CV
Project 3: bi-Mean Gaussian Automatic Threshold Selection

Adrian Noa

Due 10/2/2022

My Algorithms:

loadHistogram(&inFile):
This method loads the histogram into an array and return the highest pixel count from an in-file stream address

1) Max ← -999
2) **Loop**: i → 0 to maxVal+1 times
   a) numOfPixels ← inputFileRead (twice)
   b) array[i] ← numOfPixels
   c) if numOfPixels > max
      i) max ← numOfPixels
3) Return max

plotGraph(symbol):
The purpose of this algo is to fill a 2D array [maxHeight+1][maxVal+1] with a symbol, which visually represents the 1D histogram or gaussian curve

1) Loop: i → 0 to maxVal+1
2) Graph[maxHeight][i] ← '_' : set the last row to represent the x-axis
   a) **if** [i] >=0 **then:**
      i) **Loop**: j=maxHeight-array[i] → maxHeight
         (1) Graph[j][i] → symbol

addVertical(graph, thr):
This function adds a '|' symbol to the 2D character array on every row

1. Loop j → 0 to maxHeight:
   a. Graph[j][thr] = '|'

setZero(array):
Sets all positions in a 1D integer array to zero

1. **Loop**: i → 0 to maxVal+1:
   a. array[i] ← 0

plotAll(&outfile, thr):
This algorithm adds a graph to an out-file stream. The graph is the gaussian curve with the histogram overlaid, and a display of the threshold

1. drawTitle()
2. plotOverlay()

drawTitle(&outFile, string):
Used to caption all graphs with a given string, with the use of the maxVal(60-63) of the graph multiplied by 2

1. length ← ((maxVal*2+1)/2)-1
2. strLength ← length – string.length()/2
3. **Loop**: r → 0 to 3:
    a. **Loop**: i → 0 to (maxVal+1)*2:
        i. **if** first row **then** draw '*'
        ii. **if** second row:
            1. **if** i == strLength – 1 **then** draw padding(2), string, padding(2)
            2. **else if** i <= strLength -1 **then** draw '*'
            3. **else if** i > strLength + string.length()+2 **then** draw '*'
        iii. if third row ← draw '*'

plotOverlay(&outFile, thr):
Plots the gaussian curve with the histogram overlayed and adds the threshold line at column thr.

1. overlay ← Allocate 2D dynamic array, set all values to ' ', except for last row, set to '-' indicating x-axis
2. **Loop**: i → 0 to maxVal+1:
    a. **Loop**: j → maxHeight – GaussAry[i] to maxHeight: draw Gaussian curve
        i. overlay[j][i] ← gaussian curve symbol
    b. **Loop**: j → maxHeight – histAry[i] to maxHeight: draw Histogram on top
        i. **if** overlay[j][i] == gaussian symbol **then** overlay[j][i] ← 'O'
        ii. **else if** overlay[j][i] is empty **then** overlay[j][i] ← '-'
3. addVertical(overlay, thr)
4. drawToFile ← 2 loops that print to the outFile
5. deleteOverlay ← free resources granted to 2D dynamic array, the main array and the subarrays

```
/*

Created by Adrian Noa

Objective:

To take an image's histogram from a text and find the best threshold using
the bi-Gaussian automatic threshold selection method.

This process yields a Gaussian function which will be graphed along with the
histogram, and the best threshold.

Usage:

g++ noa_adrian_main.cpp -o main.exe && ./main.exe BiGuass_data2.txt
outFile1.txt outFile2.txt

*/
```

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <math.h>
#include <string>
using namespace std;

class BiMean{

public:

    int numRows, numCols, minVal, maxVal;
    int maxHeight = 0; // largest histAry[i]
    int maxGval; // maximum calculated distribution value;
    int offSet; // one-tenth of the maxVal-minVal
/*
        in bimodal histogram, the first modal occupies at least one-tenth of
        the histogram population from minVal to maxVal of the histogram
*/
    int dividePt; // Initialy set of offset, increases by 1 each iteration.
/*
        Selected treshold value is at the point at divedePt where the "distance"
        between the two bi-Gaussian curves and the histogram is the minimum
*/

    // All arrays need to be dynamically allocated at run-time
    int* histAry;       // 1D[maxVal+1] to store Histogram Array

    int* GaussAry;      // 1D[maxVal+1] to store "modified" Gaussian function

    char** histGraph;   // 2D[maxVal+1 x maxHeight+1] initialize to "blank"
/*        for displaying the histogram curve.       */

    char** GaussGraph;  // 2D[maxVal+1 x maxHeight+1] initialize to "blank"
/*        for displaying Gaussian curves in 2D.       */

    BiMean(ifstream &inFile){
    // BiMean(int numRows, int numCols,int minVal,int maxVal){

    inFile >> numRows >> numCols >> minVal >> maxVal;
        histAry = new int[maxVal+1]();
        cout << "Allocated 1D Histogram Array" << endl;
    }

    int loadHist(ifstream &inFile){ // add histogram to histAry from inFile and returns the max among hist[i]
        int numberOfPixels;
        int max = -999;
        for (int i = 0; i<maxVal+1; i++){
            inFile >> numberOfPixels >> numberOfPixels;
            histAry[i] = numberOfPixels;
            max = (numberOfPixels > max) ? max = numberOfPixels : max;
        }
        return max;
    }

    void allocate(){

        this->GaussAry = new int[maxVal+1]{0};
        cout << "Allocated 1D Gauss array" << endl;

        this->histGraph = new char*[maxHeight+1];
        cout << "Allocated Histogram 2D char array" << endl;

        this->GaussGraph = new char*[maxHeight+1];
        cout << "Allocated Gaussian graph 2D char array" << endl;

        for(int i=0; i<maxHeight+1;i++) {
            histGraph[i] = new char[maxVal+1]();
            GaussGraph[i] = new char[maxVal+1]();
            for (int j = 0; j < maxVal+1; j++) {
                histGraph[i][j]=' ';
                GaussGraph[i][j]=' ';
            }
```

```
        }
        cout << "Allocated histogram and gaussian subarrays" << endl;
    }

    void plotGraph(int* ary, char** graph, char symbol){
        int j;
        for (int i = 0; i < maxVal+1; i++){
            graph[maxHeight][i] = '_';
            if(ary[i] >= 0){
                for (j = maxHeight-ary[i]; j < maxHeight; j++) {     // display
                    graph[j][i] = symbol;                            // as
                }                                                    // line graph
            }
        }
    }

    void addVertical(char** graph, int thr){
        for (int j = 0; j < maxHeight; j++) {
            graph[j][thr] = '|';
        }
    }

    double computeMean(int leftIndex, int rightIndex, int maxHeight){
        int sum = 0;
        int numPixels = 0;
        maxHeight = 0;

        for (int i = leftIndex; i<rightIndex; i++){
            sum += histAry[i]*i;
            numPixels += histAry[i];
            if(histAry[i]>maxHeight){
                maxHeight = histAry[i];
            }
        }
        return double(sum)/(double)numPixels;
    }

    double computeVar(int leftIndex, int rightIndex, double mean){
        double sum = 0.0;
        int numPixels = 0;
        for (int i=leftIndex ; i < rightIndex ; i++){
            sum += (double)histAry[i] * pow((double)i-mean,2);
            numPixels += histAry[i];
        }
        return (double)sum/(double)numPixels;
    }

    double modifiedGauss(int x, double mean, double var, int maxHeight){
        //G(X) = maxHeight * exp( - ( (x-mean)^2 / (2* c2) )
        double G = (double)maxHeight * exp(-1 * pow(x-mean, 2) / (2*var));
        return G;
    }

    void setZero(int* ary){
        for (int i = 0; i < maxVal+1; i++) {
            ary[i]=0;
        }
    }

    int biMeanGauss(int dividePt, ofstream &outFile){
        outFile << "DividePt" << "\tLeftSum" << "\t\tRightSum" << "\tTotalSum" << "\tDifference" <<
"\tBestThreshold" << endl;
        int bestThr = dividePt;
        double sum1;
        double sum2;
        double total;
        double minSumDiff = 99999.9;
        while( dividePt < (maxVal - offSet)){
            setZero(GaussAry);
            sum1 = fitGauss(0, dividePt, GaussAry);
            sum2 = fitGauss(dividePt, maxVal, GaussAry);
            total = sum1 + sum2;
            if(total<minSumDiff) {
```

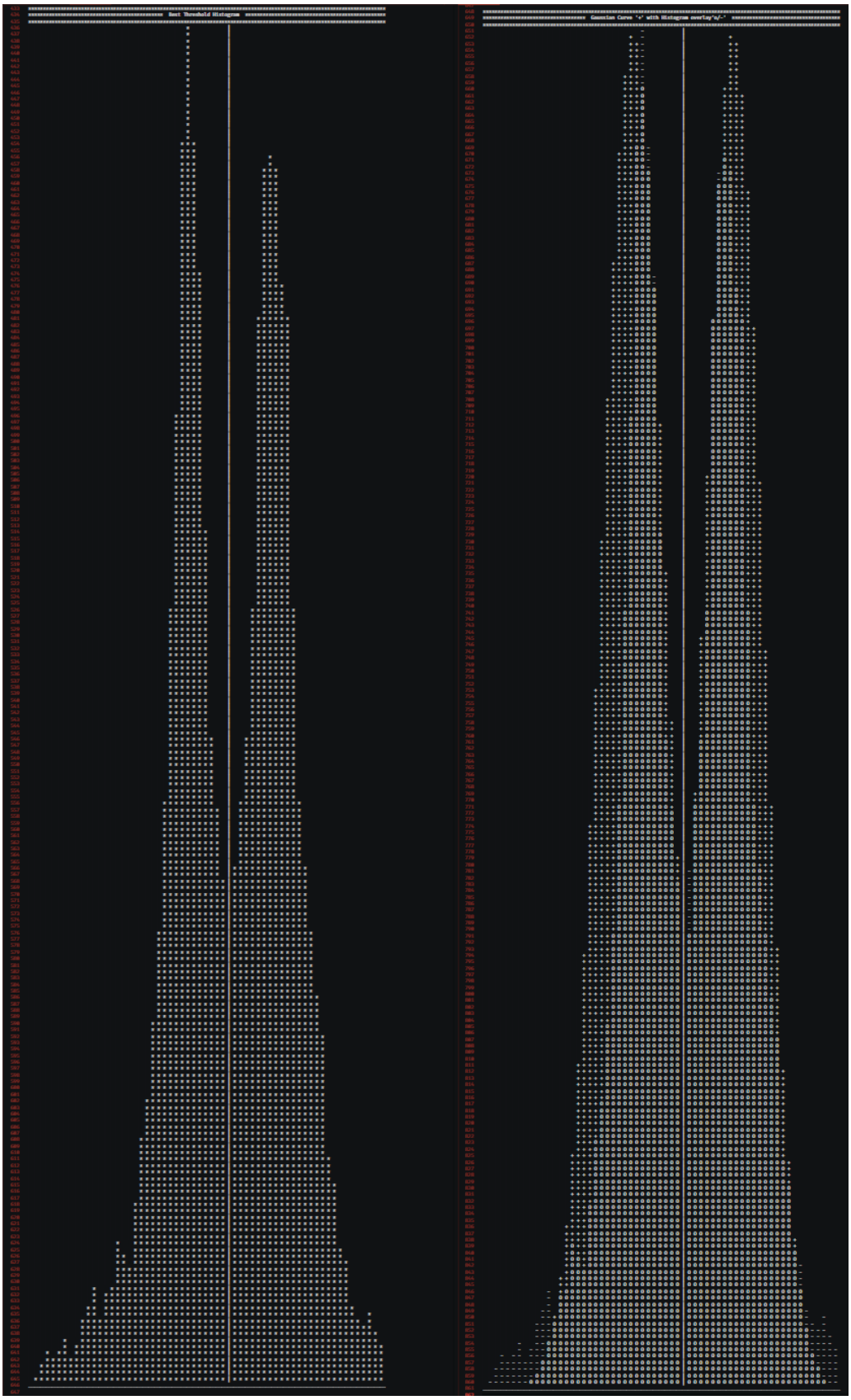
```
                minSumDiff = total;
                bestThr = dividePt;
            }
            toFile(outFile, dividePt, sum1, sum2, total, minSumDiff, bestThr);
            dividePt++;
        }
        return bestThr;
    }

    void toFile(ofstream &outFile, int dividePt, double sum1, double sum2, double total, double minSumDiff, int
bestThr){
        outFile << "\t" << dividePt
                << "\t\t" << sum1
                << "\t\t" << sum2
                << "\t\t" << total
                << "\t\t" << minSumDiff
                << "\t\t\t" << bestThr << endl;
    }

    double fitGauss(int leftIndex, int rightIndex, int* GaussAry){
        double mean, var, Gval, maxGval, sum = 0.0;
        mean = computeMean(leftIndex, rightIndex, maxHeight);
        var = computeVar(leftIndex, rightIndex, mean);
        for (int i = leftIndex; i <= rightIndex; i++){
            Gval = modifiedGauss(i, mean, var, maxHeight);
            sum += abs(Gval - (double)histAry[i]);
            GaussAry[i] = (int)Gval;
        }
        return sum;
    }

    void bestFitGauss(int bestThrVal){
        double sum1, sum2;
        setZero(GaussAry);
        sum1 = fitGauss(0, bestThrVal, GaussAry);
        sum2 = fitGauss(bestThrVal, maxVal, GaussAry);
    }

    void bestThr(ofstream &outFile, int bestThrVal){
        outFile << "Best Threshold value: " << bestThrVal << endl << endl;
    }

    void drawGraph(ofstream &outFile, char** graphAry, string str, int* ary){
        drawTitle(outFile, str);
        for (int i = 0; i < maxHeight+1; i++){
            for (int j = 0; j < maxVal+1; j++){
                if(i == maxHeight) outFile << graphAry[i][j] << graphAry[i][j];
                else outFile << graphAry[i][j] << " ";
            } outFile << endl;
        } outFile << endl;
    }

    void plotAll(ofstream &outFile, int bestThr){
        drawTitle(outFile, "Gaussian Curve '+' with Histogram overlay'o/-'");
        plotOverlay(outFile, bestThr);
    }

    void drawTitle(ofstream &outFile, string str){
        int l = ((maxVal*2+1)/2)-1;
        int sl = l - str.length()/2;
        for(int r=0; r<3; r++){
            for (int i = 0; i < (maxVal+1)*2; i++){
                if(r==0) outFile << "*";
                else if(r==1){
                    if(i==sl-1) outFile << "  " << str << "  ";
                    else if(i<=sl-1) outFile << "*";
                    else if (i>sl+str.length()+2) outFile << "*";
                } else outFile << "*";
            } outFile << endl;
        }
    }

    void plotOverlay(ofstream &outFile, int bestThr){
```

```
        char** overlay = allocateOverlay();

        for (int i = 0; i < maxVal+1; i++) {            // draw Gaussian Curve

            for (int j = maxHeight-GaussAry[i]; j < maxHeight; j++) overlay[j][i] = '+';

            for (int j = maxHeight-histAry[i]; j < maxHeight; j++){  // draw Histogram
                if(overlay[j][i] == '+') overlay[j][i] = 'O';
                else if (overlay[j][i] == ' ')overlay[j][i] = '-';
                // else overlay[j][i] = '^';
            }
        }

        addVertical(overlay, bestThr);
        drawToFile(outFile, overlay);
        deleteOverlay(overlay);
    }

    char** allocateOverlay(){
        char **overlay = new char*[maxHeight+1];
        for(int j=0; j<maxHeight+1; j++) {
            overlay[j] = new char[maxVal+1];
            for (int i = 0; i < maxVal+1; i++){
                if(j==maxHeight) overlay[j][i] = '_';
                else overlay[j][i] = ' ';
            }
        }
        cout << "allocated 2d for overlay" << endl;
        return overlay;
    }

    void drawToFile(ofstream &outFile, char** graph){
        for (int i = 0; i < maxHeight+1; i++){
            for (int j = 0; j < maxVal+1; j++){
                if(i==maxHeight) outFile << graph[i][j] << graph[i][j];
                else outFile << graph[i][j] << " ";
            }
            outFile << endl;
        }
    }
    void deleteOverlay(char** graph){
        for (int i = 0; i < maxHeight+1; i++) delete[] graph[i];
        delete[] graph;
    }

    ~BiMean(){
        delete[] histAry;
        delete[] GaussAry;
        cout << "deleted histogram and gaussian dynamic arrays" << endl;
        for(int i=0; i<maxHeight+1; i++){
            delete[] histGraph[i];
            delete[] GaussGraph[i];
        }
        cout << "delete all subarrays in histgraph and gausGraph" << endl;
        delete[] histGraph;
        cout << "deleting histGraph memory allocation" << endl;
        delete[] GaussGraph;
        cout << "deleting gausGraph memory allocation";
    }

};

int main(int argc, const char* argv[]) {
    cout << endl;

    if (argc != 4){
        printf("Not enough arguments\n");
        return 1;
    }

    ifstream inFile(argv[1]);
    ofstream outFile1(argv[2]);
```

```
    ofstream outFile2(argv[3]);

    if (!inFile.is_open()) {
        cout << "Unable to open file" << endl;
        exit(2);
    }

    BiMean biMean = BiMean(inFile);

    // int numRows, numCols, minVal, maxVal;
    biMean.maxHeight = biMean.loadHist(inFile);

    biMean.allocate();

    // imageProcessing.shout();
    biMean.plotGraph(biMean.histAry, biMean.histGraph, '*');
    biMean.drawGraph(outFile1, biMean.histGraph, "Histogram Graph", biMean.histAry);

    biMean.offSet = (biMean.maxVal - biMean.minVal) / 10;

    biMean.dividePt = biMean.offSet;
    int bestThrVal = biMean.biMeanGauss(biMean.dividePt, outFile2);

    biMean.bestFitGauss(bestThrVal);
    biMean.plotGraph(biMean.GaussAry, biMean.GaussGraph, '+');
    biMean.drawGraph(outFile1, biMean.GaussGraph, "Gaussian Curve Graph", biMean.GaussAry);

    biMean.bestThr(outFile1, bestThrVal);

    biMean.addVertical(biMean.histGraph, bestThrVal);

    biMean.drawGraph(outFile1, biMean.histGraph, "Best Threshold Histogram", biMean.histAry);

    biMean.plotAll(outFile1, bestThrVal);

    inFile.close();
    outFile1.close();
    outFile2.close();

    return 0;
}
```

Output1 Part 1: Histogram and bi-Mean Gaussian Curve using Data2

Output1 Part 2: Histogram with best Threshold and
Overlay of Gaussian curve and histogram, with best Threshold using Data2



Output2 Part 2: bi-Mean Gaussian function intermediate results for debugging

Data1

DividePt	LeftSum	RightSum	TotalSum	PrevDiff	BestThr
6	696.943	5234.73	5931.67	5931.67	6
7	781.642	5193.74	5975.38	5931.67	6
8	865.607	5155.22	6020.83	5931.67	6
9	946.713	5131.83	6078.55	5931.67	6
10	1010.86	5121.3	6132.16	5931.67	6
11	1055.12	5133.06	6188.18	5931.67	6
12	1075.9	5144.84	6220.74	5931.67	6
13	1054.09	5119.39	6173.48	5931.67	6
14	1060.08	5039.89	6099.97	5931.67	6
15	1025.16	4881.07	5906.23	5906.23	15
16	970.29	4640.43	5610.72	5610.72	16
17	894.12	4341.38	5235.5	5235.5	17
18	804.034	3948.28	4752.32	4752.32	18
19	726.246	3508.27	4234.52	4234.52	19
20	659.696	3043.94	3703.64	3703.64	20
21	587.214	2514.78	3102	3102	21
22	527.929	1911.38	2439.31	2439.31	22
23	536.504	1523.23	2059.74	2059.74	23
24	554.377	1321.82	1876.2	1876.2	24
25	577.531	1135.22	1712.75	1712.75	25
26	596.343	1031.46	1627.8	1627.8	26
27	618.212	941.053	1559.26	1559.26	27
28	642.45	864.21	1506.66	1506.66	28
29	671.286	791.432	1462.72	1462.72	29
30	701.219	734.12	1435.34	1435.34	30
31	729.867	690.243	1420.11	1420.11	31
32	756.924	657.761	1414.68	1414.68	32
33	781.313	636.008	1417.32	1414.68	32
34	810.592	617.7	1428.29	1414.68	32
35	857.202	594.62	1451.82	1414.68	32
36	921.876	567.983	1489.86	1414.68	32
37	1009.22	540.412	1549.63	1414.68	32
38	1128.67	520.453	1649.13	1414.68	32
39	1346.61	493.288	1839.9	1414.68	32
40	1615.78	471.679	2087.46	1414.68	32
41	1977.97	448.945	2426.91	1414.68	32
42	2360.38	427.105	2787.48	1414.68	32
43	2843.94	396.732	3240.67	1414.68	32
44	3263.71	404.702	3668.42	1414.68	32
45	3647.51	454.881	4102.39	1414.68	32
46	3990.38	510.297	4500.68	1414.68	32
47	4322.48	553.484	4875.96	1414.68	32
48	4630.04	569.755	5199.8	1414.68	32
49	4888.7	595.11	5483.81	1414.68	32
50	5064.27	594.315	5658.58	1414.68	32
51	5172.43	600.53	5772.96	1414.68	32
52	5228.72	590.098	5818.82	1414.68	32
53	5257.02	580.684	5837.71	1414.68	32
54	5259.16	589.885	5849.05	1414.68	32
55	5246.38	585.211	5831.59	1414.68	32
56	5222.82	578.88	5801.7	1414.68	32

Data2

DividePt	LeftSum	RightSum	TotalSum	PrevDiff	BestThr
5	463.731	1637.17	2100.9	2100.9	5
6	609.974	1623.78	2233.76	2100.9	5
7	761.131	1605.62	2366.76	2100.9	5
8	856.009	1598.01	2454.02	2100.9	5
9	1000.24	1586.31	2586.56	2100.9	5
10	1160	1565.62	2725.61	2100.9	5
11	1273.75	1541.97	2815.72	2100.9	5
12	1371.26	1513.73	2884.99	2100.9	5
13	1446.13	1499.85	2945.98	2100.9	5
14	1541.37	1489.12	3030.5	2100.9	5
15	1628.73	1479.89	3108.63	2100.9	5
16	1723.7	1470.51	3194.22	2100.9	5
17	1802.94	1462.97	3265.9	2100.9	5
18	1870.66	1452.27	3322.93	2100.9	5
19	1941.16	1453.73	3394.89	2100.9	5
20	1991.64	1472.46	3464.1	2100.9	5
21	2009.79	1494.93	3504.72	2100.9	5
22	1996.31	1513.6	3509.91	2100.9	5
23	1944.86	1524.18	3469.04	2100.9	5
24	1875.53	1517.97	3393.5	2100.9	5
25	1779.54	1480.81	3260.35	2100.9	5
26	1683.09	1409.58	3092.68	2100.9	5
27	1568.81	1310.98	2879.79	2100.9	5
28	1437.25	1171.03	2608.29	2100.9	5
29	1319.94	1032.76	2352.7	2100.9	5
30	1187.77	883.083	2070.86	2070.86	30
31	1075.59	751.855	1827.44	1827.44	31
32	994.376	659.256	1653.63	1653.63	32
33	948.436	577.519	1525.96	1525.96	33
34	931.431	529.259	1460.69	1460.69	34
35	953.997	535.117	1489.11	1460.69	34
36	1016.72	560.174	1576.89	1460.69	34
37	1082.76	622.044	1704.8	1460.69	34
38	1178.78	680.395	1859.18	1460.69	34
39	1303.85	735.044	2038.89	1460.69	34
40	1445.04	787.095	2232.13	1460.69	34
41	1567.45	819.552	2387	1460.69	34
42	1660.09	869.238	2529.33	1460.69	34
43	1734.7	926.007	2660.71	1460.69	34
44	1782.01	996.283	2778.3	1460.69	34
45	1769.53	1015.19	2784.72	1460.69	34
46	1743.69	1052.91	2796.6	1460.69	34
47	1721.89	1057.23	2779.12	1460.69	34
48	1695.53	1057.43	2752.96	1460.69	34
49	1667.87	1050.9	2718.78	1460.69	34
50	1637.94	1037.08	2675.01	1460.69	34
51	1598.47	1012.6	2611.07	1460.69	34
52	1580.18	954.268	2534.45	1460.69	34
53	1587.04	877.957	2465	1460.69	34
54	1597.5	780.717	2378.22	1460.69	34