Approximation eines Kreuzgewölbes anhand von Messpunkten und Berechnung von dessen Oberfläche

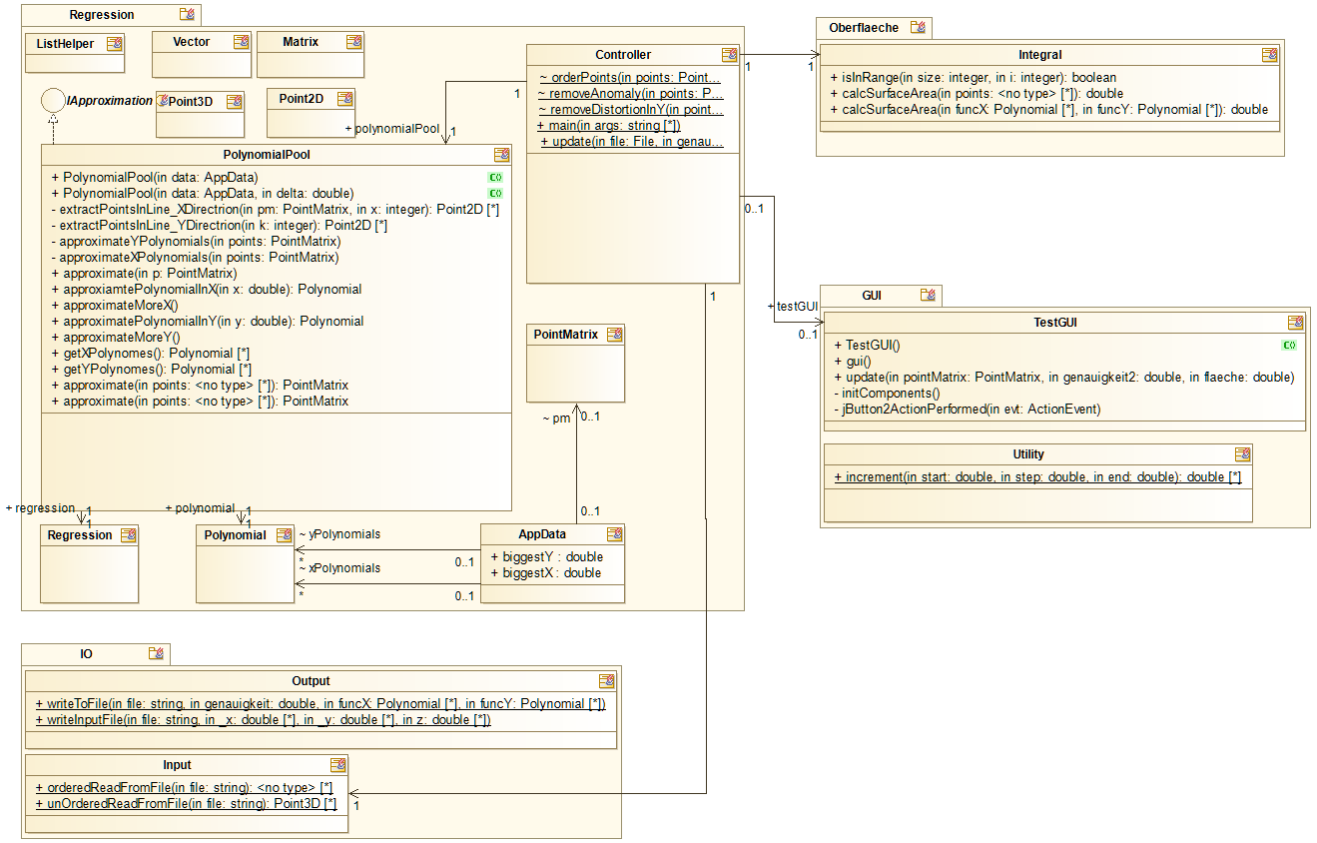
Dokumentation zur Fächerübergreifenden Projektarbeit

Eingereicht von:
Sascha Knapp
Paul Söldner

Datum: 23. Mai 2017

Inhaltsverzeichnis

1 Klassendiagramm



2 Input

Zum Einlesen der Punkte aus einer Textdatei, werden die x-, y- und z-Werte der Punkte zeilenweise in der Datei gespeichert und jeweils mit einem Leerzeichen getrennt. Die Punkte können dabei unsortiert an das Programm übergeben werden.

Listing 1: Input-Datei

```
x_1 y_1 z_1
x_2 y_2 z_2
...
```

Die Datei wird mit Hilfe eines FileReaders und BufferedReaders eingelesen. Die einzelnen Zeilen werden an Leerzeichen und Tabulatoren getrennt und als Punkt-Objekte einer

ArrayList hinzugefügt.

Listing 2: Input-Quellcode

```
public static ArrayList<ArrayList<Point3D>> orderedReadFromFile(String file)
    throws IOException {
      FileReader fr = new FileReader(file);
      BufferedReader br = new BufferedReader(fr);

      String zeile = "";
      int zeilennummer = 0;
      ArrayList<ArrayList<Point3D>> ergebnis = new
          ArrayList<ArrayList<Point3D>>();
      ArrayList<Point3D> liste;
      Point3D p;
      while ((zeile = br.readLine()) != null) {
         if (zeile.isEmpty())
            continue;
         String[] split = zeile.split("\\s+");
         liste = new ArrayList<Point3D>();
         for (int i = 0; i < split.length; i += 3) {

            p = new Point3D(Double.parseDouble(split[i]),
                Double.parseDouble(split[i + 1]),
                  Double.parseDouble(split[i + 2]));
            // p.getPunkt();
            liste.add(p);
         }
         ergebnis.add(zeilennummer, liste);
         zeilennummer++;
      }

      br.close();

      System.out.println("File '" + file + "' wurde erfolgreich eingelesen.");

      return ergebnis;
   }
```

3 Testdaten - Erzeugung

Um Testdaten zu erzeugen haben wir uns eine konkrete Funktion gesucht. Diese wird in der Klasse „GenerateTest“benutzt.

Listing 3: Funktion zur Testdaten-Erzeugung

```
public static double f1(double x, double y) {
   double z = 100 - x*x - y*y;
   return z;
}
```

Die Funktion wird in einem Bereich x und y betrachtet. Diese Bereiche werden jeweils in flachen Feldern gespeichert. Die z-Werte der Funktion an den jeweiligen x und y werden in einem 2-Dimensionalen Feld gespeichert. Die gefüllten Felder werden an die „Output"-Klasse gegeben, die Punkte getrennt von Absätzen in ein Textdokument schreibt.

4 Approximation - Regressionspolynome

Die Klasse „PolynomialPool" stellt Methoden zum Berechnen der ersten Polynome anhand der gemessenen Punkte zur Verfügung. Dabei werden alle in einer Reihe liegenden Punkte in eine ArrayList gegeben und an die Klasse „Regression"gegeben. Hier wird auch angegeben, welchen Grad die Polynome haben sollen. Wir haben eine Konstante mit dem Wert von 2 Grad gewählt.

Listing 4: Polynome mit x als Raumkoordinate

```
private void approximateXPolynomials(PointMatrix points) //wenn noch keine
    Sttzpunkte vorhanden sind
{
   for(int i = 0; i < points.stepsInXDirection(); i++)
   {
      Regression r = new Regression();
      ArrayList<Point2D> p = extractPointsInLine_XDirectrion(points, i);
      Polynomial polynomial = r.approximate(p, kDegree);
      polynomial.setRoomCoordinate(data.pm.getPoint(i, 0).getX());
      data.xPolynomials.add(polynomial);
   }
}
```

5 Berechnung der Oberfläche

Ein einzelnes Flächenstück ergibt sich aus $|\overrightarrow{\boldsymbol{r_u}} \times \overrightarrow{\boldsymbol{r_v}}|\Delta u \Delta v$
Die folgende Methode berechnet dies nun für jedes Polynom und iteriert über die Stützpunkte, um die entsprechenden Funktionswerte z für x und y auszurechnen. Dabei wird in der Methode die Krümmung an den Stellen außer Acht gelassen.

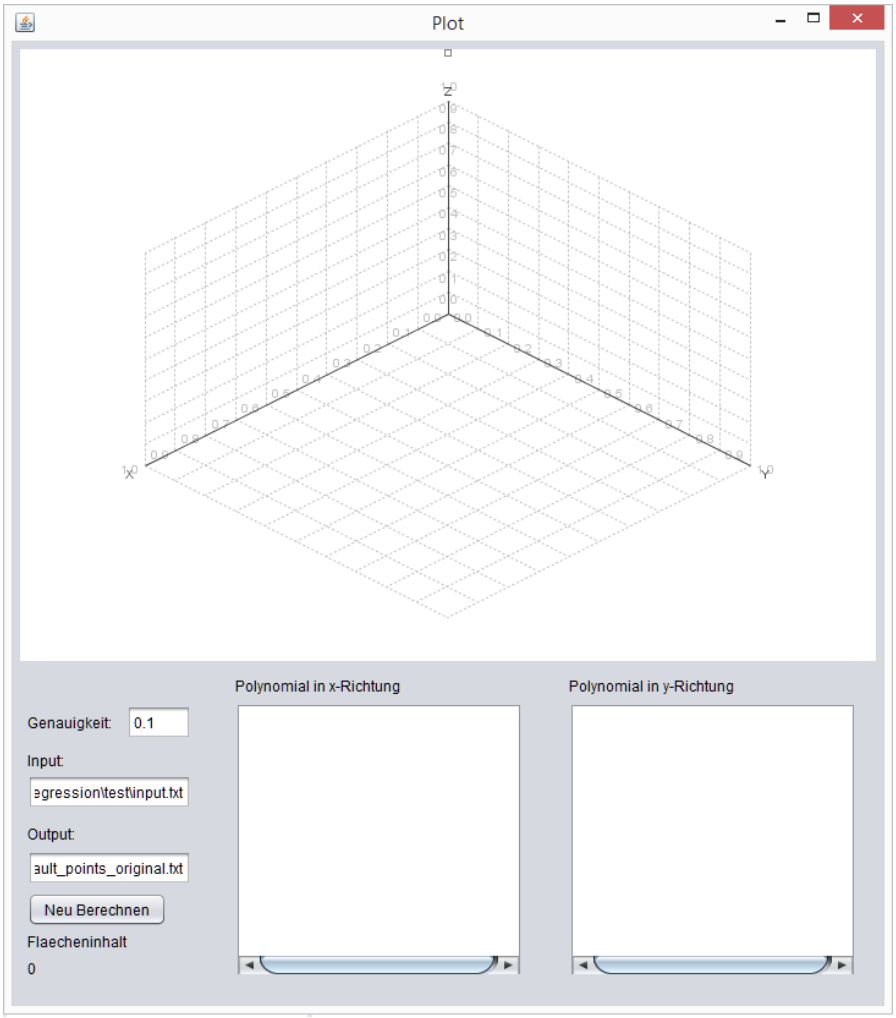
```
public double calcSurfaceArea(ArrayList<Polynomial> funcX,
    ArrayList<Polynomial> funcY)
{
    Double result = 0.0;
    for(int i = 0; i < funcX.size()-1; i++)
    {
        for(int j = 0; j < funcY.size()-1; j++)
        {
            Polynomial currFuncX = funcX.get(i);
            Polynomial nextFuncX = funcX.get(i+1);
            Polynomial currFuncY = funcY.get(j);
            Polynomial nextFuncY = funcY.get(j+1);
            Point3D px1 = new Point3D(currFuncX.getRoomCoordinate(),
                currFuncY.getRoomCoordinate(),
                currFuncX.derivation(currFuncY.getRoomCoordinate()));
            Point3D py1 = new Point3D(currFuncX.getRoomCoordinate(),
                currFuncY.getRoomCoordinate(),
                currFuncY.derivation(currFuncX.getRoomCoordinate()));
            Point3D px2 = new Point3D(nextFuncX.getRoomCoordinate(),
                currFuncY.getRoomCoordinate(),
                nextFuncX.derivation(currFuncY.getRoomCoordinate()));
            Point3D py2 = new Point3D(currFuncX.getRoomCoordinate(),
                nextFuncY.getRoomCoordinate(),
                nextFuncY.derivation(currFuncX.getRoomCoordinate()));

            Double xDelta = px2.getX()-px1.getX();
            Double yDelta = py2.getY()-py1.getY();

            Vector rx = new Vector(px2.getX()-px1.getX(),
                px2.getY()-px1.getY(), px2.getZ()-px1.getZ());
            Vector ry = new Vector(py2.getX()-py1.getX(),
                py2.getY()-py1.getY(), py2.getZ()-py1.getZ());

            Vector crossProduct = new Vector(rx.y() *
                ry.z()-rx.z()*ry.y(),rx.z()*ry.x()-rx.x()*ry.z(),
                rx.x()*ry.y()-rx.y()*ry.x());
            Double sum = crossProduct.sum()*xDelta*yDelta;
            result += sum;
        }
    }
    System.out.println("Oberflche: " + result + " FE");
    return result;
}
```

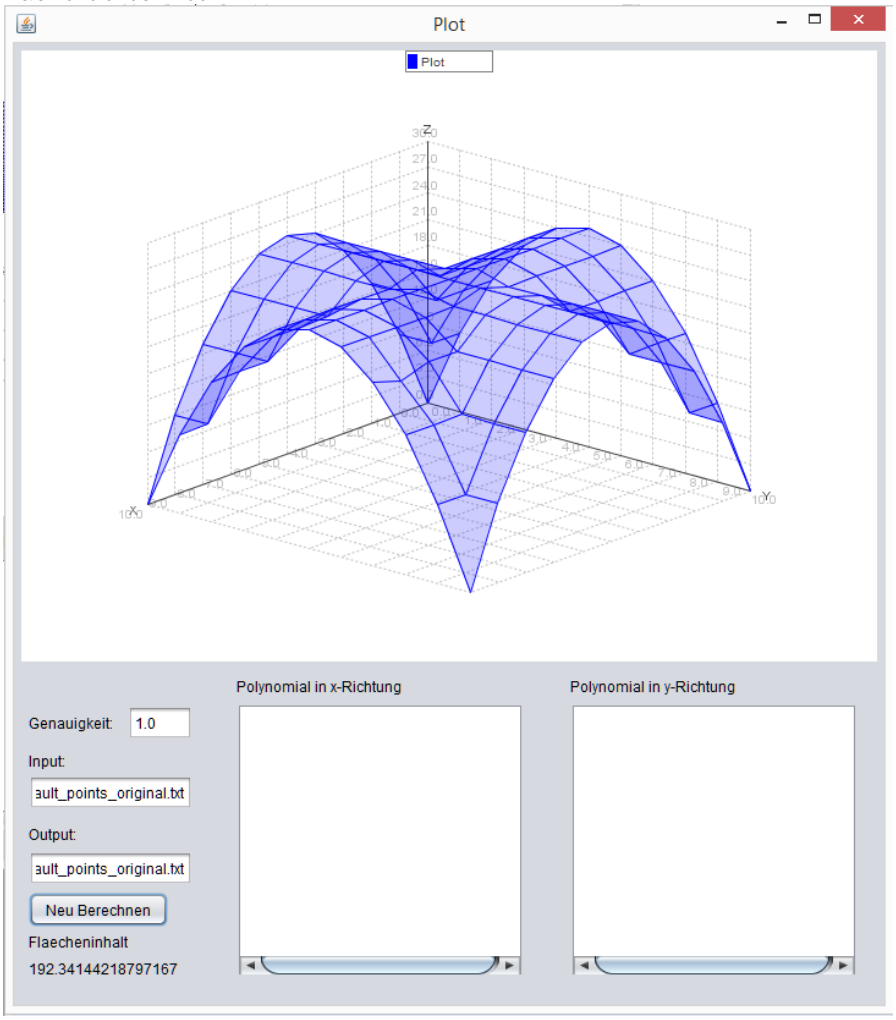
6 Benutzeroberfläche



Die Benutzeroberfläche wurde mit Hilfe des Tools „JMathPlot“ erstellt, die dem Benutzer nach dem Einlesen der Eingabedatei direkt den dazugehörigen Plot anzeigt.
Es können die Pfade zu der Eingabe- bzw. Ausgabedatei und die Genauigkeit eingeben werden. Die Genauigkeit bestimmt, wie viele Extrapunkte zwischen den eingegeben berechnet werden.
Diese Eingaben werden an den „Controller“ weitergegeben, der ggf. weitere Punkte und anschließend die Fläche berechnen lässt. Am Ende wird dem Benutzer auf der GUI der Oberflächeninhalt angezeigt.

7 Beispiel

Werden perfekte Punkte (in x- und y-Richtung auf einer Linie) an das Programm übergeben, wird der Plot korrekt angezeigt und eine näherungsweise korrekte Oberfläche berechnet.



8 Quellen

- JMathPlot - https://github.com/yannrichet/jmathplot

9 Ausblick

9.1 Echte Punkte

Derzeit ist das Programm nur mit „perfekten“ Punkten lauffähig. Es fehlt eine funktionierende Routine, die Punkte, die nicht in einer Reihe liegen, so zurecht zu rückt, dass auch Regressionspolynome generiert werden können.

9.2 Fehlerbehebung bei Verfeinerung

Die Möglichkeit der Verfeinerung weist auch noch Schwächen auf. So treten nach einer Verfeinerung Fehler bei der grafischen Darstellung auf und bei der Berechnung der Oberflläche.

9.3 Verbesserung der Oberflächenberechnung

Die Oberfläche wird derzeit durch eine sehr einfache Näherung berechnet, die für ein genaues Ergebnis sehr lange für die Berechnung benötigt. Die Krümmung der Fläche wird dabei nicht betrachtet. Es könnten allerdings genauere Ergebnisse in viel kürzerer Zeit ermittelt werden, wenn diese auch untersucht würde.