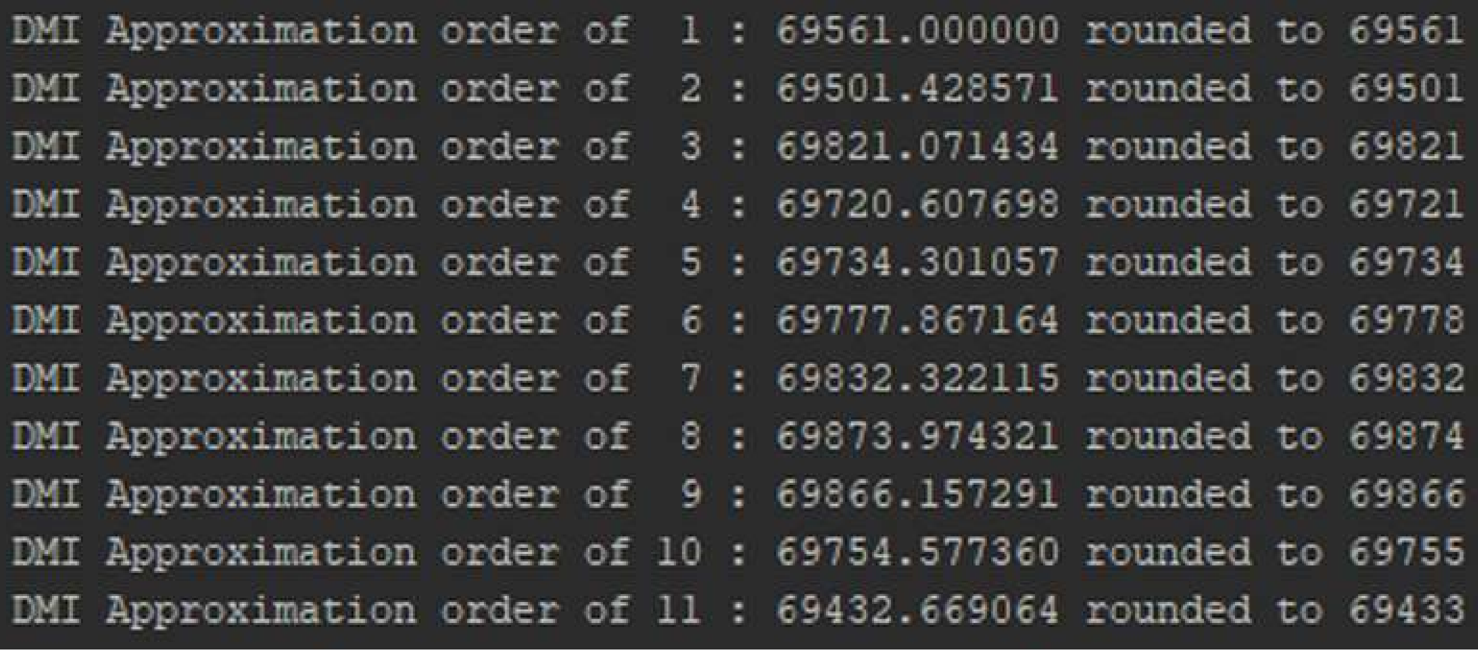
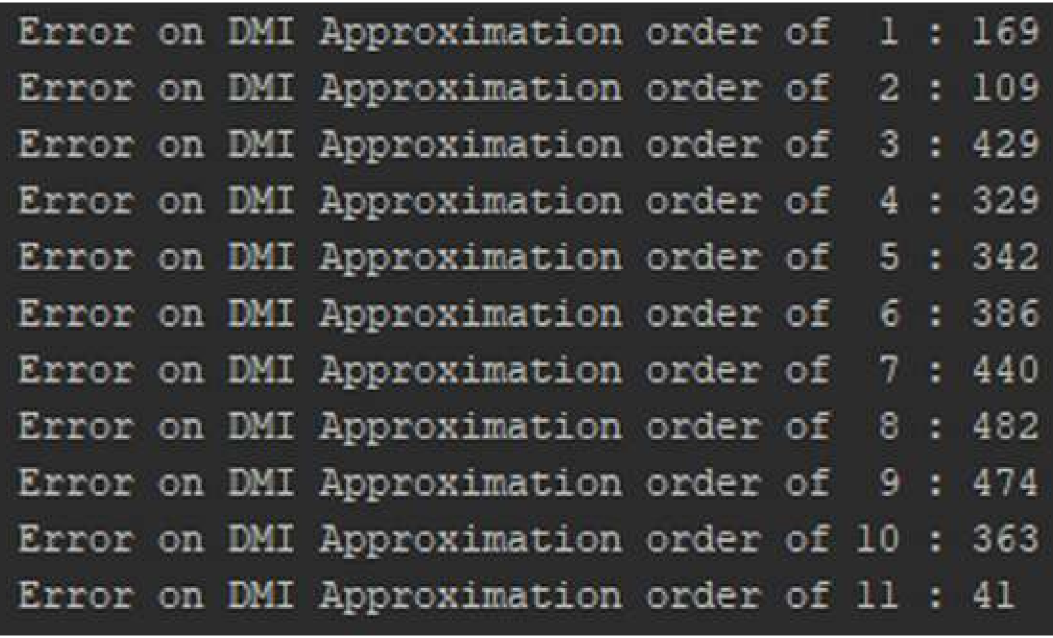
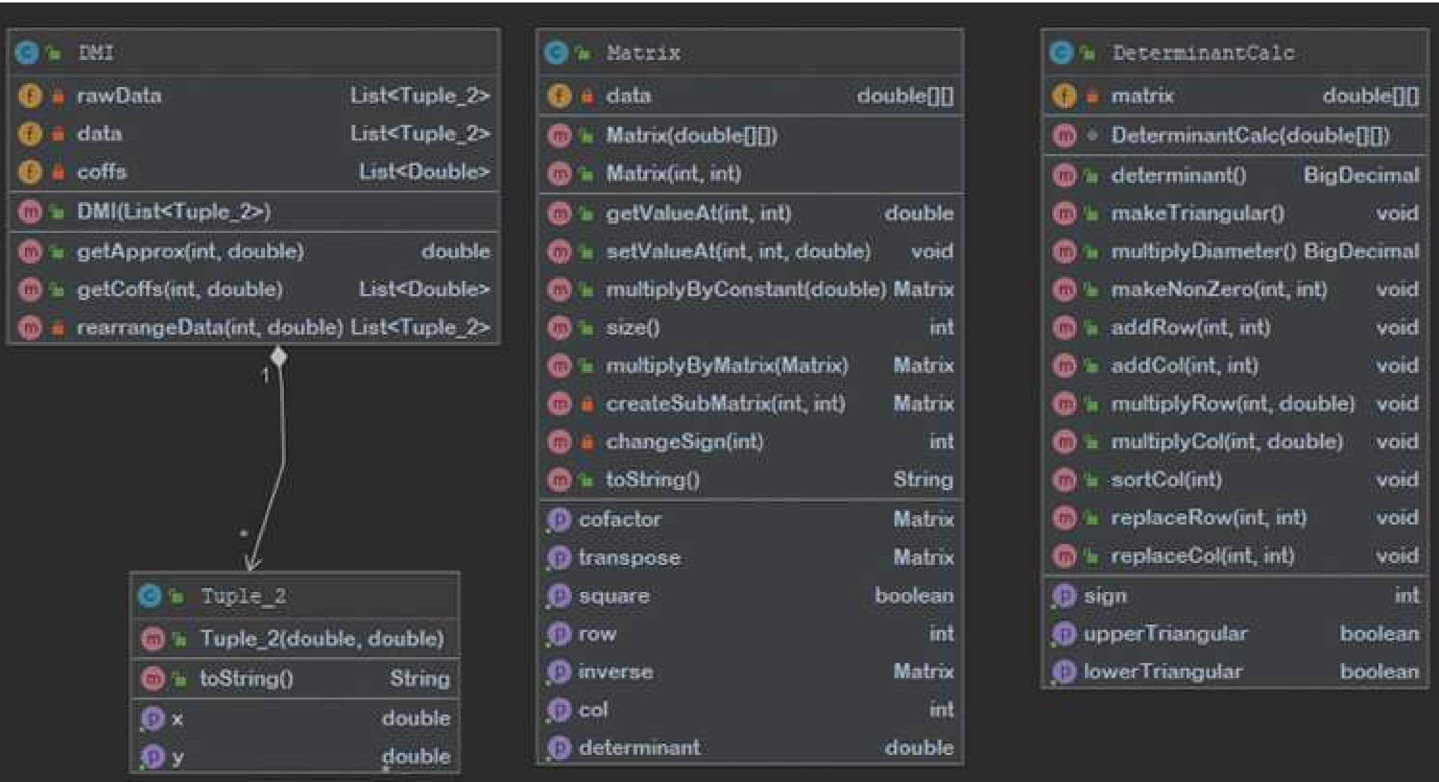
Direct Method Interpolation

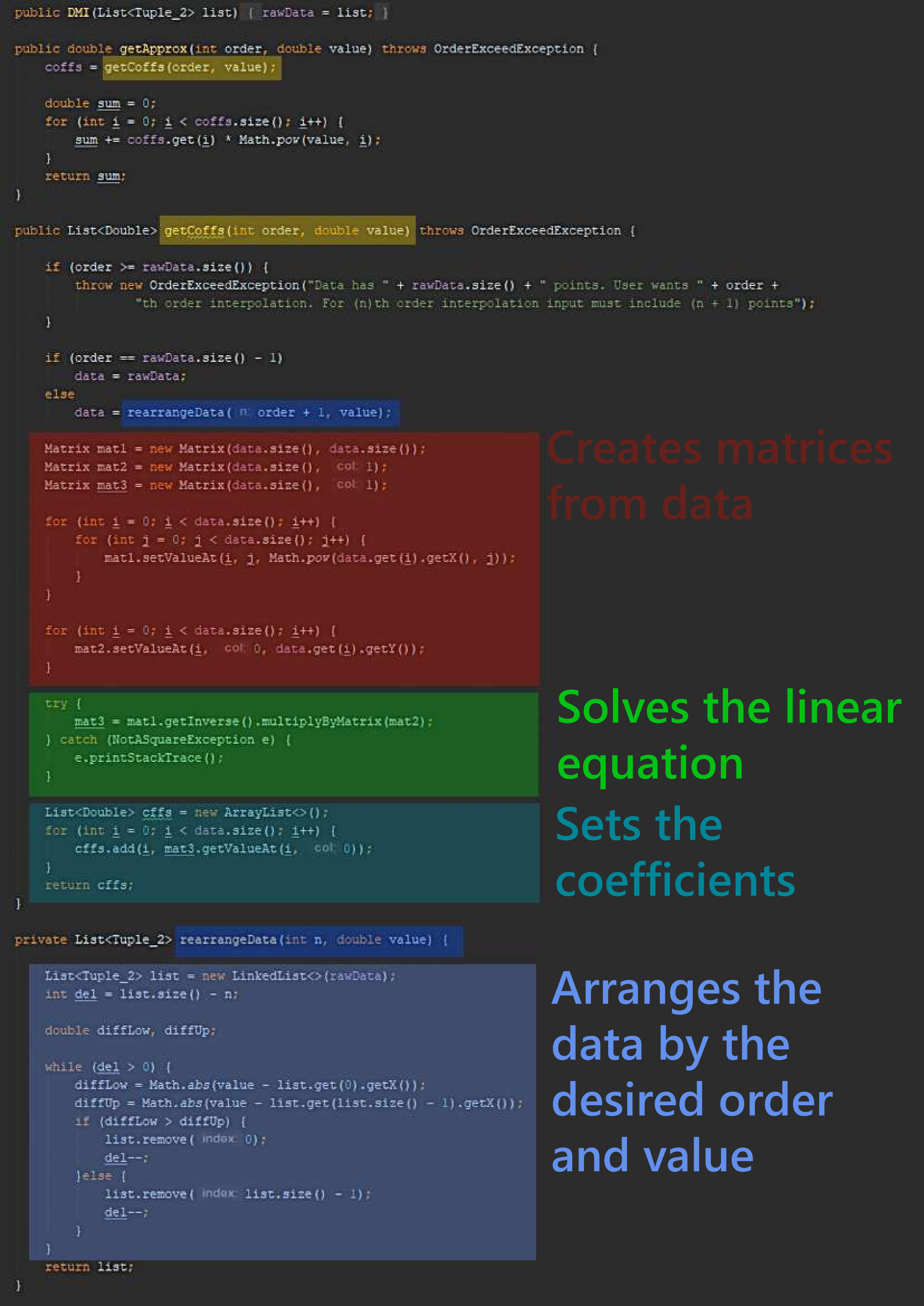


```
DMI Approximation order of  1 : 69561.000000 rounded to 69561
DMI Approximation order of  2 : 69501.428571 rounded to 69501
DMI Approximation order of  3 : 69821.071434 rounded to 69821
DMI Approximation order of  4 : 69720.607698 rounded to 69721
DMI Approximation order of  5 : 69734.301057 rounded to 69734
DMI Approximation order of  6 : 69777.867164 rounded to 69778
DMI Approximation order of  7 : 69832.322115 rounded to 69832
DMI Approximation order of  8 : 69873.974321 rounded to 69874
DMI Approximation order of  9 : 69866.157291 rounded to 69866
DMI Approximation order of 10 : 69754.577360 rounded to 69755
DMI Approximation order of 11 : 69432.669064 rounded to 69433
```



```
Error on DMI Approximation order of  1 : 169
Error on DMI Approximation order of  2 : 109
Error on DMI Approximation order of  3 : 429
Error on DMI Approximation order of  4 : 329
Error on DMI Approximation order of  5 : 342
Error on DMI Approximation order of  6 : 386
Error on DMI Approximation order of  7 : 440
Error on DMI Approximation order of  8 : 482
Error on DMI Approximation order of  9 : 474
Error on DMI Approximation order of 10 : 363
Error on DMI Approximation order of 11 : 41
```





```java
public DMI(List<Tuple_2> list) { rawData = list; }

public double getApprox(int order, double value) throws OrderExceedException {
    coffs = getCoffs(order, value);

    double sum = 0;
    for (int i = 0; i < coffs.size(); i++) {
        sum += coffs.get(i) * Math.pow(value, i);
    }
    return sum;
}

public List<Double> getCoffs(int order, double value) throws OrderExceedException {

    if (order >= rawData.size()) {
        throw new OrderExceedException("Data has " + rawData.size() + " points. User wants " + order +
                "th order interpolation. For (n)th order interpolation input must include (n + 1) points");
    }

    if (order == rawData.size() - 1)
        data = rawData;
    else
        data = rearrangeData( n: order + 1, value);

    Matrix mat1 = new Matrix(data.size(), data.size());
    Matrix mat2 = new Matrix(data.size(),  col: 1);
    Matrix mat3 = new Matrix(data.size(),  col: 1);

    for (int i = 0; i < data.size(); i++) {
        for (int j = 0; j < data.size(); j++) {
            mat1.setValueAt(i, j, Math.pow(data.get(i).getX(), j));
        }
    }

    for (int i = 0; i < data.size(); i++) {
        mat2.setValueAt(i,  col: 0, data.get(i).getY());
    }

    try {
        mat3 = mat1.getInverse().multiplyByMatrix(mat2);
    } catch (NotASquareException e) {
        e.printStackTrace();
    }

    List<Double> cffs = new ArrayList<>();
    for (int i = 0; i < data.size(); i++) {
        cffs.add(i, mat3.getValueAt(i,  col: 0));
    }
    return cffs;
}

private List<Tuple_2> rearrangeData(int n, double value) {

    List<Tuple_2> list = new LinkedList<>(rawData);
    int del = list.size() - n;

    double diffLow, diffUp;

    while (del > 0) {
        diffLow = Math.abs(value - list.get(0).getX());
        diffUp = Math.abs(value - list.get(list.size() - 1).getX());
        if (diffLow > diffUp) {
            list.remove( index: 0);
            del--;
        }else {
            list.remove( index: list.size() - 1);
            del--;
        }
    }
    return list;
}
```

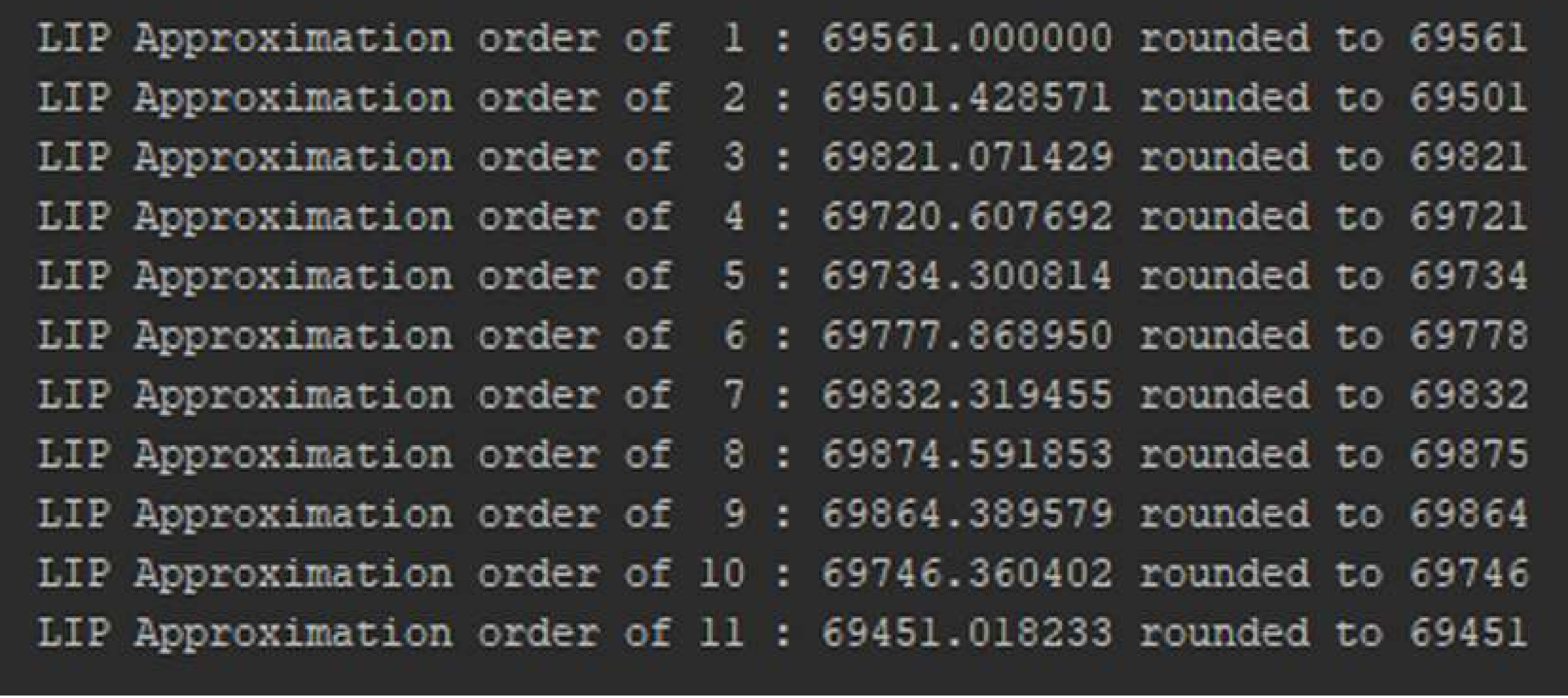
Creates matrices from data

Solves the linear equation

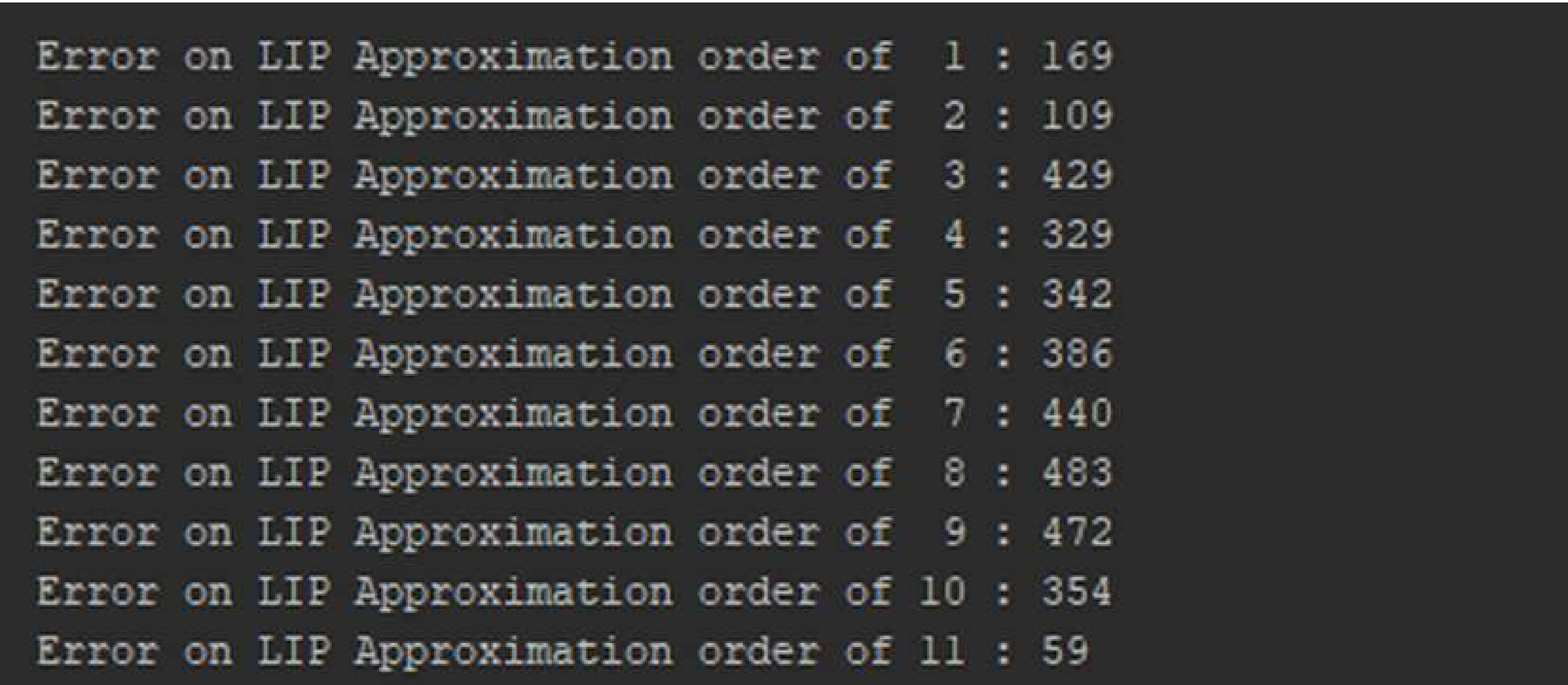
Sets the coefficients

Arranges the data by the desired order and value

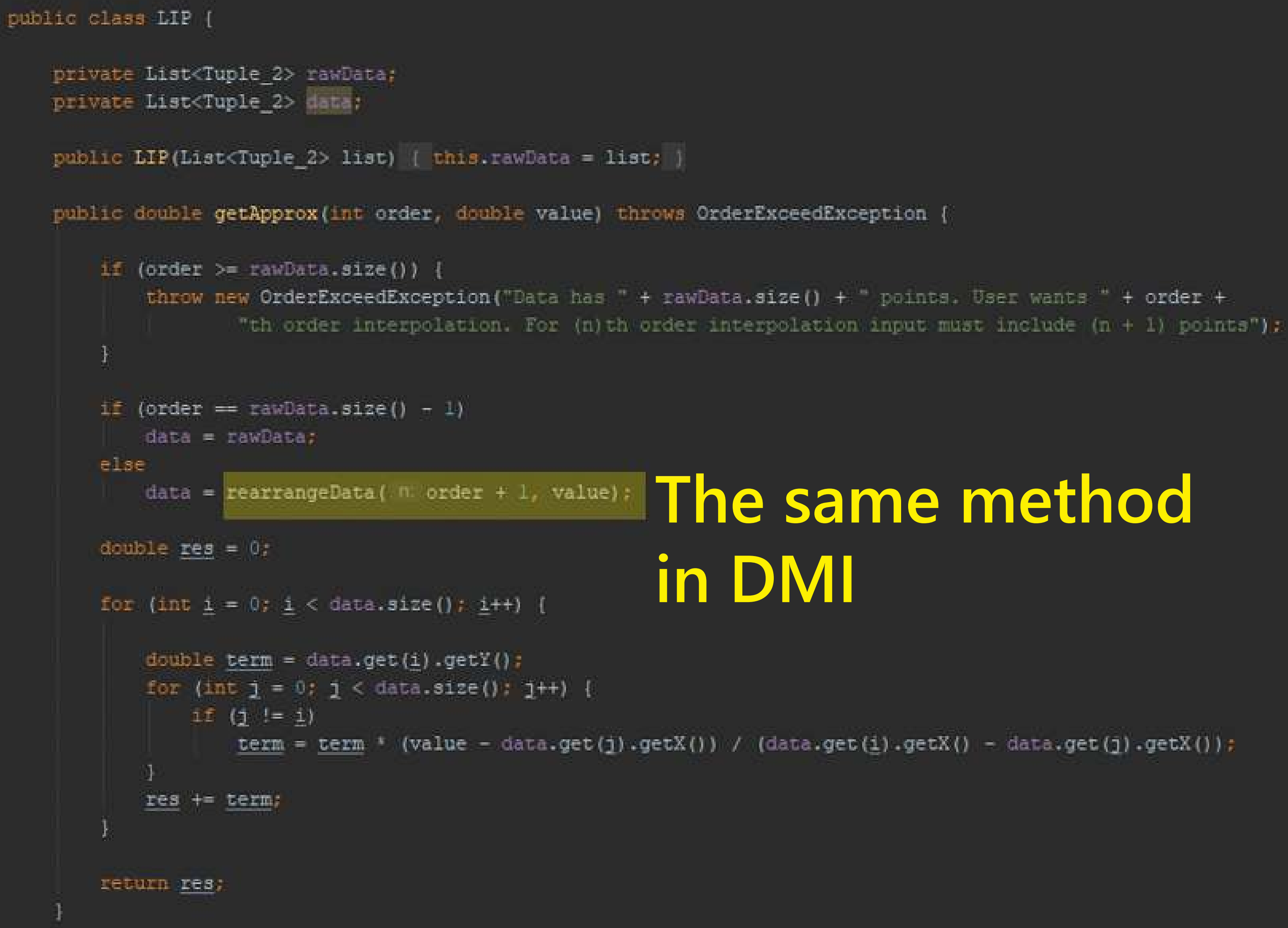
Lagrange Interpolation Polynomial



```
LIP Approximation order of  1 : 69561.000000 rounded to 69561
LIP Approximation order of  2 : 69501.428571 rounded to 69501
LIP Approximation order of  3 : 69821.071429 rounded to 69821
LIP Approximation order of  4 : 69720.607692 rounded to 69721
LIP Approximation order of  5 : 69734.300814 rounded to 69734
LIP Approximation order of  6 : 69777.868950 rounded to 69778
LIP Approximation order of  7 : 69832.319455 rounded to 69832
LIP Approximation order of  8 : 69874.591853 rounded to 69875
LIP Approximation order of  9 : 69864.389579 rounded to 69864
LIP Approximation order of 10 : 69746.360402 rounded to 69746
LIP Approximation order of 11 : 69451.018233 rounded to 69451
```



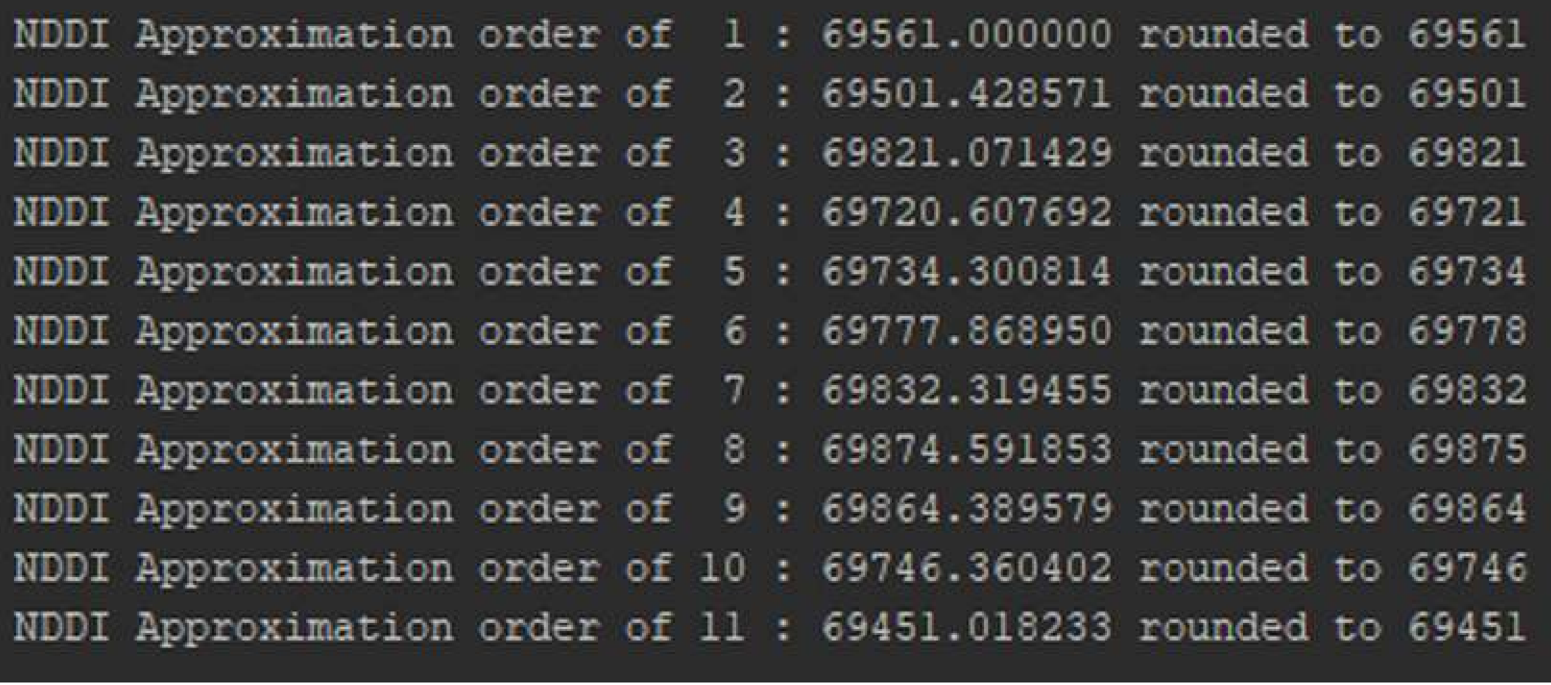
```
Error on LIP Approximation order of  1 : 169
Error on LIP Approximation order of  2 : 109
Error on LIP Approximation order of  3 : 429
Error on LIP Approximation order of  4 : 329
Error on LIP Approximation order of  5 : 342
Error on LIP Approximation order of  6 : 386
Error on LIP Approximation order of  7 : 440
Error on LIP Approximation order of  8 : 483
Error on LIP Approximation order of  9 : 472
Error on LIP Approximation order of 10 : 354
Error on LIP Approximation order of 11 : 59
```



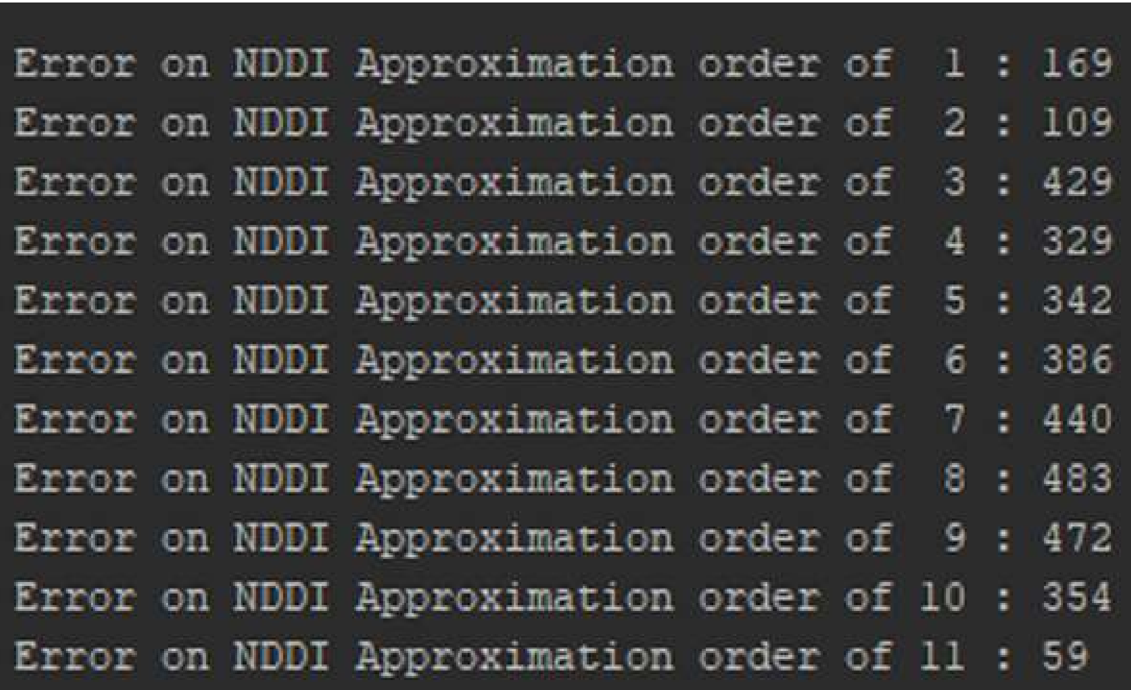
```java
public class LIP {

    private List<Tuple_2> rawData;
    private List<Tuple_2> data;

    public LIP(List<Tuple_2> list) { this.rawData = list; }

    public double getApprox(int order, double value) throws OrderExceedException {

        if (order >= rawData.size()) {
            throw new OrderExceedException("Data has " + rawData.size() + " points. User wants " + order +
                    "th order interpolation. For (n)th order interpolation input must include (n + 1) points");
        }

        if (order == rawData.size() - 1)
            data = rawData;
        else
            data = rearrangeData( n: order + 1, value);

        double res = 0;

        for (int i = 0; i < data.size(); i++) {

            double term = data.get(i).getY();
            for (int j = 0; j < data.size(); j++) {
                if (j != i)
                    term = term * (value - data.get(j).getX()) / (data.get(i).getX() - data.get(j).getX());
            }
            res += term;
        }

        return res;
    }
```

The same method in DMI

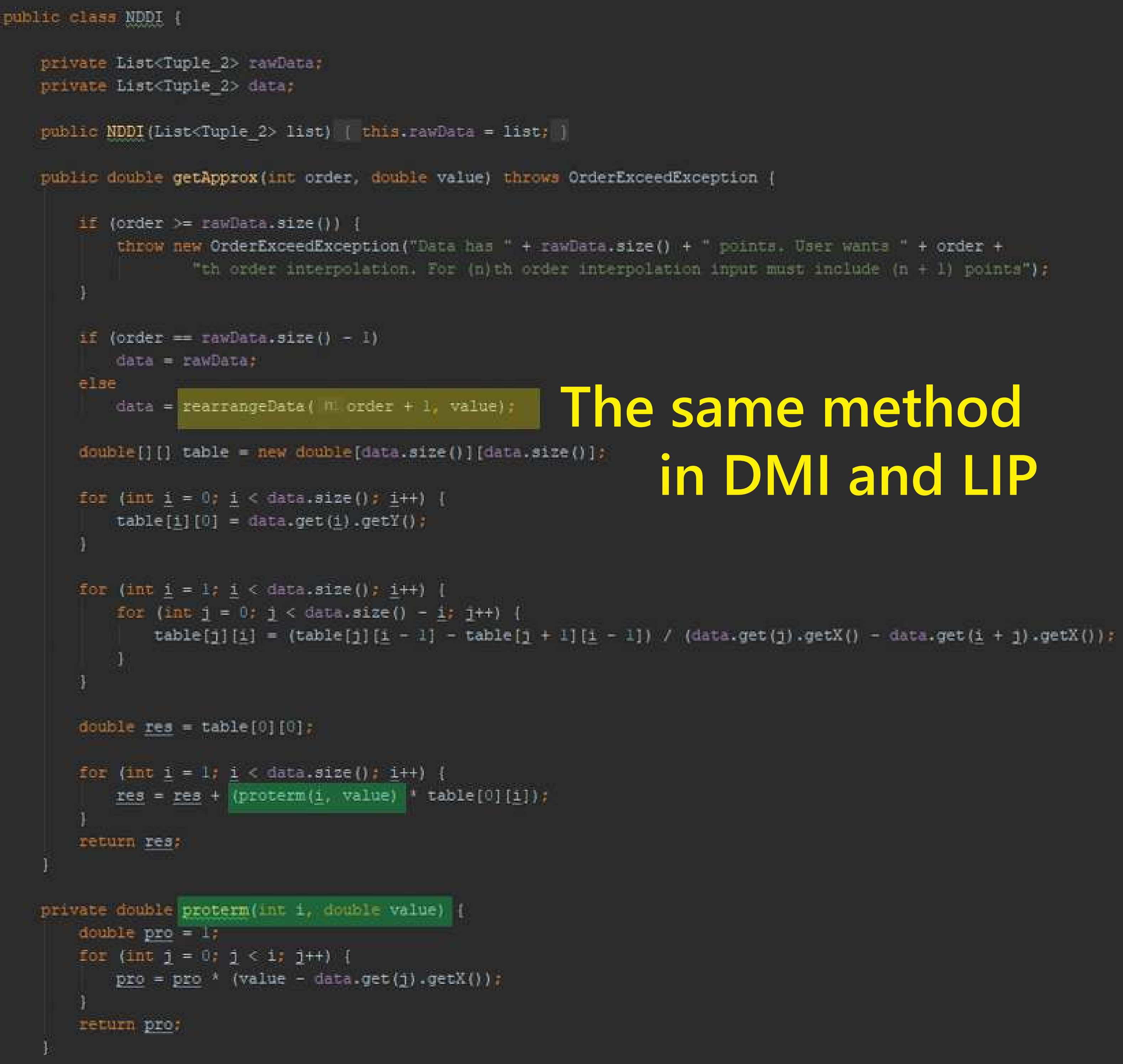
Newton's
Divided Difference Interpolation



```
NDDI Approximation order of  1 : 69561.000000 rounded to 69561
NDDI Approximation order of  2 : 69501.428571 rounded to 69501
NDDI Approximation order of  3 : 69821.071429 rounded to 69821
NDDI Approximation order of  4 : 69720.607692 rounded to 69721
NDDI Approximation order of  5 : 69734.300814 rounded to 69734
NDDI Approximation order of  6 : 69777.868950 rounded to 69778
NDDI Approximation order of  7 : 69832.319455 rounded to 69832
NDDI Approximation order of  8 : 69874.591853 rounded to 69875
NDDI Approximation order of  9 : 69864.389579 rounded to 69864
NDDI Approximation order of 10 : 69746.360402 rounded to 69746
NDDI Approximation order of 11 : 69451.018233 rounded to 69451
```



```
Error on NDDI Approximation order of  1 : 169
Error on NDDI Approximation order of  2 : 109
Error on NDDI Approximation order of  3 : 429
Error on NDDI Approximation order of  4 : 329
Error on NDDI Approximation order of  5 : 342
Error on NDDI Approximation order of  6 : 386
Error on NDDI Approximation order of  7 : 440
Error on NDDI Approximation order of  8 : 483
Error on NDDI Approximation order of  9 : 472
Error on NDDI Approximation order of 10 : 354
Error on NDDI Approximation order of 11 : 59
```



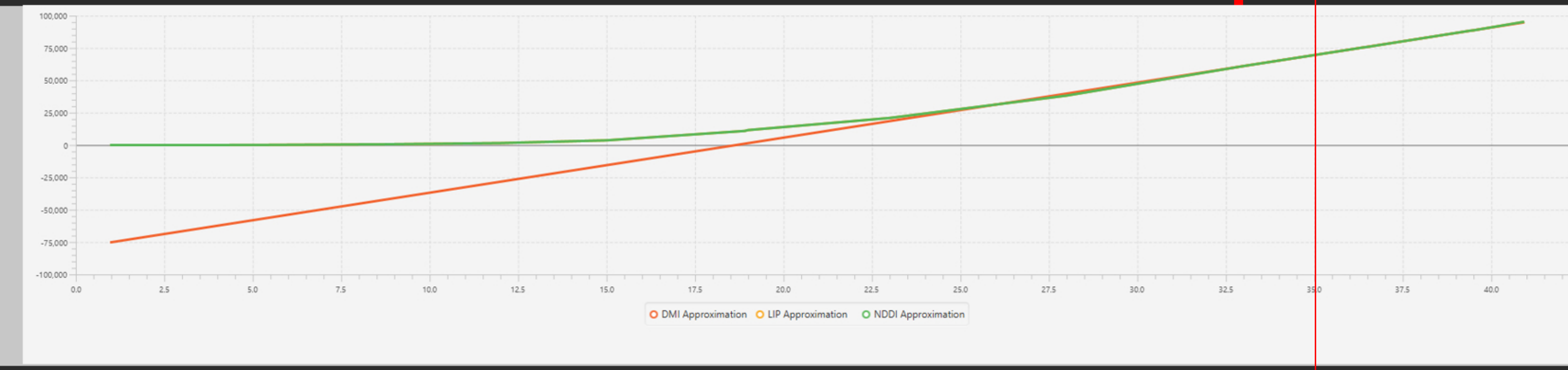
```java
public class NDDI {

    private List<Tuple_2> rawData;
    private List<Tuple_2> data;

    public NDDI(List<Tuple_2> list) { this.rawData = list; }

    public double getApprox(int order, double value) throws OrderExceedException {

        if (order >= rawData.size()) {
            throw new OrderExceedException("Data has " + rawData.size() + " points. User wants " + order +
                    "th order interpolation. For (n)th order interpolation input must include (n + 1) points");
        }

        if (order == rawData.size() - 1)
            data = rawData;
        else
            data = rearrangeData( m: order + 1, value);

        double[][] table = new double[data.size()][data.size()];

        for (int i = 0; i < data.size(); i++) {
            table[i][0] = data.get(i).getY();
        }

        for (int i = 1; i < data.size(); i++) {
            for (int j = 0; j < data.size() - i; j++) {
                table[j][i] = (table[j][i - 1] - table[j + 1][i - 1]) / (data.get(j).getX() - data.get(i + j).getX());
            }
        }

        double res = table[0][0];

        for (int i = 1; i < data.size(); i++) {
            res = res + (proterm(i, value)) * table[0][i]);
        }
        return res;
    }

    private double proterm(int i, double value) {
        double pro = 1;
        for (int j = 0; j < i; j++) {
            pro = pro * (value - data.get(j).getX());
        }
        return pro;
    }
```

The same method in DMI and LIP

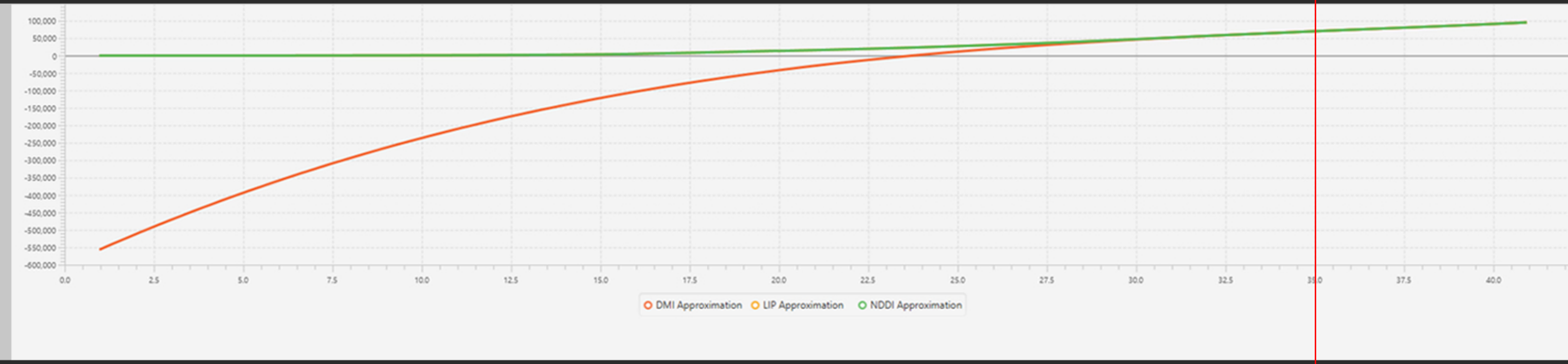
Plot

1st Order

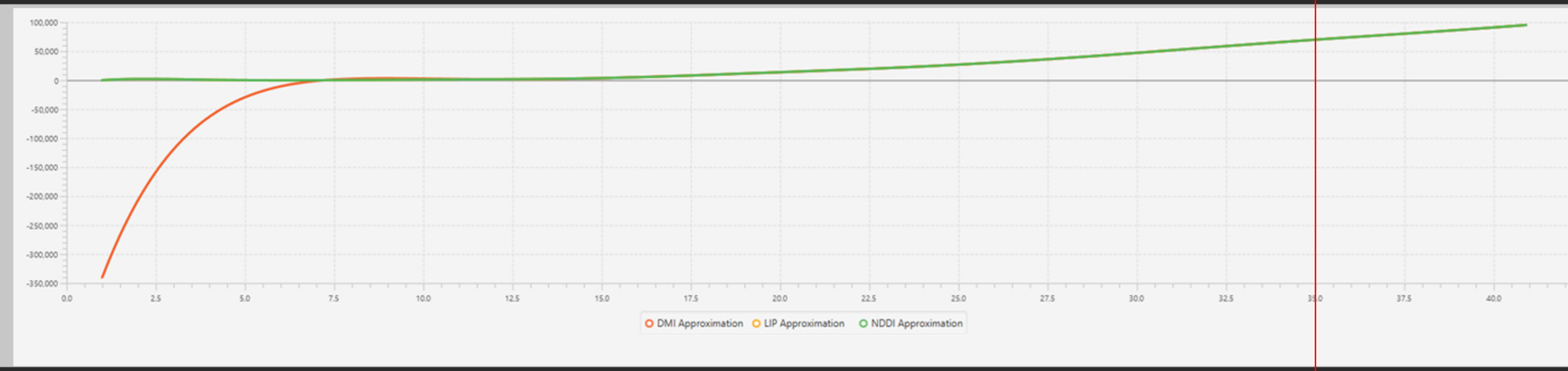
April 15



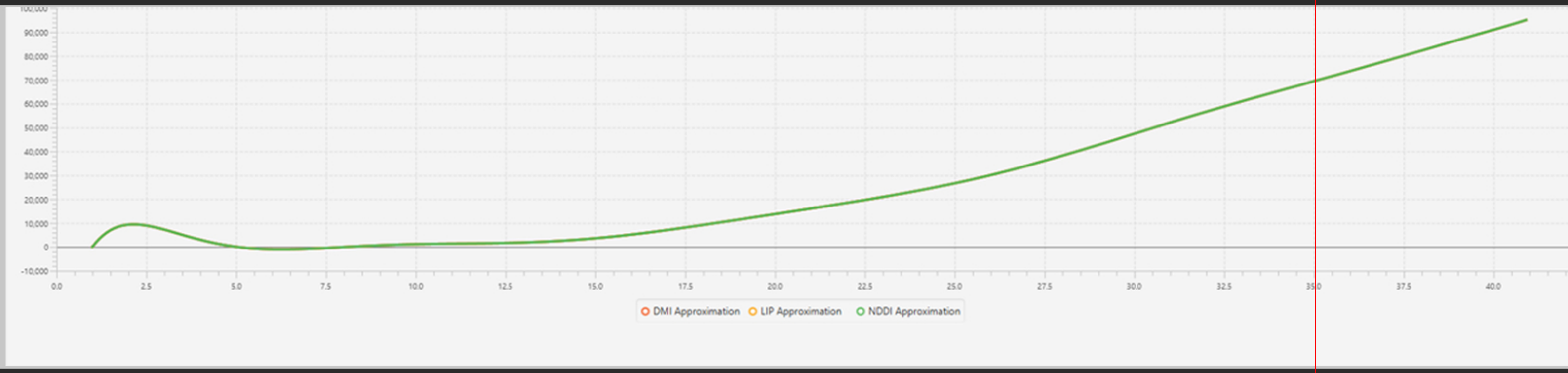
3rd Order



8th Order



11th Order


DMI Approximation
LIP Approximation
NDDI Approximation