```java
package hyperDap.base.testHelpers;

import static org.junit.Assert.assertEquals;
import java.util.ArrayList;
import org.junit.jupiter.api.Test;
import hyperDap.base.types.dataSet.ValueDataSet;

public class TestCalcDerivDepth {

  private ValueDataSet<Double> makeSet() {
    return new ValueDataSet<Double>(0, 1, 0.001, d -> Double.valueOf(d));
  }

  // simple polynomials
  //
  ********************************************************************************
  ******************************

  @Test
  void constant() {
    double value = 5.0;
    ValueDataSet<Double> set = makeSet();
    for (int i = 0; i < 50; i++) {
      set.add(value);
    }
    set.calcDerivDepths();
    for (int i = 0; i < set.size(); i++) {
      assertEquals(0, set.getDerivDepthsByIndex(i));
    }
  }

  @Test
  void linear() {
    ValueDataSet<Double> set = makeSet();
    for (int i = 0; i < 50; i++) {
      set.add(5.0 + i);
    }
    set.calcDerivDepths();
    for (int i = 0; i < set.size(); i++) {
      // System.out.println(set.getDerivDepthsByIndex(i));
      assertEquals(1, set.getDerivDepthsByIndex(i));
    }
  }

  @Test
  void square() {
    ValueDataSet<Double> set = makeSet();
    for (int i = 0; i < 50; i++) {
      set.add(5.0 + Math.pow(i, 2));
    }
    set.calcDerivDepths();
    for (int i = 0; i < set.size(); i++) {
      // System.out.println(set.getDerivDepthsByIndex(i));
      assertEquals(2, set.getDerivDepthsByIndex(i));
    }
  }

  @Test
  void cubic() {
    int power = 3;
    ValueDataSet<Double> set = makeSet();
    for (int i = 0; i < 50; i++) {
      set.add(5.0 + Math.pow(i, power));
    }
    set.calcDerivDepths();
    for (int i = 0; i < set.size(); i++) {
      // System.out.println(set.getDerivDepthsByIndex(i));
      assertEquals(power, set.getDerivDepthsByIndex(i));
    }
  }

  @Test
  void quad() {
```

```
72        int power = 4;
73        ValueDataSet<Double> set = makeSet();
74        for (int i = 0; i < 50; i++) {
75          set.add(5.0 + Math.pow(i, power));
76        }
77        set.calcDerivDepths();
78        for (int i = 0; i < set.size(); i++) {
79          // System.out.println(set.getDerivDepthsByIndex(i));
80          assertEquals(power, set.getDerivDepthsByIndex(i));
81        }
82      }
83
84      @Test
85      void polynom5() {
86        int power = 5;
87        ValueDataSet<Double> set = makeSet();
88        for (int i = 0; i < 50; i++) {
89          set.add(5.0 + Math.pow(i, power));
90        }
91        set.calcDerivDepths();
92        for (int i = 0; i < set.size(); i++) {
93          // System.out.println(set.getDerivDepthsByIndex(i));
94          assertEquals(power, set.getDerivDepthsByIndex(i));
95        }
96      }
97
98      @Test
99      void polynom6() {
100       int power = 6;
101       ValueDataSet<Double> set = makeSet();
102       for (int i = 0; i < 50; i++) {
103         set.add(5.0 + Math.pow(i, power));
104       }
105       set.calcDerivDepths();
106       for (int i = 0; i < set.size(); i++) {
107         // System.out.println(set.getDerivDepthsByIndex(i));
108         assertEquals(power, set.getDerivDepthsByIndex(i));
109       }
110     }
111
112     // @Test
113     void polynom7() {
114       int power = 7;
115       ValueDataSet<Double> set = makeSet();
116       for (int i = 0; i < 50; i++) {
117         set.add(5.0 + Math.pow(i, power));
118       }
119       set.calcDerivDepths();
120       for (int i = 0; i < set.size(); i++) {
121         // System.out.println(set.getDerivDepthsByIndex(i));
122         assertEquals(power, set.getDerivDepthsByIndex(i));
123       }
124     }
125
126     // @Test
127     void polynom8() {
128       int power = 8;
129       ValueDataSet<Double> set = makeSet();
130       for (int i = 0; i < 50; i++) {
131         set.add(5.0 + Math.pow(i, power));
132       }
133       set.calcDerivDepths();
134       for (int i = 0; i < set.size(); i++) {
135         // System.out.println(set.getDerivDepthsByIndex(i));
136         assertEquals(power, set.getDerivDepthsByIndex(i));
137       }
138     }
139
140     // @Test // here the maxDepth is reached and Integer.MAX_VALUE is assigned
141     void polynom9() {
142       int power = 9;
143       ValueDataSet<Double> set = makeSet();
144       for (int i = 0; i < 50; i++) {
```

```
145         set.add(5.0 + Math.pow(i, power));
146       }
147       set.calcDerivDepths();
148       for (int i = 0; i < set.size(); i++) {
149         // System.out.println(set.getDerivDepthsByIndex(i));
150         assertEquals(Integer.MAX_VALUE, set.getDerivDepthsByIndex(i));
151       }
152     }
153
154     // changes
155     //
      ********************************************************************************
      ******************************
156
157     @Test
158     void constantToLinear() {
159       ValueDataSet<Double> set = makeSet();
160       for (int i = 0; i < 50; i++) {
161         set.add(5.0);
162       }
163       for (int i = 1; i < 51; i++) {
164         set.add(5.0 + i);
165       }
166       set.calcDerivDepths();
167       // System.out.println("\nderivDepths:");
168       for (int i = 0; i < 49; i++) {
169         // System.out.println(String.format("%s: %s", i, set.getDerivDepthsByIndex(i)));
170         assertEquals(0, set.getDerivDepthsByIndex(i));
171       }
172       // System.out.println(String.format("%s: %s", 49, set.getDerivDepthsByIndex(49)));
173       assertEquals(-1, set.getDerivDepthsByIndex(49));
174       for (int i = 50; i < set.size(); i++) {
175         // System.out.println(String.format("%s: %s", i, set.getDerivDepthsByIndex(i)));
176         assertEquals(1, set.getDerivDepthsByIndex(i));
177       }
178     }
179
180     @Test
181     void constantToSquare() {
182       int power = 2;
183       ValueDataSet<Double> set = makeSet();
184       for (int i = 0; i < 50; i++) {
185         set.add(5.0);
186       }
187       for (int i = 1; i < 51; i++) {
188         set.add(5.0 + Math.pow(i, power));
189       }
190       set.calcDerivDepths();
191       // System.out.println("\nderivDepths:");
192       for (int i = 0; i < 49; i++) {
193         // System.out.println(String.format("%s: %s", i, set.getDerivDepthsByIndex(i)));
194         assertEquals(0, set.getDerivDepthsByIndex(i));
195       }
196       // System.out.println(String.format("%s: %s", 49, set.getDerivDepthsByIndex(49)));
197       assertEquals(-1, set.getDerivDepthsByIndex(49));
198       for (int i = 50; i < set.size(); i++) {
199         // System.out.println(String.format("%s: %s", i, set.getDerivDepthsByIndex(i)));
200         assertEquals(power, set.getDerivDepthsByIndex(i));
201       }
202     }
203
204     @Test
205     void squareToConstant() {
206       int power = 2;
207       ValueDataSet<Double> set = makeSet();
208       ArrayList<Integer> depthsExpected = new ArrayList<Integer>();
209       double temp = 0;
210       for (int i = 0; i < 25; i++) {
211         temp = Math.pow(i, power);
212         set.add(temp);
213         depthsExpected.add(power);
214       }
215       depthsExpected.remove(depthsExpected.size() - 1);
```

```
216       depthsExpected.add(-1);
217       for (int i = 0; i < 25; i++) {
218         set.add(temp);
219         depthsExpected.add(0);
220       }
221       for (int i = 0; i < set.size(); i++) {
222         // System.out.println(String.format("%s: %s ?= %s", i,
          set.getDerivDepthsByIndex(i),
223         // depthsExpected.get(i)));
224         assertEquals(depthsExpected.get(i).intValue(), set.getDerivDepthsByIndex(i));
225       }
226     }
227
228     @Test
229     void linearToSquare() {
230       int power = 2;
231       double base = 0.0;
232       double step = 1.0;
233       ValueDataSet<Double> set = makeSet();
234       ArrayList<Integer> depthsExpected = new ArrayList<Integer>();
235       double temp = 0;
236       for (int i = 0; i < 25; i++) {
237         temp = i;
238         set.add(temp);
239         depthsExpected.add(1);
240       }
241       depthsExpected.add(-1);
242       for (int i = 1; i < 25; i++) {
243         set.add(temp + Math.pow(i, power));
244         depthsExpected.add(power);
245       }
246       for (int i = 0; i < set.size(); i++) {
247         // System.out.println(String.format("%s: %s", i, set.getDerivDepthsByIndex(i)));
248         assertEquals(depthsExpected.get(i).intValue(), set.getDerivDepthsByIndex(i));
249       }
250     }
251
252     @Test
253     void biasInConstant() {
254       ValueDataSet<Double> set = makeSet();
255       for (int i = 0; i < 50; i++) {
256         set.add(5.0);
257       }
258       for (int i = 1; i < 51; i++) {
259         set.add(10.0);
260       }
261       set.calcDerivDepths();
262       // System.out.println("\nderivDepths:");
263       for (int i = 0; i < 49; i++) {
264         // System.out.println(String.format("%s: %s", i, set.getDerivDepthsByIndex(i)));
265         assertEquals(0, set.getDerivDepthsByIndex(i));
266       }
267       // System.out.println(String.format("%s: %s", 49, set.getDerivDepthsByIndex(49)));
268       assertEquals(-1, set.getDerivDepthsByIndex(49));
269       for (int i = 50; i < set.size(); i++) {
270         // System.out.println(String.format("%s: %s", i, set.getDerivDepthsByIndex(i)));
271         assertEquals(0, set.getDerivDepthsByIndex(i));
272       }
273     }
274
275     // further polynomial
276     //
      ********************************************************************************
      *********************************
277
278     @Test
279     void sqareWithinConstant() {
280       int power = 2;
281       double base = 0;
282       double step = 1;
283       double value = 5.0;
284       ValueDataSet<Double> set = makeSet();
285       for (int i = 0; i < 30; i++) {
```

```
286         set.add(value);
287       }
288       double temp = 0;
289       for (int i = 0; i < 15; i++) {
290         temp = value + Math.pow(base + i * step, power);
291         set.add(temp);
292       }
293       for (int i = 0; i < 30; i++) {
294         set.add(temp);
295       }
296       for (int i = 0; i < 30; i++) {
297         assertEquals(0, set.getDerivDepthsByIndex(i));
298       }
299       assertEquals(-1, set.getDerivDepthsByIndex(30));
300       for (int i = 31; i < 30 + 14; i++) {
301         assertEquals(power, set.getDerivDepthsByIndex(i));
302       }
303       assertEquals(-1, set.getDerivDepthsByIndex(30 + 14));
304       for (int i = 30 + 15; i < set.size(); i++) {
305         assertEquals(0, set.getDerivDepthsByIndex(i));
306       }
307       // for (int i = 0; i < set.size() - 10; i++) {
308       // System.out.println(String.format("%s: %s", i, set.getDerivDepthsByIndex(i)));
309       // }
310     }
311
312     @Test
313     void squareOverZero() {
314       int power = 2;
315       double base = -10;
316       double step = 0.1;
317       ValueDataSet<Double> set = makeSet();
318       for (int i = 0; i < 500; i++) {
319         set.add(Math.pow(base + i * step, power));
320       }
321       for (int i = 0; i < set.size(); i++) {
322         // System.out.println(set.getDerivDepthsByIndex(i));
323         assertEquals(power, set.getDerivDepthsByIndex(i));
324       }
325     }
326
327   }
328   // end of class
329
```