COL216 Assignment-1

Dhruv Kumar Gupta 2019CS10346
Swaraj Gawande 2019CS10406

February 2021

1 Design Overview

1.1 Input and Output

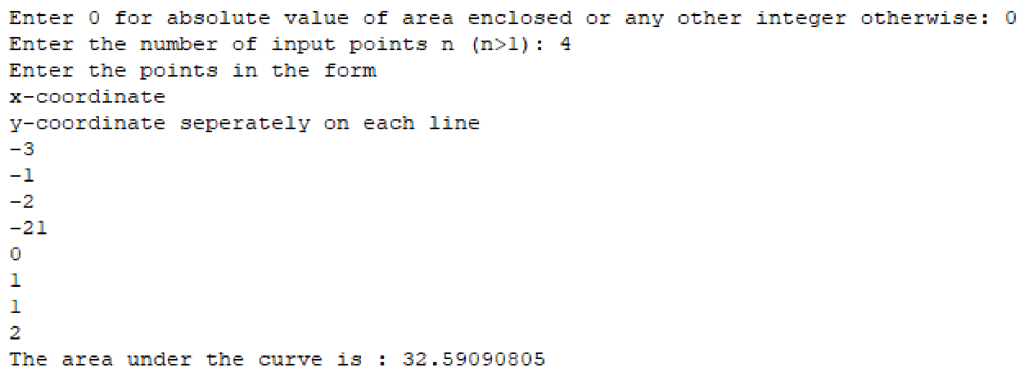
The Program when run, asks for two integers.

- **Mode** : Used to determine sign to be taken for negative areas.
- **Num** : Number of coordinates which will be entered

The program waits for **num** number of coordinates with x and y coordinates in different line.

After the input of all **num** points, the program prints the area in floating point.

Example screen :



```
Enter 0 for absolute value of area enclosed or any other integer otherwise: 0
Enter the number of input points n (n>1): 4
Enter the points in the form
x-coordinate
y-coordinate seperately on each line
-3
-1
-2
-21
0
1
1
2
The area under the curve is : 32.59090805
```

1.2 Mode

Due to 2 polarised opinions on Piazza on the decision of sign of area below x axis, we decided to provide both the options to the user.
The program asks for an integer input at the start. If the input is 0, the areas below the x axis are taken as positive. For any other input, they are taken as negative. This is made possible by the approach used by us explained below.

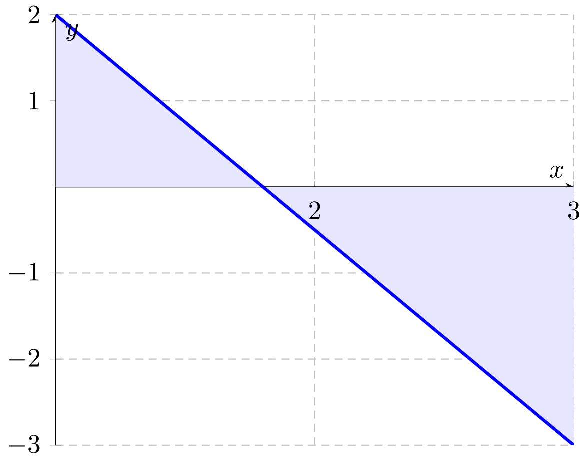
1.3 Overflow choices

The program uses standard integer registers and float registers and therefore handles upto $2^{31}-1$, accounting for sign bit. The values stored in float registers would be less than or equal to integer registers according to the approach. This range is reasonable for our particular use as the inputs are to be taken from keyboard and not a file, and hence would be smaller. In our approach we are only checking for overflow in the registers where it is most likely to happen with inputs which are well within range.

2 Explanation of approach and code

2.1 Mathematical description

Consider the diagram :



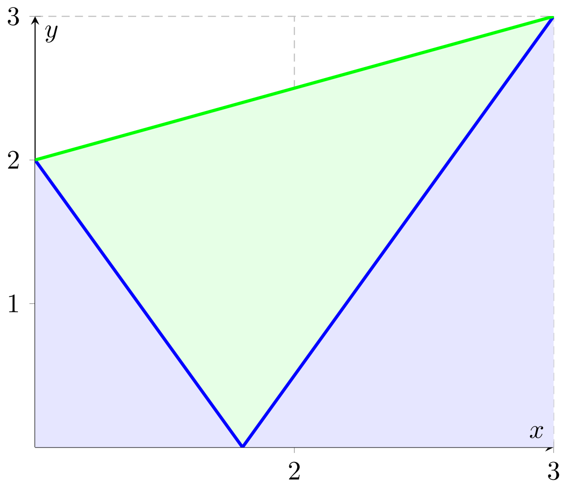
There are two modes and hence 2 different ways to calculate the area.

When the area below x axis is considered negative, then a simple trapezium formula, that is

$$Area = \frac{(y_1 + y_2)}{2}(x_2 - x_1)$$

where, $y_1 = 2$, $y_2 = -3$, $x_1 = 1$ and $x_2 = 3$ according to diagram.

When the absolute values of the area is needed to be added, then, marking the absolute value gives :



The required blue area can be though of as, area of trapezium (green + blue) - green area. The green area is given by the formula

$$area = \frac{|y_1||y_2|}{|y_1| + |y_2|}(x_2 - x_1)$$

while the trapezium area is given by trapezium formula.

This shows that same trapezium formula can be used in both cases while subtracting green areas at the end. This is the approach used to calculate area while doing minimum amount of floating points calculations to reduce error.

2.2 Code explanation

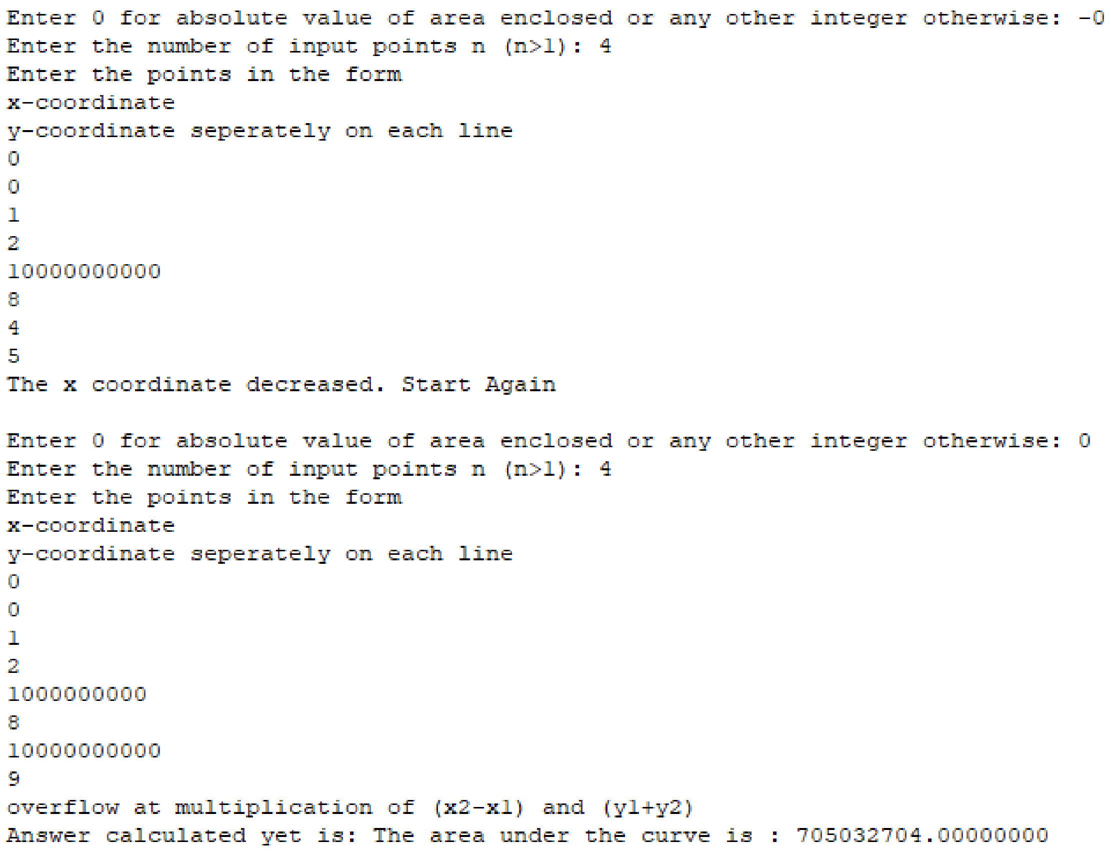
The code keeps track of mode, total coordinates, and looping variable in registers **s0** , **s1** , **s2** respectively. The new coordinates are stored in registers **t1** and **t2** and older coordinates are stored in registers **s3** and **s4**.

Depending on mode register **s0**, sum of either absolute values of y coordinates are stored or simple sum is stored in register **t7**. In both modes, register **s5** contains the sum of products of value in **t7** and difference of x coordinates upto that iteration. This will be an integer value as per the specifications.

If mode is for absolute value and a cross is encountered where one y coordinate is positive while the other is negative, then the green area is calculated and the sum of all green areas is kept in a float register **f12**.

At the end of all iterations, the integer sum in **s5** is converted to float and divided by 2 to get the sum of trapezium areas and the green area sum in **f12** is subtracted and stored in **f12** only as that register is used for display.

There are overflow checks after some operations which are susceptible to overflow and which might get ignored by QTspim. The check is based on standard overflow checking as was explained in COL215. A check is being done when we multiply (x2-x1) and sum of y coordinates and another when we add this product to the integer area sum. While the prior is actually helpful the later is is actually caught by QTspim already. The program also checks for decreasing x coordinates and gives an error prompting the user to start again.[1] The output during exceptions and errors are :



```
Enter 0 for absolute value of area enclosed or any other integer otherwise: -0
Enter the number of input points n (n>1): 4
Enter the points in the form
x-coordinate
y-coordinate seperately on each line
0
0
1
2
10000000000
8
4
5
The x coordinate decreased. Start Again

Enter 0 for absolute value of area enclosed or any other integer otherwise: 0
Enter the number of input points n (n>1): 4
Enter the points in the form
x-coordinate
y-coordinate seperately on each line
0
0
1
2
1000000000
8
10000000000
9
overflow at multiplication of (x2-x1) and (y1+y2)
Answer calculated yet is: The area under the curve is : 705032704.00000000
```

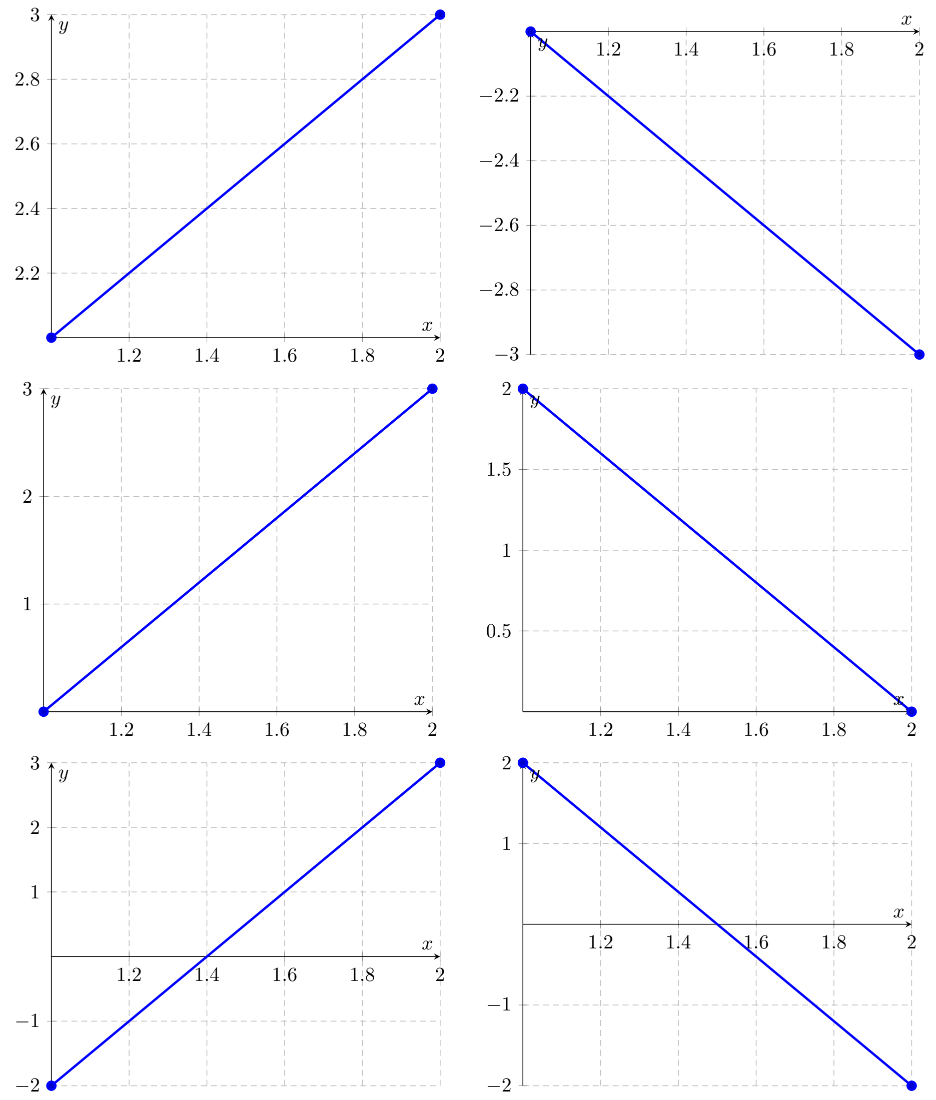
[1]Note : Equal x coordinates are considered as valid input, as they indicate zero area and represent discontinuity in graphs which is completely valid according to specifications.

3 Testing

The testing was done component and feature wise.

The loop in the program was first tested to ensure it runs exactly **num** number of times by simply writing a test string everytime the loop runs.

To ensure persistence of register values, the program was extended to calculate area just based on the trapezium formula. This was tested with the following kind of cases for 2 points along the case of $y_1 + y_2 = 0$:



After testing **combinations** of above cases, and comparing the output by making same graphs on desmos, the program was extended to all the features.

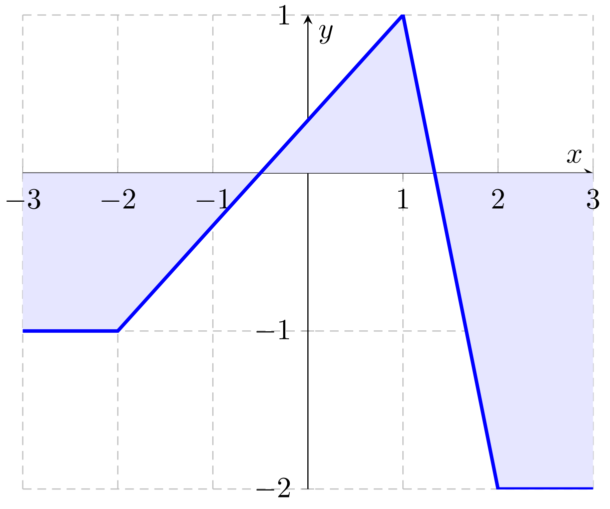
To ensure that addition of new mode and calculations did not disrupt the original formula, same testcases were run with the appropriate mode as in the previous tests. The same testcases were then run with mode 0, to ensure the outputs match with the absolute values of area.

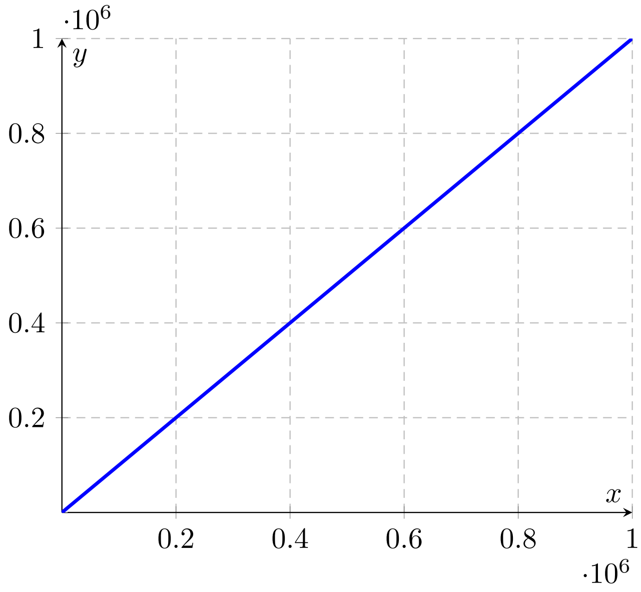
An error was found, which was due to using the same register for two values, which was found using single step feature of QTSpim, which was rectified leading to correct answer in all kinds of combinations.

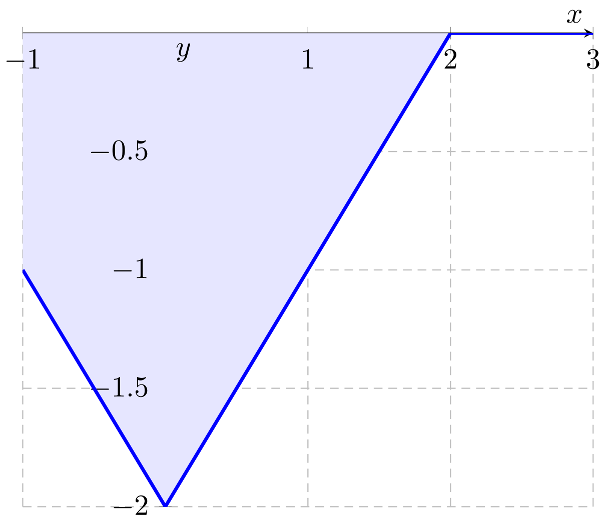
Exceptions were introduced aftwerwards for overflow, which were tested using large differences in x coordinates or large values of y coordinates. Some overflows are handled by QTSpim itself, while the other overflows in operations like multiplication were handled by either restarting the process or displaying the area upto the previous point.

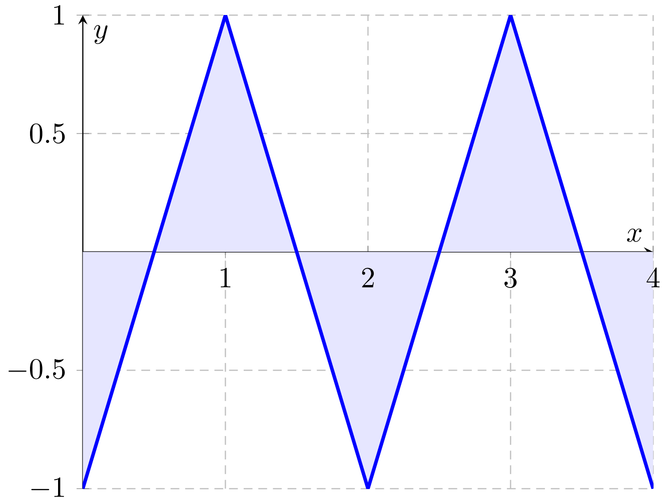
Errors like division by zero are not applicable as division by a variable only happens when $y_1 y_2 < 0$ which implies $|y_1| + |y_2| > 0$

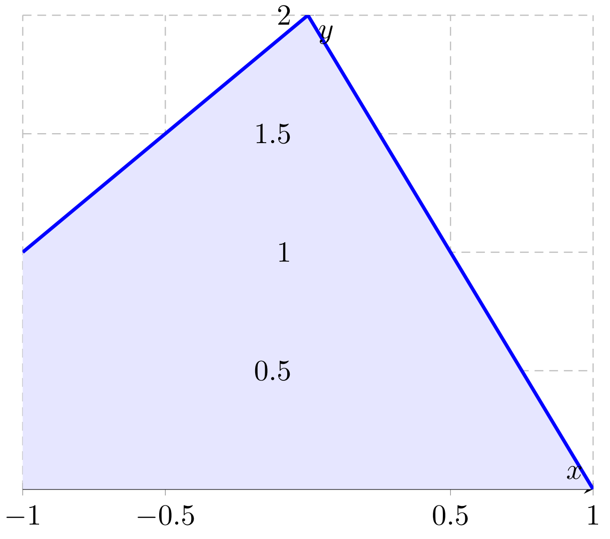
3.1 Some Testcases visualised[2]

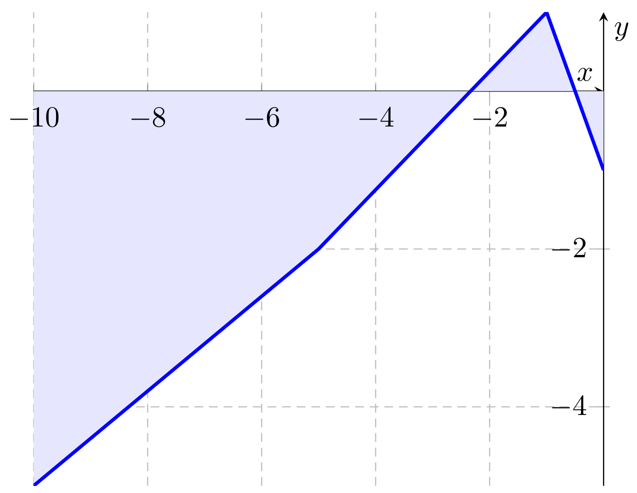












[2]The second case led to overflow.