情報科学演習１_第１期課題
グラフの描画

6321120
横溝　尚也

提出日：１０月１１日（火）

ソースコード名
棒グラフ:LineGraph.java
円グラフ:CircleGraph.java
レーダーチャート:RadarChart.java

1　棒グラフ

1.1　アルゴリズムの説明

棒グラフの描画は基本的にすべて直線で構成されているため座標を記述することで描画できる。

1.1.1　縮尺について

プログラムをする際に window の表示設定を記述する場面がある。そこで window の表示サイズ変えたときにグラフが途切れていたり、中央に出力されず右端に寄っているときれいとは言えない。そこで自分は window のサイズを４００×４００を基準としてすべてのプログラムを表した。そして、window のサイズを記述した値（例えば２００×２００）が４００×４００に対する縮尺を考え GraphicContext クラスの scale メソッドを使用して縮尺を指定した。
このアルゴリズムは棒グラフ、円グラフ、レーダーチャートすべてに使用している。

1.1.2　軸の値の設定

横軸については、初めに読み込んだデータ数を数えることでその分のメモリを描画し、そのデータが何番目なのかを出力すればよい。
縦軸の数値の設定が複雑になる。自分はデータの中の最大値を抽出し、その値が５０区切りとしたどの範囲（例えば、data.txt の最大値２１０ならば、２００～２５０）であるのかを自動で計算するプログラムを作り、縦軸の値の最大値を決めた（最大値２１０ならば２５０とした)。

1.1.3　グラフの描画について

棒グラフを描画するには、データとデータをすべて直線で結んでいく。その際、x 座標は横軸の長さをデータの要素数で割った値ごとに等間隔にした。y 座標はデータの値によってそれぞれ変化させた。

1.2　プログラムの説明

プログラムをわかりやすく説明するために、ソースコードとは別にプログラムを張り付け、それに行数を付随させたものを記載する。

```
1import javafx.application.Application;
2import javafx.scene.Scene;
3import javafx.stage.Stage;
4import javafx.scene.Group;
5import javafx.scene.canvas.Canvas;
6import javafx.scene.paint.Color;
7import javafx.scene.canvas.GraphicsContext;
8import java.io.*;
```

```
9public class LineGraph extends Application {

10    double wincoordinatex = 200;
11double wincoordinatey = 200;



12    @Override
13    public void start(Stage st) throws Exception {
14 Group root = new Group();
15 Canvas canvas = new Canvas(wincoordinatex, wincoordinatey); //window への描画の設定
16 GraphicsContext gc = canvas.getGraphicsContext2D();
17 drawShapes(gc);

18 root.getChildren().add(canvas);

19 Scene scene = new Scene(root, wincoordinatex, wincoordinatey, Color.WHITE);
20 st.setTitle("Line Graph");
21 st.setScene(scene);
22 st.show();
23    }

24    public static void main(String[] a) {
25 launch(a);
26    }

27    private void drawShapes(GraphicsContext gc) {
28 gc.scale(wincoordinatex / 400, wincoordinatey / 400);

29 int[] data = new int[1000]; //配列 pa に読み込んだ値を格納し、データ数を numOfpa 煮格納する
30 int numOfdata = 0;
31 try {
32     File file = new File("./data.txt");
33     FileReader fr = new FileReader(file);
34     BufferedReader br = new BufferedReader(fr);
35     String str = br.readLine();
36     String strdata = str;

37     while (str != null) {   //読み取った文字を全て結合する
38 str = br.readLine();
```

```
39 if (str != null){
40     strdata = strdata + " " + str;}
41     }

42     br.close();
43     String[] arraydata = strdata.split(" ");
44     for(int i = 0; i < arraydata.length; i++){
45 data[i] = Integer.parseInt(arraydata[i]);
46     }
47     numOfdata = arraydata.length;


48 } catch (FileNotFoundException e) {    //例外が生じた時の処理
49     System.err.println("ファイルが開けませんでした．");
50 } catch (IOException e) {
51     System.err.println("ファイルの読み込みに失敗しました．");
52 } catch (NumberFormatException e) {
53     System.err.println("整数化に失敗しました。");
54 }
//ここまでは３つのグラフ全てに記述するプログラム


55 int maxdata = data[0]; //データの最大値を求める
56 for(int i = 1; i < numOfdata; i++){
57     if(maxdata >= data[i]){
58     }else{ maxdata = data[i];
59     }
60 }
61 int staOfvertical = 0;          //グラフの縦軸をデータの最大値によって設定
62 for(int i = 0; i<1000; i++){
63     if((maxdata >= i*50) && (maxdata < i*50 + 50)){
64 staOfvertical = i*50 + 50;
65 break;
66     }
67 }

68 gc.setFill(Color.DARKGRAY);
69 gc.fillRect(50, 50, 300, 200); //背景の描画
70 gc.setStroke(Color.BLACK);
71 gc.setFill(Color.BLACK);
```

```
72 gc.strokeLine(50, 250, 350, 250); // 横軸
73 gc.strokeLine(50, 250, 50, 50);   // 縦軸

74 for (int i = 0; i < (staOfvertical/50 +1); i++) {
75     gc.strokeLine(48, 50 + i * 200 / (staOfvertical/50),
76   350, 50 + i * 200 / (staOfvertical/50));// 縦軸の目盛
77     gc.fillText(String.valueOf(i * 50), 20,
78 250 - i * 200 / (staOfvertical/50));      // 縦軸の値
79 }

80 for (int i = 0; i < numOfdata + 1; i++) {
81     gc.strokeLine(50+(i*300/numOfdata), 250,
82   50+(i*300/numOfdata), 252);     // 横軸の目盛
83 }

84 for (int i = 0; i < numOfdata; i++) {
85     gc.setFill(Color.BLACK);
86     gc.fillText(String.valueOf(i + 1),
87 57 + (i * 300 / numOfdata), 263); // 横軸の値
88 }

89 gc.setStroke(Color.BLUE); //棒の塗りつぶし
90 for(int i = 0; i < numOfdata - 1; i++){
91     gc.strokeLine(60 + (i * 300 / numOfdata), 250 - data[i]*200/staOfvertical
92   ,60+((i+1)*300/numOfdata), 250-data[i+1]*200/staOfvertical);
93 }
94    }
95}
```

１～８行目：既存のクラスの機能（例えば GraphicsContext）を使用してプログラムを使用するためのインポートしている。
９行目以降：LineGraph クラスの記述である。
１０～２３行目：window の描画の設定を主にしている。
２８行目：縮尺を計算し、すべての座標に縮尺を反映させている。
２９～４３行目：、読み込んだデータが数行であるときはそれぞれの行のデータを文字列として結合している。データを読み込み配列に格納。
４４～４７行目：データ数を調べている。
４８～５４行目：ファイルを読み込むまでのプログラムで例外が起きたときの処理。
５５行目以降：棒グラフの描画についてのプログラム。

５５～６７行目：データの最大値を求め、縦軸の値の最大値をいくつにするかくり返し文を用いて調べている。
６８～８８行目：横軸や縦軸背景の色など棒以外の描画
８９～９５行目：読み込んだデータを使用して棒を描画、内部を塗りつぶしている。

1.3　プログラムの実行

図１は data.txt を読み込んで実行したときの結果である。
図１の場合、データの最大値は 210 であるため、縦軸の最大値は 250 になるようプログラムしてある。
次にデータの最大値を 298 とし、データ数をさらに増やした 20 に変更して実行した物が図２である。
次に window のサイズを２００×２００としたとき（縮尺 $\frac{1}{2}$）の出力結果が図３である。円グラフ、レーダーチャートともに window の大きさへの対応は同様であるため、以後出力結果は省略する。

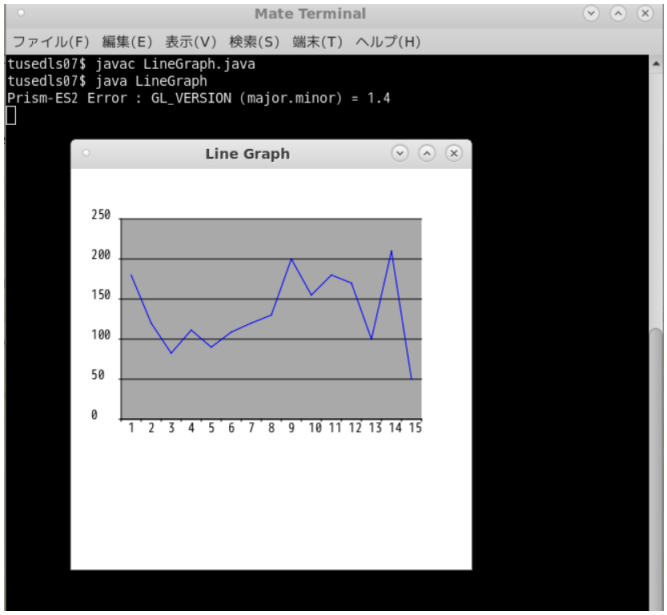

図１　data.txt の実行結果

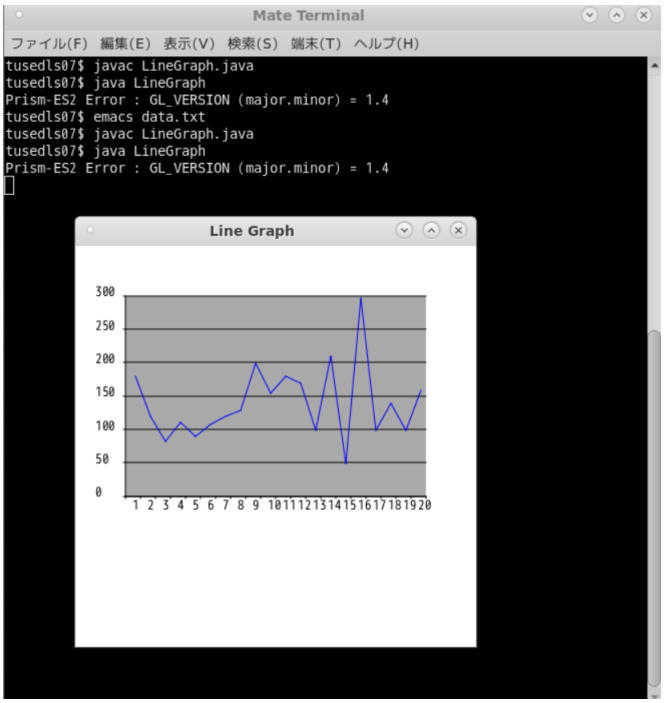

図 2　データ最大値 298、データ数 20 の実行結果

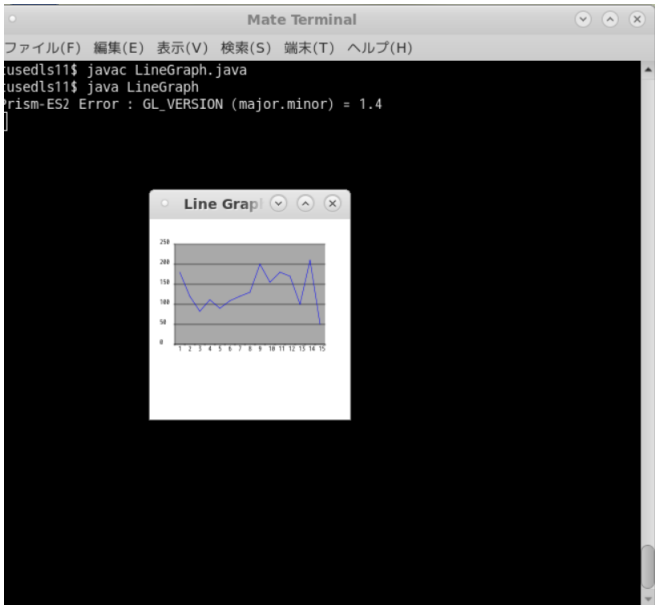

図 3　window の縮尺を変えた実行結果

1.4 考察

棒グラフを書く当たって基本的なアルゴリズムはすべて直線で構成されていることもあり、そこまで難しくはなく工夫点はあまりない。
GraphicContext クラスにある直線を引く、文字を出力する、色を塗りつぶす機能を使用すると棒グラフが書けることが分かった。

縦軸の設定はデータの最大値によって変えているため、グラフを飛び出して直線が描画されることはあり得ない。これが今回採用したアルゴリズムの利点である。しかし、大きすぎる値や外れ値が現れたときに、出力がきれいではなくなるのが欠点である。プログラムの欠点は詳しく後述する。

1.4.1 このプログラムの欠点

プログラムを実行したり、このレポートを書きながら自分のプログラムの改良点が見つかった。

一つ目は外れ値への対処である。このプログラムではデータの最大値によって縦軸の値を設定している。そのためデータの最大値が外れ値であった場合（データの最大値２００かつ、他のデータがすべて１０程度であったとき)、ほとんどの直線が縦軸の最下底で描画されてしまい、理想のグラフとは言えない。今回改善することはできなかったが、改善方法を考察する。
まず初めに外れ値が存在するとき、プログラムが自動で検出する必要がある。ここでいくつか外れ値検出方法について調べたので大まかに記載する（参考文献使用)。まず初めに各データの距離をすべて計算する。そしてある点からの距離を大きい順に並べ、距離が大きかったものを外れ値として検出する方法がある。これを k-近傍法という。他にも異常値を $a(x') = (\frac{x'-\mu}{\sigma})^2$（x': 各データ μ: 平均値,σ: 分散）として計算して検出するホテリング理論や、スミルノフ=グラブス検定などもある。
これらの方法で外れ値を検出できたとしたら、その外れ値以外で最大値を抽出し、縦軸の値を決めたり、横に波線を引き、縦軸の値を外れ値の部分だけ飛ばせばよい。

二つ目はデータがすべて大きかった時の描画がきれいでは荷ことである。一つ目と同様に縦軸の値設定に関するもので、データがすべて大きな値の時（１００００程度）に問題が生じる。自分は５０間隔の範囲のどこにデータの最大値が存在しているか調べ、５０ごとに縦軸の値を出力している。そのため縦軸の値が出すぎてしまう。この時の実行結果が以下のようになってしまう。

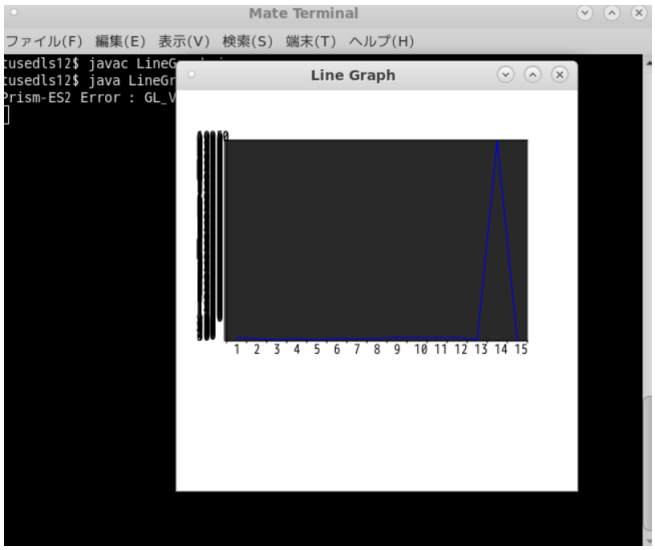

図 4　データの値が大きいときの欠点

2　円グラフ

2.1　アルゴリズムの説明

縮尺についてはアルゴリズム、プログラム、考察共に同じである。

2.1.1　円グラフの描画について

自分は window ４００×４００を基準として、考えた。円の中心の座標と半径をはじめに決めて円グラフそれぞれの色で塗りつぶすときに中心座標と半径を使用した。GraphicContext クラス内に扇形を塗りつぶすメソッドがあるのでそれを使用する。
出力全体を見ると図形は円になっているが各データがあらわす図形は扇形である。そのため、各データがあらわす扇形を描画することで扇形の角の大きさの合計が３６０度となるように描画していけばよい。

2.1.2　扇形を塗りつぶすパラメータについて

扇形を塗りつぶす際には、パラメータとして
(中心 x 座標、中心 y 座標、円弧の幅、円弧の高さ、円弧の始角 (度単位)、円弧の角の大きさ (度単位)
) が必要である。
中心座標は初めに決めた円の中心座標と同じ、円弧の幅と高さは直径に等しい。

2.1.3　扇形の始角と角の大きさについて

k 番目の始角を求める時、k-1 番目までの塗りつぶした角の大きさの合計を求めておく必要がある。よってあるパラメータ s（プログラム２０行目）を用意しておき、一つの扇方が塗りつぶされていくごとにその角の大きさを足し合わせておく。そうすることで k 番目の始角 s と簡単に表すことができる。
次に角の大きさを求める前に、全データの合計 t（プログラム２１行目）を用意しておく。すると、

$$(\text{角の大きさ}) = 360 \times (\text{各データの値}) \div (\text{全データの合計値})$$

と表せる。

2.1.4　各データ値の円の外への出力方法

円弧の外側にそれぞれデータの値を出力したいがこの時パラメータ x,y 座標が必要である。棒グラフとは違い、円の弧の外にデータの値を出力したいときにその座標を求めることが困難である。そこで自分は極座標を用いて座標を表し、それを直交座標へと変換した。以下がその式である。

データの値を出力する座標
`(x,y)=(s + (r + 10)cos θ , t + (r + 10)sin θ )`
(∵ s,t 円の中心座標,r: 円の半径，θ :(扇形の始角)+(扇形の角の大きさ) ÷２)

2.2　プログラムの説明

window の設定や、ファイルからの読み取り、例外処理は棒グラフと同様であるため省略する。

```
1 gc.strokeOval(110,110,180,180); //円の描画
2 gc.setLineWidth(2);

3 gc.setStroke(Color.BLACK); //右に記載するデータ番号と色についての大枠の描画
4 gc.strokeLine(340,50,380,50);
5 gc.strokeLine(340,50 + (numOfdata * 20) + 6 ,380, 50 + (numOfdata * 20) + 6);
6 gc.strokeLine(340,50,340,50 + (numOfdata * 20) + 6);
7 gc.strokeLine(380,50,380,50 + (numOfdata * 20) + 6);
8 gc.setFill(Color.LIGHTGRAY);
9 gc.fillRect(340,50,40,numOfdata * 20 + 6);


10 double total = 0;//各データの全体に占める割合を算出するために、データの合計値を計算
11 for(int i = 0;i < numOfdata;i++){
12     total = total + data[i];
13 }

14 Color[] color = new Color[numOfdata]; //乱数を使用して色をランダムに指定
15 Random rnd = new Random();
16 for(int i = 0; i < numOfdata; i++){
17     color[i] = Color.rgb(rnd.nextInt(256),rnd.nextInt(256)
18  ,rnd.nextInt(256));
19 }


20 double s = 90;
21 double t = -(Math.PI / 2);
22 for(int i = 0; i < numOfdata; i++){  //円ぐらふの色の描画
23     gc.setFill(color[i]);
24     gc.fillArc(110,110,180,180,s,-(data[i] * 360 / total ),ArcType.ROUND);
25     s = s - (data[i] * 360 / total);
26     gc.setFill(Color.BLACK);

27     gc.fillText(String.valueOf(data[i]), //数値の出力
28 190 + 110 * Math.cos(t + (Math.PI * data[i] / total / 2)
29      + (Math.PI / 36) ),
30 203 + 110 * Math.sin(t + (Math.PI * data[i] / total /2)
31      + (Math.PI /36)) );
32     t = t + (2 * Math.PI * data[i] / total);
```

```
    //各データの詳細を記載するときの番号と色の描画
33      gc.fillText(String.valueOf(i+1), 360 , 67 + (i * 20));
34      gc.setStroke(Color.BLACK);
35      gc.setLineWidth(1);
36       gc.strokeLine(347, 60 + (i * 20), 355, 60 + (i * 20));
37      gc.strokeLine(347, 68 + (i * 20), 355, 68 + (i * 20));
38      gc.strokeLine(347, 60 + (i * 20), 347, 68 + (i * 20));
39      gc.strokeLine(355, 60 + (i * 20), 355, 68 + (i * 20));

40      gc.setFill(color[i]);
41      gc.fillRect(347, 60 + (i * 20), 8, 8);
42  }
43  }
44}
```

１～９行目：円と右側に描画する長方形、の描画
１０～１９行目：後に使用するデータの合計値に対する各データの割合の計算と、配列にランスによって指定した色の格納
２０～３２行目：扇形とデータの値を描画。この時原点から鉛直上向きの直線から円グラフが始めらないといけないので度数法では９０，ラジアン表記では-$\pi \div 2$ をすべての角に反映させている。
３３～４４行目右側に描画する正方形と色の描画

2.3 プログラムの実行

図５は data.txt を読み込んだ時の実行結果である。図６はデータ数を７個に少なくした時の実行結果である。

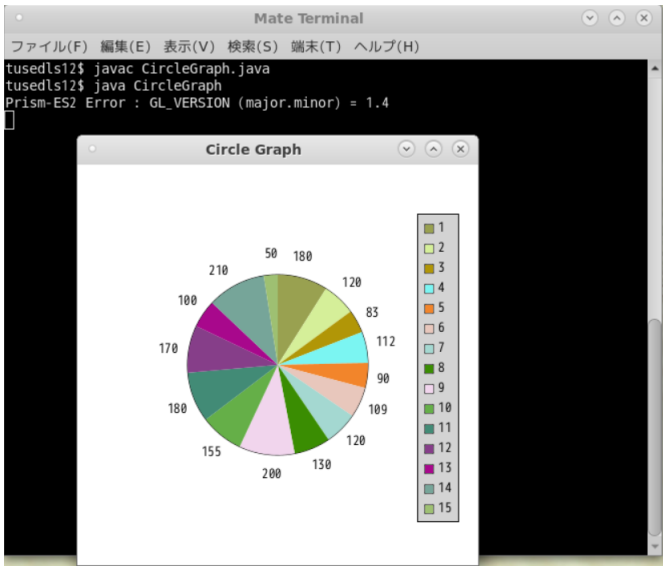

図 5　data.txt の実行結果

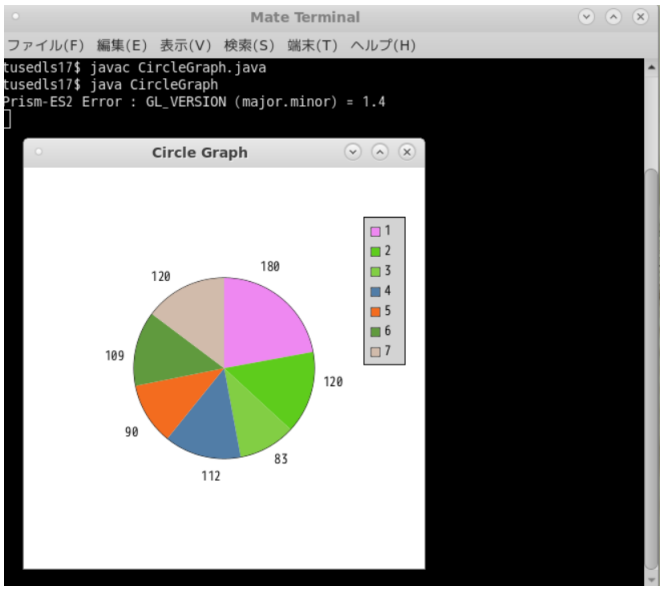

図 6　データ数 7 個の実行結果

2.4　考察

2.4.1　各データの色の割り当てについて

色の割り当てはランダムにしたいため、乱数を使用して色を指定した。乱数を使用して出力された色を配列に格納することで後に描画するときに使用する。また、配列の要素数はデータ数にすればよいこともわかる。

2.4.2　テキストの出力について

円の外部にデータの値を出力させたいときに、正確に座標計算できていても数字位置が少しズレた実行結果となった。これは、数字が二けた以上であるときに左端の数字に座標が適用されるため、桁数が増えるごとに右にズレていってしまっているとかが得られる。その為、円グラフにきれいに出力するために、入力する x 座標をすこし小さくすることで全体的にきれいな数値配置を実現させた。

2.4.3　データ数への対応について

出力したときの右の長方形（データの色と番号を表す表）は点と点を結び、その中を灰色で塗りつぶすことで描画している。この線を引くときの座標を定数にしてしまうと、データ数が変化したときに不格好になってしまう。そこで自分はデータ数に応じて長方形の縦の長さを指定して、要素が少なくなった時でもきれいに描画されるようにした。それが実行結果の図７である。

2.4.4　三角関数のパラメータについて

扇形を塗りつぶす際のパラメータである角の大きさは度数法と指定されていた。しかし、極座標を使用して座標を表したときは三角関数の使用が必須であった。この時三角関数の変数 θ はラジアンで入力しなければならないこともわかった。

2.4.5　プログラムの改善点

このプログラムでは、各データの割り当てられた番号と色を右に出力する際に、データ数１８個よりも少ないものには背景の灰色の長方形が要素数に合わせて縮小するようになっている（実行結果の図６参照)。これは一つのデータについて y 座標 20 ずつ幅をとり、その長さ分の長方形を出力している。つまり、データ数が大きくなると window の大きさよりも長方形やデータ詳細を配置する y 座標が大きくなりはみ出てしまう。
データ数を増やしたときの実行結果が図７を下に示す。

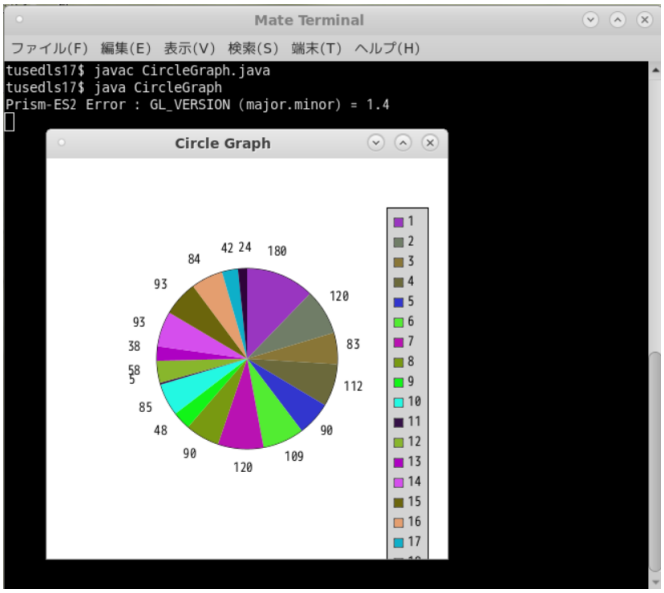

図 7　データ数を増やしたときの欠点

3　レーダーチャート

レーダーチャートは基本的に棒グラフと円グラフのアルゴリズムやプログラムを使用することで実現可能であるため、レポートは少なめに記入する。
縮尺、縦軸の値設定についてはアルゴリズム、プログラム、考察共に同じである。

3.1　アルゴリズムの説明

レーダーチャートを描画する際に新しく必要な考え方は、多角形内を塗りつぶす方法である。これはすべての点の x,y 座標座標を格納した配列がパラメータとして設定されているメソッドを使用する。そのため、読み込んだデータを各点としてプロットし、その座標を格納した配列を用意する必要がある。

3.1.1　データの座標の表し方

座標を表すには円グラフと同様に極座標を使用する。極座標で表すために必要なパラメータは半径 r と角度 θ である。

θ について考える。

（k 番目のデータに使用する角度 θ）=2 × π ÷（データ数）× k
と求めることができる。

次に半径 r について考える。r は棒グラフと同様にデータの値を反映させなければいけなく、今回自分は中心からデータの値を表す軸の長さを１３０とした。

(k 番目の半径 r)=130 ×（データの値）÷（軸の最大値）
と求めることができる。

3.1.2 背景の網目の作成方法

くり返し文の中にくり返し文を入れることで中心から各点方向の軸と、値がいくつなのか見やすくするための軸（棒グラフでいう縦軸）を同時に描画する。

3.2 プログラムの説明

```
1 int maxdata = data[0]; //データの最大値を求める
2 for(int i = 1; i < numOfdata; i++){
3     if(maxdata >= data[i]){
4     }else{
5 maxdata = data[i];
6     }
7 }

8 int staOfvertical = 0;          //グラフの縦軸をデータの最大値によって設
9 for(int i = 0; i<1000; i++){
10     if((maxdata >= i*50) && (maxdata < i*50 + 50)){
11 staOfvertical = i*50 + 50;
12 break;
13  }
14}

15 gc.setStroke(Color.BLACK);
16 gc.setFill(Color.BLACK);
17 double[] coordinateOfx = new double[numOfdata];
18 double[] coordinateOfy = new double[numOfdata];

19 for(int i = 0; i < numOfdata; i++){
20     for(int j = 1; j < (staOfvertical / 50) + 1; j++){ //背景の網目の描画-1
21 gc.strokeLine(200 + 130 / (staOfvertical/50) * j *
22       Math.cos(-(Math.PI/2) + i * 2*Math.PI / numOfdata),
23       200 + 130 / (staOfvertical/50) * j *
24       Math.sin(-(Math.PI/2) + i * 2*Math.PI / numOfdata),
25       200 + 130 / (staOfvertical/50) * j *
26       Math.cos(-(Math.PI/2) + (i+1) * 2*Math.PI / numOfdata),
27       200 + 130 / (staOfvertical/50) * j *
```

```
28       Math.sin(-(Math.PI/2) + (i+1) * 2*Math.PI / numOfdata));
29     }

    //多角形の内部を塗りつぶすために、データの x,y 座標をそれぞれ配列に格納
30     coordinateOfx[i] = 200 + (130 * data[i] / staOfvertical) *
31 Math.cos(-(Math.PI/2) + (i * 2*Math.PI / numOfdata));
32     coordinateOfy[i] = 200 + (130 * data[i] / staOfvertical) *
33 Math.sin(-(Math.PI/2) + (i * 2*Math.PI / numOfdata));
34 }

//レーダーチャートの内部の塗りつぶし
35 gc.setFill(Color.CORNFLOWERBLUE);
36 gc.fillPolygon(coordinateOfx, coordinateOfy, numOfdata);
37 gc.setFill(Color.BLACK);



38 for(int i = 0; i < numOfdata; i++){
39     gc.strokeLine(200, 200,  //背景の網目の描画-
40   200 + 130 *
41   Math.cos(-(Math.PI/2) + (i * 2*Math.PI / numOfdata)),
42   200 + 130 *
43   Math.sin(-(Math.PI/2) + (i * 2*Math.PI / numOfdata)));

44     gc.fillText(String.valueOf(i+1), //データ番号の記述
45 196 + 140 *
46 Math.cos(-(Math.PI/2) + (i * 2*Math.PI / numOfdata)),
47 204 + 140 *
48 Math.sin(-(Math.PI/2) + (i * 2*Math.PI / numOfdata)));

    //各データを直線で結ぶ-1
49     gc.strokeLine(200 + (130 * data[i] / staOfvertical) *
50   Math.cos(-(Math.PI/2) + (i * 2*Math.PI / numOfdata)),
51   200 + (130 * data[i] / staOfvertical) *
52   Math.sin(-(Math.PI/2) + (i * 2*Math.PI / numOfdata)),
53   200 + (130 * data[i+1] / staOfvertical) *
54   Math.cos(-(Math.PI/2) + ((i+1) * 2*Math.PI / numOfdata)),
55   200 + (130 * data[i+1] / staOfvertical) *
56   Math.sin(-(Math.PI/2) + ((i+1) * 2*Math.PI / numOfdata)));
57 }
//各データを直線で結ぶ-2(最後の直線)
```

```
58 gc.strokeLine(200 + (130 * data[numOfdata-1] / staOfvertical) *
59       Math.cos(-(Math.PI/2) + ((numOfdata-1) * 2*Math.PI / numOfdata)),
60       200 + (130 * data[numOfdata-1] / staOfvertical) *
61       Math.sin(-(Math.PI/2) + ((numOfdata-1) * 2*Math.PI / numOfdata)),
62       200,200 - (130 * data[0] / staOfvertical));
```

//縦軸の値の記述

```
63 for(int i = 0; i < (staOfvertical / 50) + 1; i++){
64     gc.fillText(String.valueOf(i * 50 ),
65 203, 200 - (i * 6500  / staOfvertical));
66 }
67 }
68}
```

１～１４行目；データの最大値を求めて縦軸の値をいくつにするか計算する。
１９～２９，３８～４８行目：背景の網目を描画
１５～１８，３０～３４行目：多角形の内部の塗りつぶしのために座標を配列に格納
４９～６２：塗りつぶしだけでなく多角形を黒線でなぞりたいため、直線を描画

3.3　プログラムの実行

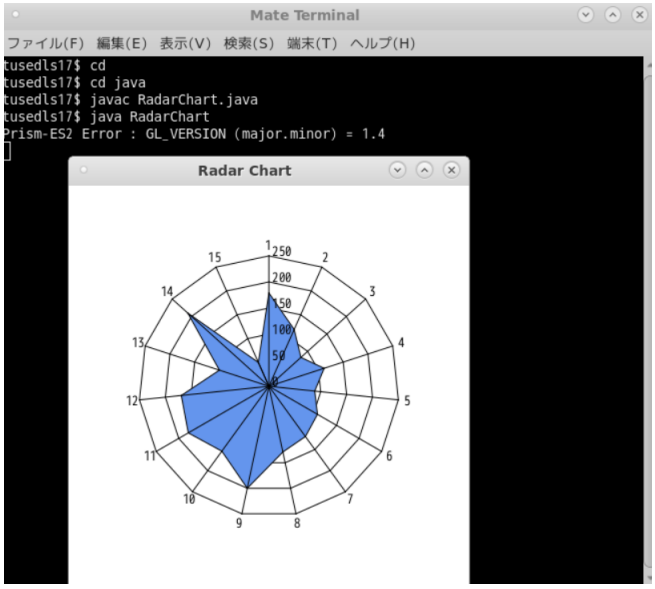

図 8　data.txt の実行結果

3.4　考察

棒グラフと円グラフのレポートで改善点や描画方法は考察したのであまり書くことがありませんでした。棒グラフと同様に、データの値の最大値が大きな数値だったときに、背景の網目が複雑になりすぎてしまうことが改善点であるので、レーダーチャートでも外れ値検出をする必要がる。

4　参考文献

・外れ値検出 - Qiita
qiita.com/tk-tatsuro/items/a49ccab4441dc1dfc86c