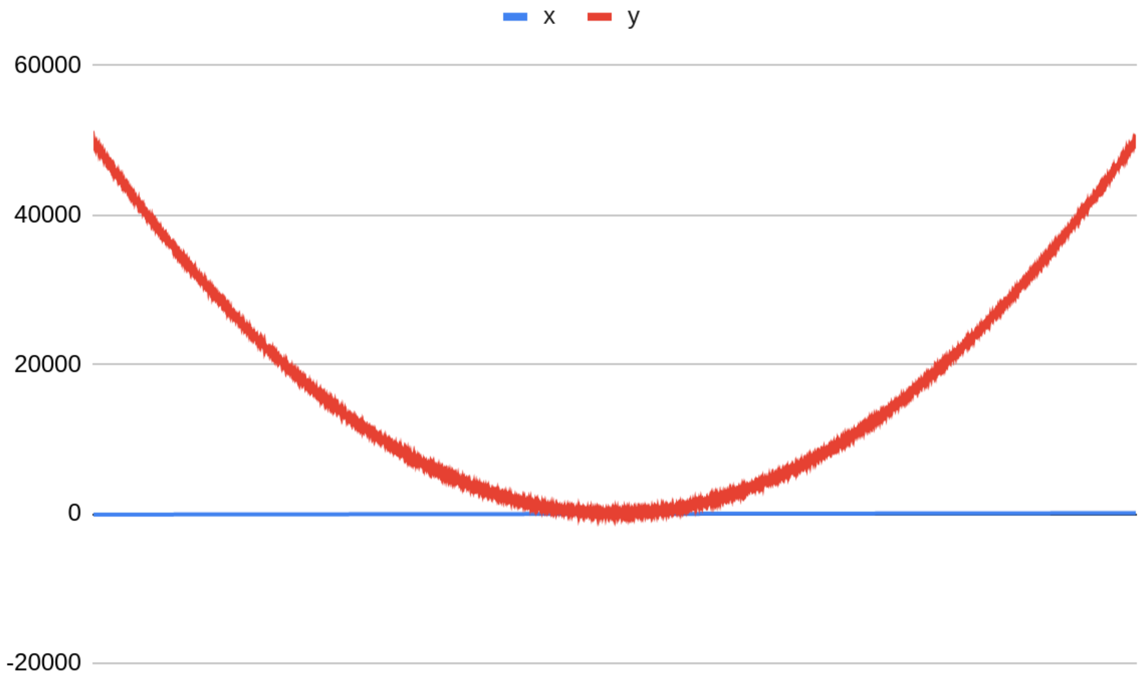
salt range: [1;100]



salt range: [1;10]



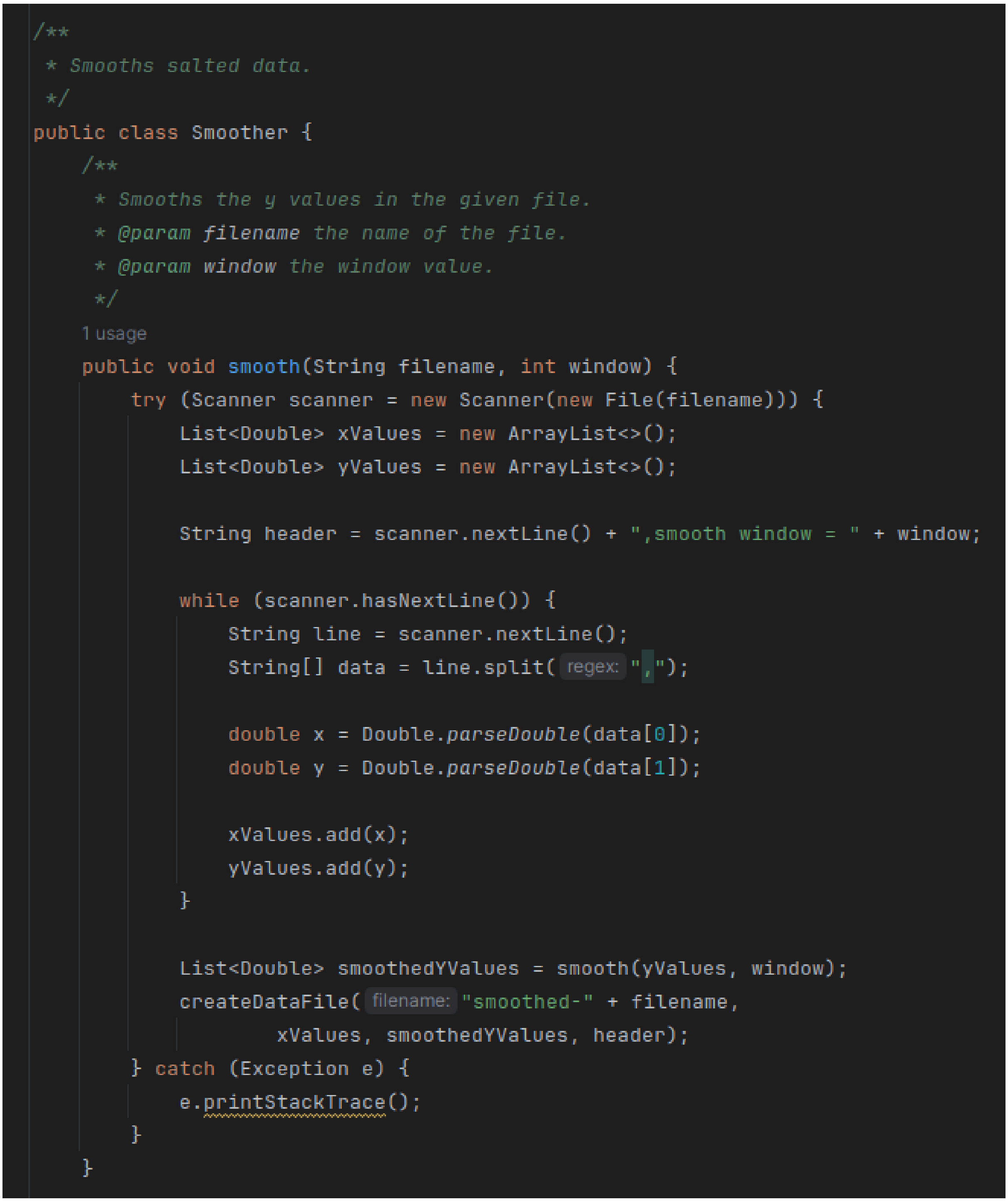
salt range: [1;50]



For this function, the salt range should be significant in the upper boundary should be at last 50 - any lower boundary would make the salting negligible.

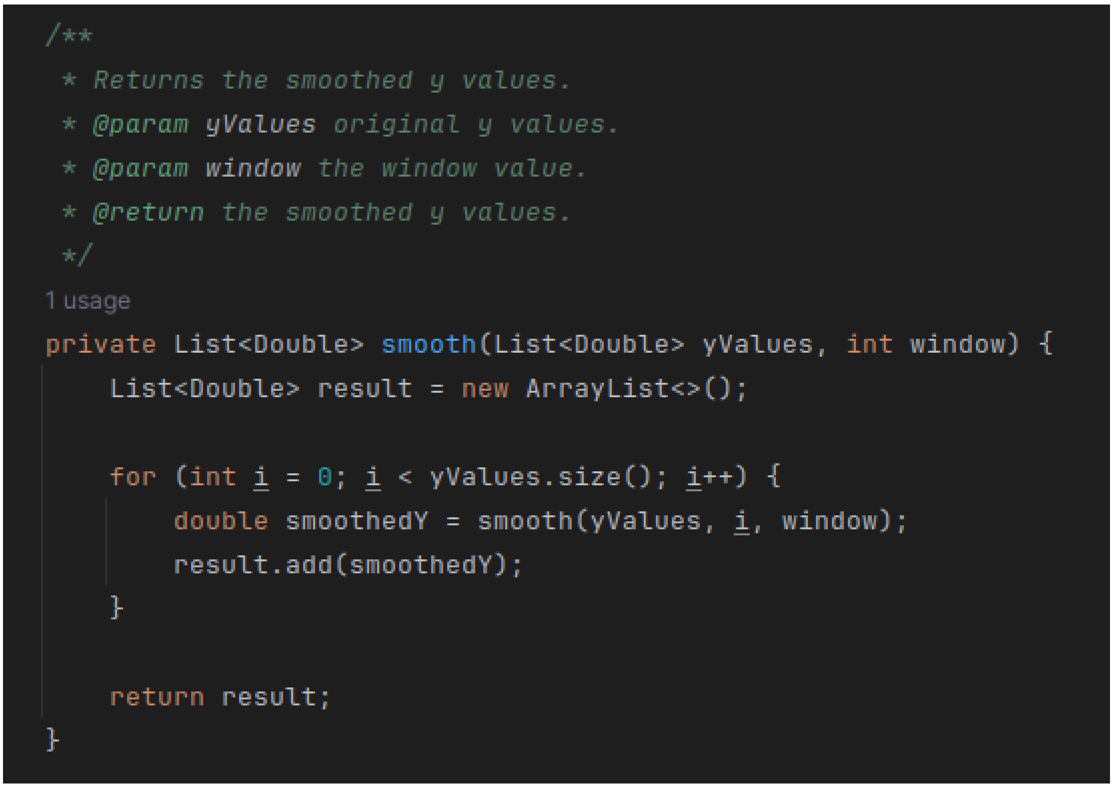
Smoothing

The **Smoother** class contains the code for smoothing out the data. The **smooth** method accepts the filename and the window and creates a new file with smoothed y values:



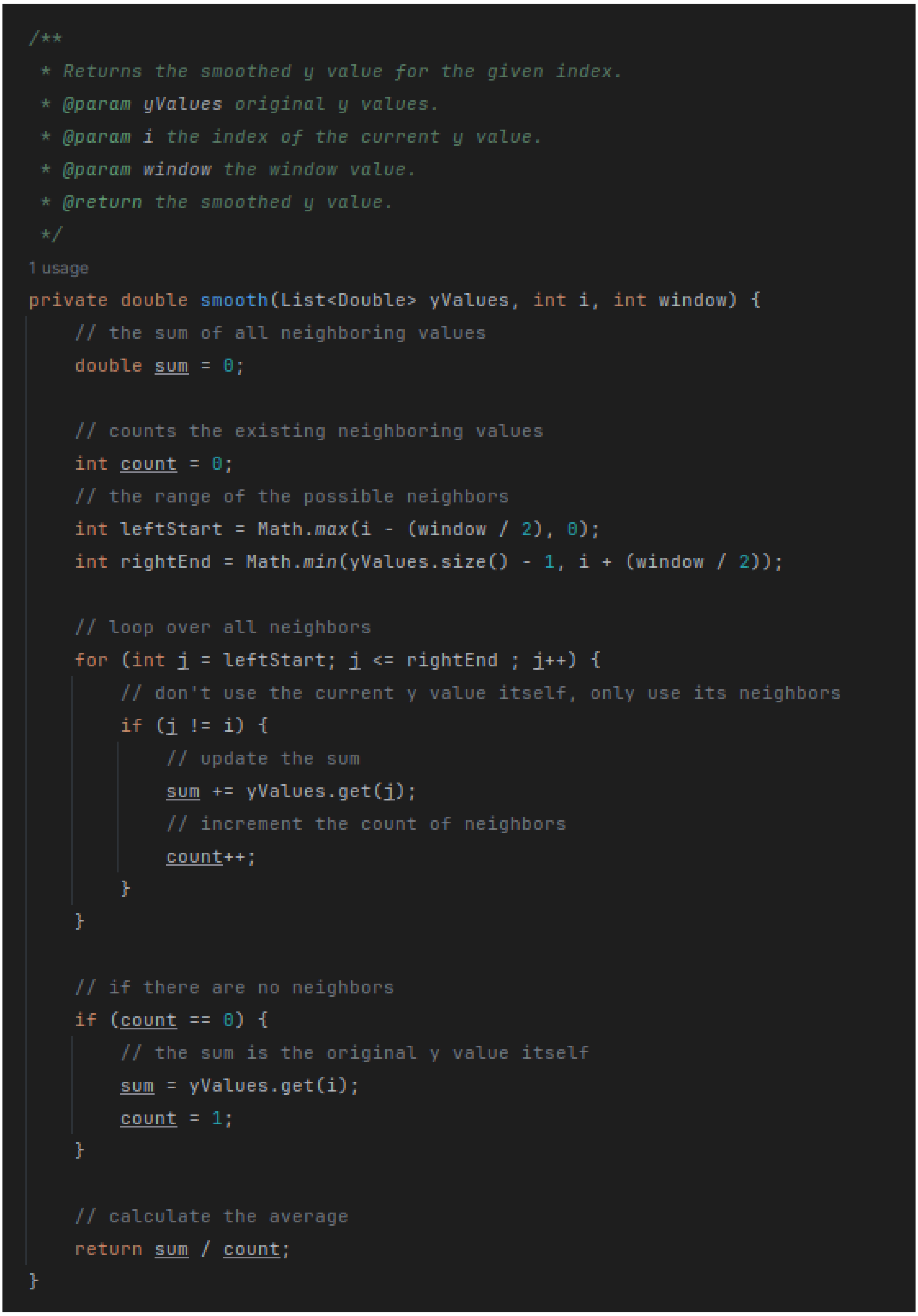
```
/**
 * Smooths salted data.
 */
public class Smoother {
    /**
     * Smooths the y values in the given file.
     * @param filename the name of the file.
     * @param window the window value.
     */
    1 usage
    public void smooth(String filename, int window) {
        try (Scanner scanner = new Scanner(new File(filename))) {
            List<Double> xValues = new ArrayList<>();
            List<Double> yValues = new ArrayList<>();

            String header = scanner.nextLine() + ",smooth window = " + window;

            while (scanner.hasNextLine()) {
                String line = scanner.nextLine();
                String[] data = line.split( regex: ",");

                double x = Double.parseDouble(data[0]);
                double y = Double.parseDouble(data[1]);

                xValues.add(x);
                yValues.add(y);
            }

            List<Double> smoothedYValues = smooth(yValues, window);
            createDataFile( filename: "smoothed-" + filename,
                    xValues, smoothedYValues, header);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

The overloaded **smooth** method accepts the original y values and the window value and returns smoothed y values:



```java
/**
 * Returns the smoothed y values.
 * @param yValues original y values.
 * @param window the window value.
 * @return the smoothed y values.
 */
1 usage
private List<Double> smooth(List<Double> yValues, int window) {
    List<Double> result = new ArrayList<>();

    for (int i = 0; i < yValues.size(); i++) {
        double smoothedY = smooth(yValues, i, window);
        result.add(smoothedY);
    }

    return result;
}
```

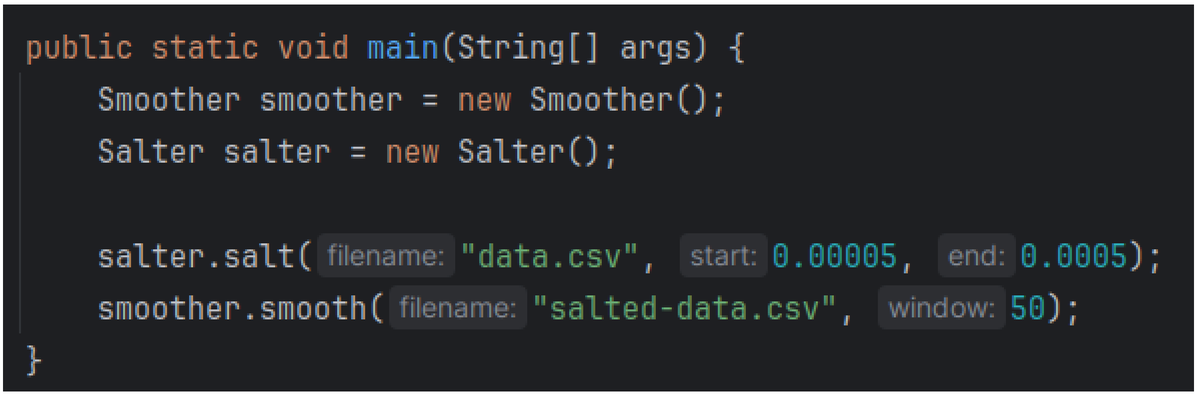
Another overloaded **smooth** method accepts the original y values, the index of the current y value, and the window value, and returns the smoothed y value:



```java
/**
 * Returns the smoothed y value for the given index.
 * @param yValues original y values.
 * @param i the index of the current y value.
 * @param window the window value.
 * @return the smoothed y value.
 */
1 usage
private double smooth(List<Double> yValues, int i, int window) {
    // the sum of all neighboring values
    double sum = 0;

    // counts the existing neighboring values
    int count = 0;
    // the range of the possible neighbors
    int leftStart = Math.max(i - (window / 2), 0);
    int rightEnd = Math.min(yValues.size() - 1, i + (window / 2));

    // loop over all neighbors
    for (int j = leftStart; j <= rightEnd ; j++) {
        // don't use the current y value itself, only use its neighbors
        if (j != i) {
            // update the sum
            sum += yValues.get(j);
            // increment the count of neighbors
            count++;
        }
    }

    // if there are no neighbors
    if (count == 0) {
        // the sum is the original y value itself
        sum = yValues.get(i);
        count = 1;
    }

    // calculate the average
    return sum / count;
}
```

The **createDataFile** is taken as is from the **Salter** class.

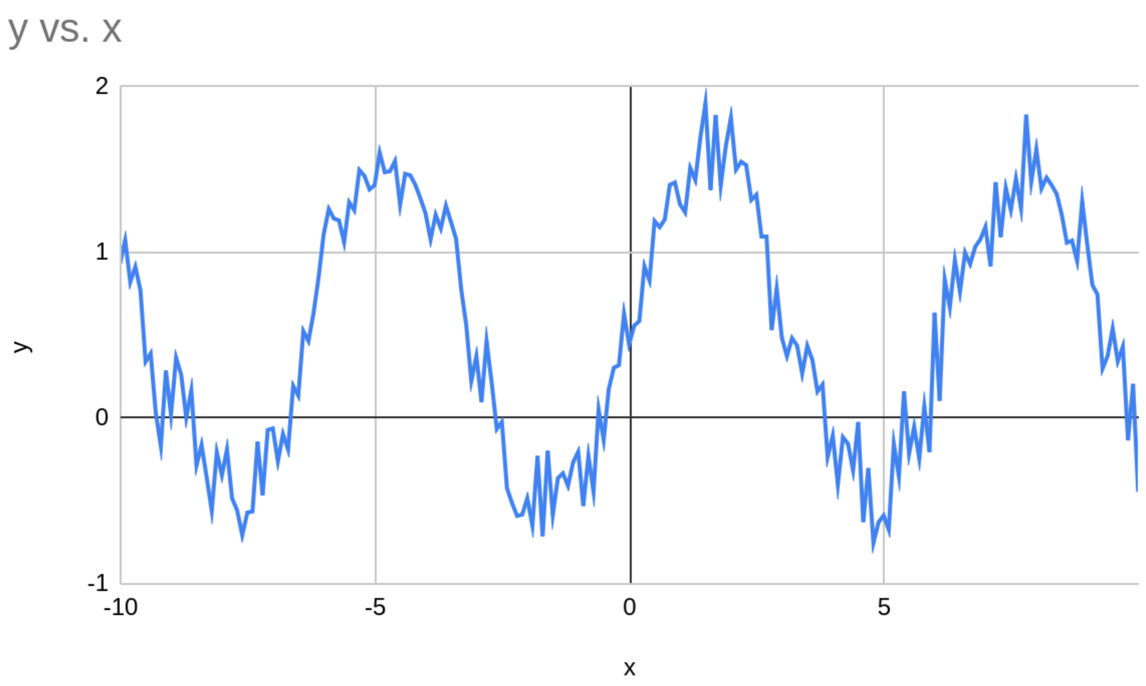
The **main** method is used to first generate a salted file, and then smooth it:



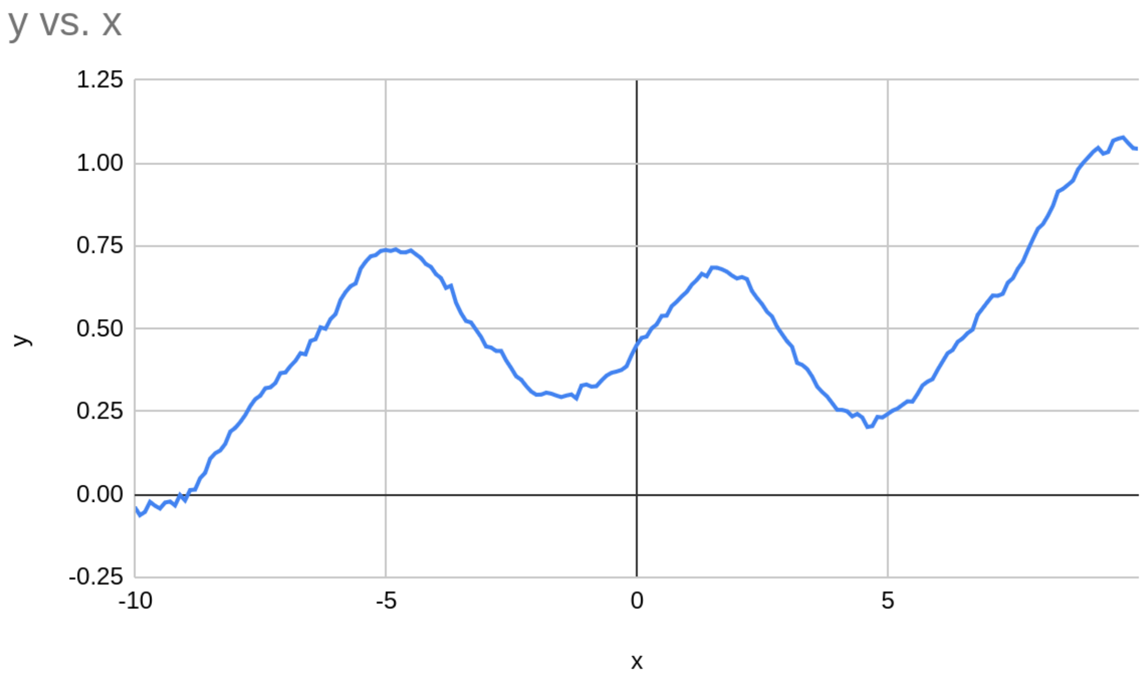
```
public static void main(String[] args) {
    Smoother smoother = new Smoother();
    Salter salter = new Salter();

    salter.salt( filename: "data.csv",  start: 0.00005,  end: 0.0005);
    smoother.smooth( filename: "salted-data.csv",  window: 50);
}
```

Smoothing **data.csv**

salt range: [0.00005;0.0005]
smooth window = 3



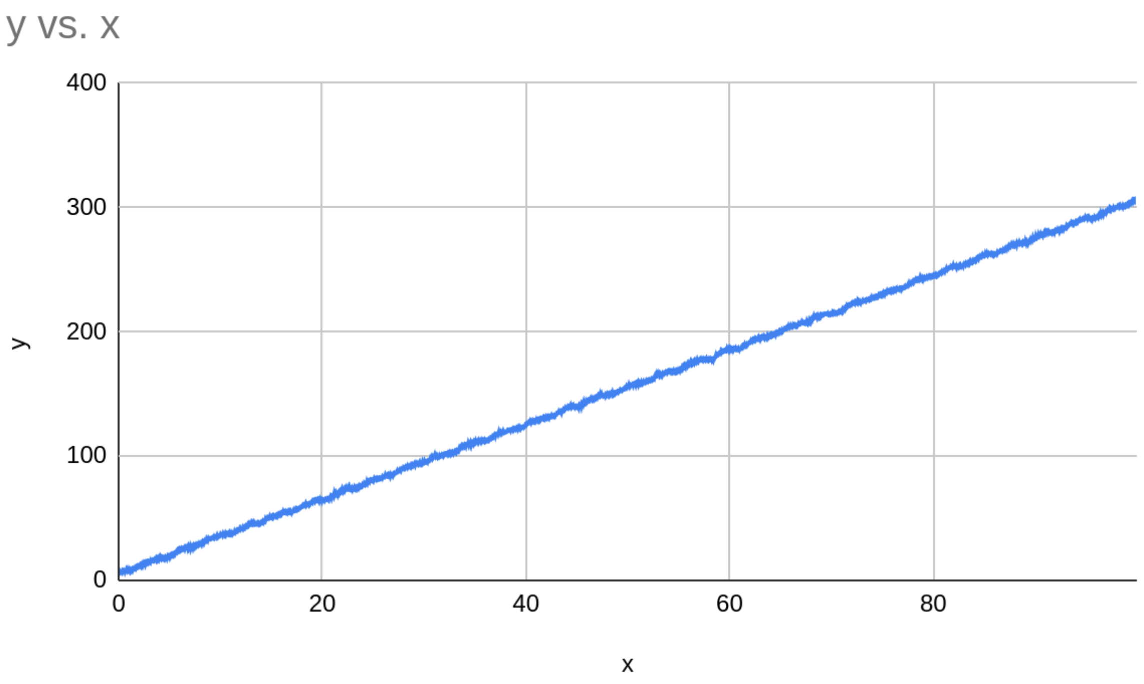
salt range: [0.00005;0.0005]
smooth window = 50



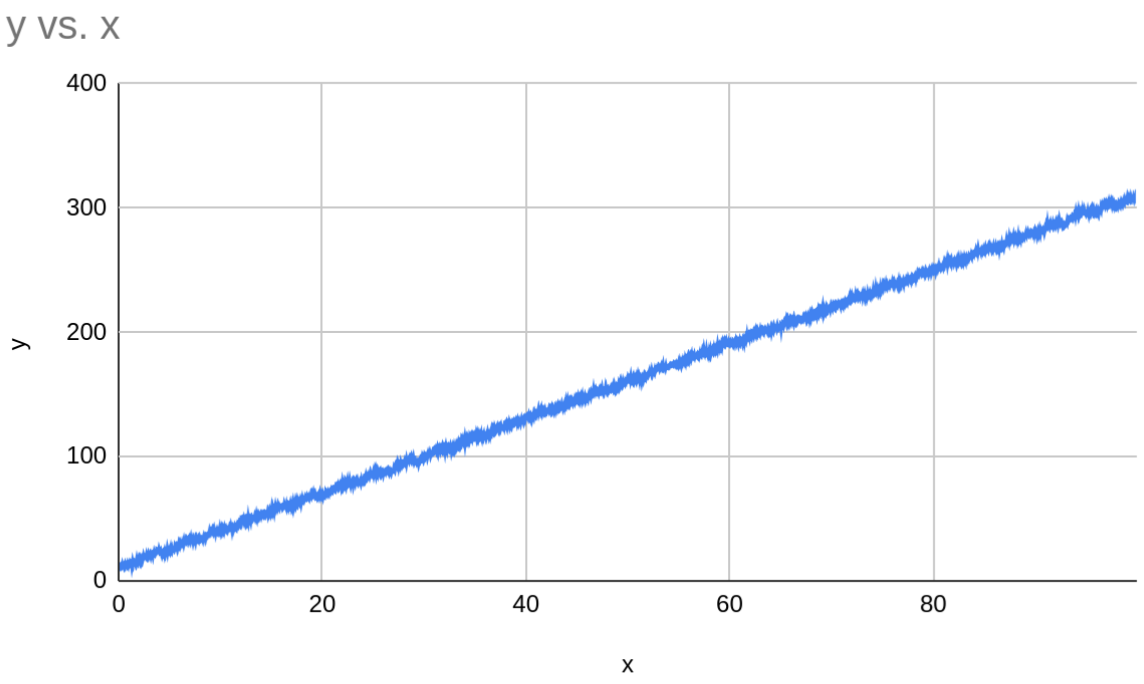
The larger the smooth window, the more smoothed out the chart is - because the data is more averaged out.

Smoothing **test.csv**

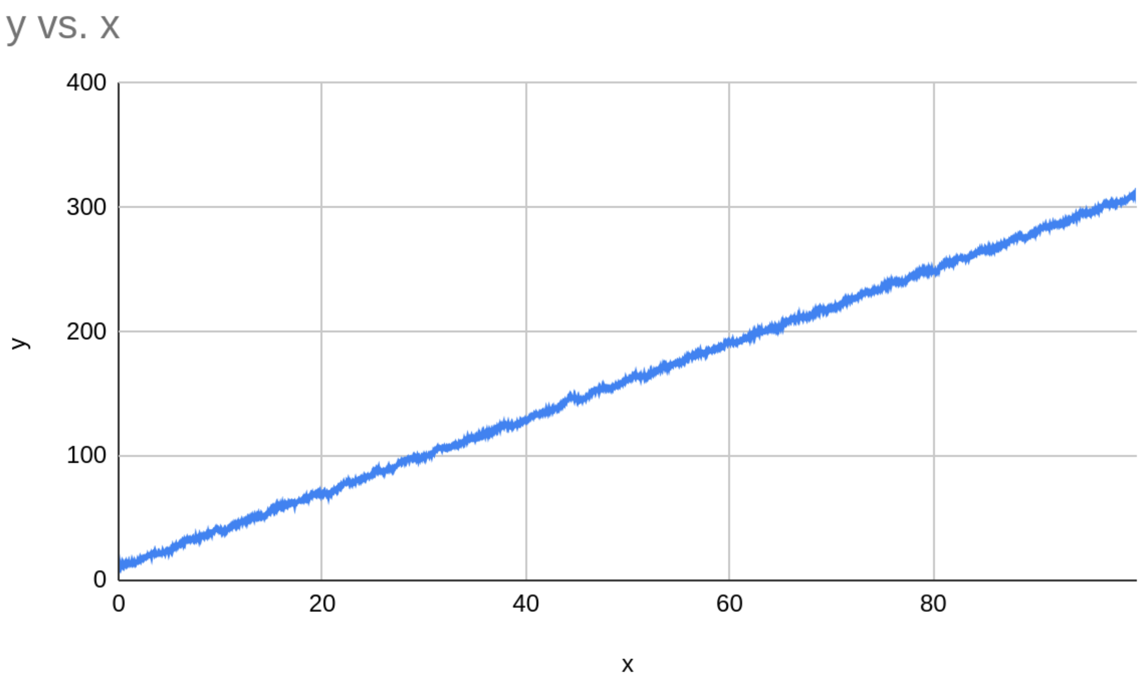
salt range: [0;10]
smooth window = 50



salt range: [0;20]
smooth window = 50



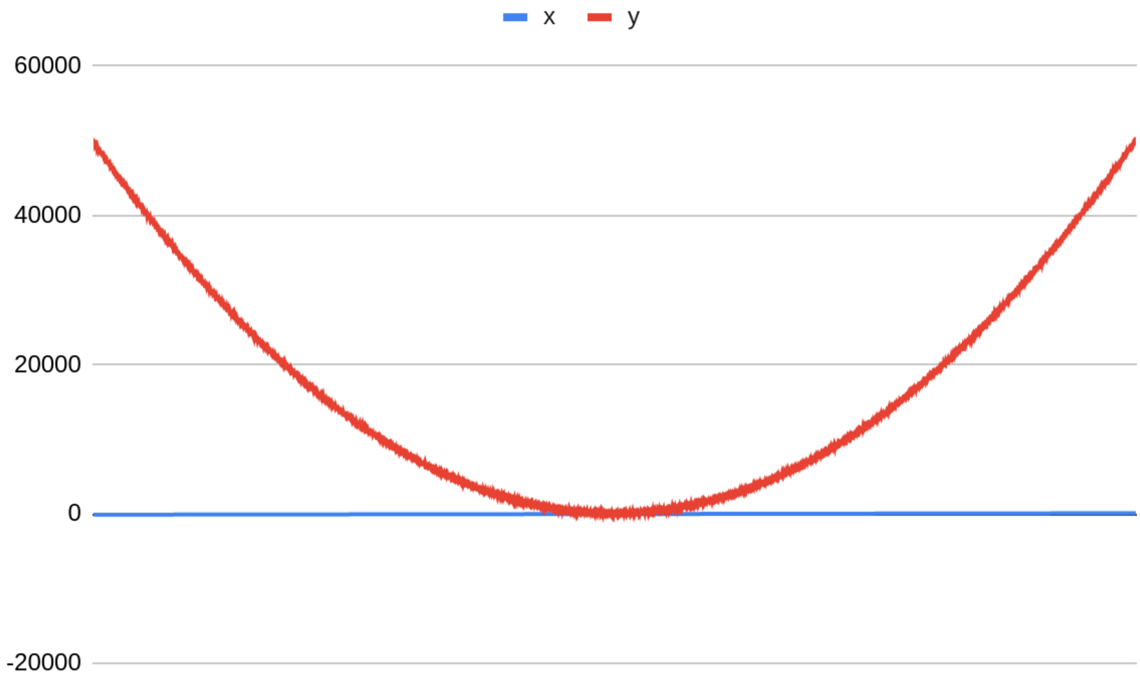
salt range: [0;20]
smooth window = 70



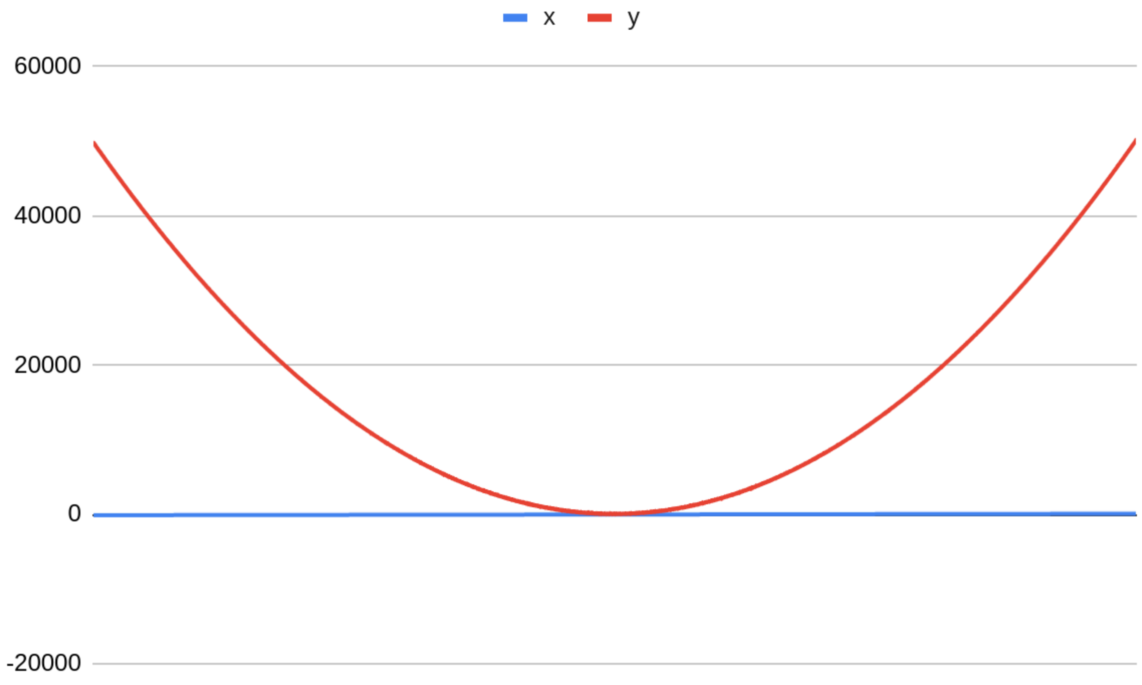
Again, increasing the window value makes the chart more smooth.

Smoothing **extra.csv**

salt range: [1;100]
smooth window = 5



salt range: [1;100]
smooth window = 20



Increasing the smooth window from 5 to 20 made the chart completely smooth.

Ways to make the programs better

The program could be bettered in the following ways:

- Generate files with unique names, that include the configuration information (window value, salt range) that help with finding the files with specific configuration faster
- Create an extra class that accepts all the possible parameters (function, salt range, window value, etc) and goes through all the steps automatically: generating, salting and smoothing data
- Make it possible to read the parameters from another csv file and generate all data automatically, so that the parameters could be dynamic without any need to change the code in the **main** methods