```
1
     package hyperDap.generator.main;
3
     import java.util.ArrayList;
4
     import java.util.Random;
5
     import java.util.function.DoubleFunction;
6
     import java.util.function.Function;
7
     import hyperDap.base.helpers.Comparator;
8
     import hyperDap.base.types.dataSet.ValueDataSet;
9
10
     * This class generates a section of data specific to one function.
11
12
      * 
      * The intended use is to initialise a {@code GenSegment} with the intended values
13
      and then retrieve
14
      * lists of data points from {@link #generateValues(double, int)} as needed, before
      leaving the
      * Object to be garbage-collected. calling data generation methods repeatedly with
15
16
      * parameters will produce the same data points within limits given to any
      randomness (this will be
17
      * added later to simulate noise).
      * 
18
19
      * Each Object represents a function of the format \{0 \text{ code a * Func}(x + b) + c\},
      where {@code Func}
20
      * is a mathematical function specified by {@code functionEncoding} at construction
      time. The
2.1
      * function is shifted such that it passes through the {@code intercept} at {@code
      x = -step, which
      * is expected to be the last value before this function begins. This aligns it with
22
      the previous
23
      * values to transition smoothly.
     * 
24
25
     * The {@code functionEncding} may specify: <br>
26
     * {@code constant} : {@code y = c} <br>
27
      * {@code linear} : {@code y = a * (x + b) + c} <br/>
28
      * {@code square} : {@code y = a * (x + b)^2 + c} <br>
     * {@code cubic} : {@code y = a * (x + b)^3 + c} <br >
29
      * {@code exp} : {@code y = a * Math.E^(x + b) + c} <br>
30
31
      * {@code sine} : {@code y = a * sin(x + b) +c} <br>
32
      * 
33
      * Here {@code a} translates to the {@code scale} specified at construction, {@code
     b) to
34
      * {@code shiftX}, while {@code c} is defined at construction such that the function
35
      * returns @{@code intercept} for {@code x=-step}.
36
37
     * @author soenk
38
39
40
    public class GenSegment {
41
42
      private double step;
43
      private double a;
44
      private double b;
45
      private double c;
46
      private Function<Double, Double> func;
47
      private Random rand = new Random();
48
49
       /**
50
       * The default constructor.
51
52
        * @param functionEnccoding A {@link String} endocing of the function to be
53
        * @param scale Used to scale and make the function more or less 'steep' {@code =>
54
        * @param shiftX Shifts the function right or left. Use to fit split up functions
        together (e.g.
55
                 bias) {@code => b}
56
        * @param intercept Used to ensure the first value is in line with previous values
        and is assumed
57
                 to be the last value before this segment. \{@code = f(-step) + c\}
58
59
60
       public GenSegment (String functionEnccoding, double scale, double shiftX, double
```

```
intercept,
 61
             double step) throws IllegalArgumentException {
 62
           this.step = step;
 63
          a = scale;
 64
          b = shiftX;
 6.5
          c = 0;
 66
          this.defineFunction(functionEnccoding);
 67
          c = intercept - f(-step);
 68
          System.out.println(String.format("%s Generating Segment of %s with a= %s, b= %s
          c= %s",
 69
               GenSegment.class, functionEnccoding, this.a, this.b, this.c));
 70
        }
 71
 72
         * Classifies the function represented by this Object based on {@code encoding.}
 73
 74
 75
          * @param encoding A {@link String} encoding of the function that is to be modelled.
 76
 77
        private void defineFunction(String encoding) throws IllegalArgumentException {
 78
          encoding = encoding.toLowerCase();
 79
          switch (encoding) {
             case "constant":
 80
 81
               func = x \rightarrow 0.0;
 82
               break;
 83
             case "linear":
 84
               func = x \rightarrow x;
 8.5
               break;
 86
             case "square":
 87
               func = x \rightarrow Math.pow(x, 2);
 88
               break;
 89
             case "cubic":
 90
               func = x \rightarrow Math.pow(x, 3);
 91
               break;
 92
             case "exp":
 93
               func = x \rightarrow Math.pow(Math.E, x);
 94
               a = a / 1000;
 95
               break;
 96
             case "sine":
 97
               func = x \rightarrow Math.sin(x);
 98
               break;
 99
             default:
100
               throw new IllegalArgumentException (
101
                   String.format("'%s' is not a valid function encoding!", encoding));
102
          }
103
        }
104
105
         ^{\star} Returns a single value of the function specified in this Object.
106
         * 
107
         * This is specified as \{\emptyset code\ a^*\ Function(x + b) + c\}.
108
109
110
          * @param x The {@code xValue} to be fed into the function.
111
          * \texttt{@return} The \{\texttt{@code yValue}\} corresponding to \{\texttt{@code x}\}.
112
        private double f(double x) {
113
114
          return a * this.func.apply(x + b) + c;
115
116
117
        /**
118
         * Returns a single value of the function specified by this Object with added
         noise. Noise here is
119
          * a value added to {@link #f(double)}, that is randomly taken from a normal
         distribution around
120
         * zero and of standard deviation {@code noise}.
121
         * @param x The {@code xValue} that is passed to {@link #f(double)}
122
          * @param noise The {@code standard deviation} of the value added (or subtracted).
123
124
          * @return {@link #f(double) f(x)} {@code + noise *} {@link Random#nextGaussian()}.
125
126
        private double noisyF(double x, double noise) {
127
          return f(x) + noise * rand.nextGaussian();
128
        }
129
```

```
/**
130
131
         * Provides a means to reseed the internal instance of {@link Random}.
132
133
         ^{\star} This may not mean that generated data can be used for machine learning, unless
         it is generated
134
         * in small enough segments.
135
136
         * @param seed Passed to {@link Random#setSeed(long)}
137
138
        public void seedRandom(long seed) {
139
          this.rand.setSeed(seed);
140
        }
141
        /**
142
         * Encapsulation of {@link #generateValues(int, double)} with {@code noise=0}.
143
144
         ^{\star} \mbox{\em Qparam} N The number of data points to be generated.
145
         * @return An {@link ArrayList} of the generated data points.
146
147
148
        public ArrayList<Double> generateValues(int N) {
149
          return this.generateValues(N, 0.0);
150
        }
151
152
        /**
153
         * Generate a list of data points of length {@code N}, according to pre-set
         specifications and
154
         * with the set amount of noise.
155
156
         * @param N The number of data points to be generated.
157
         * @param noise The noise factor passed to {@link #noisyF(double, double)}
158
         * @return An {@link ArrayList} of the generated data points.
159
160
        public ArrayList<Double> generateValues(int N, double noise) {
161
          ArrayList<Double> list = new ArrayList<Double>();
162
          for (Integer i = 0; i < N; i++) {</pre>
163
            list.add(noisyF(i.doubleValue() * step, noise));
164
          }
165
          return list;
166
        }
167
168
        /**
169
         * Generate the specified data points and add them to the end of {@code set}.
170
171
         * Calls {@link ValueDataSet#ensureCapacity(int)} before generating data.
172
         * Encapsulates {@link #addToDoubleDataSet(ValueDataSet, int, double)} with {@code
173
         noise=0}.
174
175
         * \ensuremath{	ext{ @param}} set The \ensuremath{	ext{ @link CalueDataSet}} that the data points should be added to.
176
         * @param step The distance between data points on the x-axis.
177
         * @param N The number of data points that should be added.
178
         * @throws IllegalArgumentException If {@link ValueDataSet#getStep()} is not equal
         to the pre-set
179
                    step.
         * /
180
181
        public void addToDoubleDataSet(ValueDataSet<Double> set, int N) throws
        IllegalArgumentException {
182
          this.addToDoubleDataSet(set, N, 0.0);
183
184
        /**
185
         * Generate the specified data points with noise and add them to the end of {@code
186
         set}.
187
         * 
188
         * Calls {@link ValueDataSet#ensureCapacity(int)} before generating data.
189
190
         * Noisy values are created using {@link #noisyF(double, double)}.
191
192
         * @param set The {@link CalueDataSet} that the data points should be added to.
193
         * @param N The number of data points that should be added.
194
         * @param noise The noise factor passed to {@link #noisyF(double, double)}.
195
         * @throws IllegalArgumentException If {@link ValueDataSet#getStep()} is not equal
         to the pre-set
```

```
196
         * /
197
198
        public void addToDoubleDataSet(ValueDataSet<Double> set, int N, double noise)
199
            throws IllegalArgumentException {
200
          int size = set.size();
201
          if (this.step != set.getStep()) {
            throw new IllegalArgumentException (
203
                String.format("%s. addToDoubleDataSet() does not match preset step! %s!=%s",
204
                    GenSegment.class, this.step, set.getStep()));
205
          1
206
          set.ensureCapacity(N + set.size());
207
          double val;
208
          for (Integer i = 0; i < N; i++) {
209
            val = Double.valueOf(noisyF(i.doubleValue() * step, noise));
210
            set.add(val);
211
            // if the value is too large or small, consider it invalid
            // if (Comparator.equalApprox(0.0, val, 10000.0) == false) {
212
213
            // set.editValidityByIndex((i + size), false);
214
            // }
215
          }
216
        }
217
218
219
         * Generate the specified data points and add them to the end of {@code set}.
220
         * This method requires that {@code set} has an assigned {@code convertFromDouble}
221
222
         * {@link DoubleFunction Function} assigned.
         * 
223
224
         * {@link ValueDataSet#ensureCapacity(int)} is called before adding data points.
225
226
         * @param set The {@link CalueDataSet} that the data points should be added to.
227
         * @param N The number of data points that should be added.
         * @throws IllegalArgumentException If {@link
228
         ValueDataSet#hasConversionFunction() } returns
229
                   {@code false}.<br>
230
                   If {@link ValueDataSet#getStep()} is not equal to the pre-set step.
         * /
231
2.32
        public void addToDataSet(ValueDataSet<? extends Number> set, int N)
233
            throws IllegalArgumentException {
234
          this.addToDataSet(set, N, 0.0);
235
        }
236
        /**
237
238
        * Generate the specified data points and add them to the end of {@code set}.
239
         * This method requires that {@code set} has an assigned {@code convertFromDouble}
240
241
         * {@link DoubleFunction Function} assigned.
         * 
242
         * {@link ValueDataSet#ensureCapacity(int)} is called before adding data points.
243
         Noisy values are
244
         * created using {@link #noisyF(double, double)}.
245
246
         * @param set The {@link CalueDataSet} that the data points should be added to.
247
         * @param N The number of data points that should be added.
248
         * @param noise The noise factor passed to {@link #noisyF(double, double)}.
249
         * @throws IllegalArgumentException If {@link
         ValueDataSet#hasConversionFunction() } returns
250
                   {@code false}.<br>
251
                   If {@link ValueDataSet#getStep()} is not equal to the pre-set step.
252
         */
253
        public void addToDataSet(ValueDataSet<? extends Number> set, int N, double noise)
254
            throws IllegalArgumentException {
255
          int size = set.size();
256
          if (set.hasConversionFunction() == false) {
257
            throw new IllegalArgumentException(
258
                "ValueDataSet must have a convertFromDouble function defined!");
259
          }
260
          if (this.step != set.getStep()) {
261
            throw new IllegalArgumentException (
262
                String.format("%s. addToDoubleDataSet() does not match preset step! %s!=%s",
                    GenSegment.class, this.step, set.getStep()));
263
264
265
          set.ensureCapacity(N + set.size());
```

```
266
          double val;
267
          for (Integer i = 0; i < N; i++) {
268
            val = noisyF(i.doubleValue() * step, noise);
269
            set.add(val);
270
            if (Comparator.equalApprox(0.0, val, 10000) == false) {
271
              set.editValidityByIndex(i + size, false);
272
273
          }
274
        }
275
276
        public void addRandomToDoubleDataSet(ValueDataSet<Double> set, int N)
277
            throws IllegalArgumentException {
278
          if (this.step != set.getStep()) {
279
            throw new IllegalArgumentException(
280
                String.format("%s. addToDoubleDataSet() does not match preset step! %s!=%s",
281
                    GenSegment.class, this.step, set.getStep()));
282
283
          set.ensureCapacity(set.size() + N);
284
          Integer temp;
285
          for (int i = 0; i < N; i++) {</pre>
286
            temp = (rand.nextInt(10000) - 5000);
287
            set.add(temp.doubleValue() / 100);
288
          }
289
        }
290
291
        public void addRandomToDataSet(ValueDataSet<? extends Number> set, int N)
292
            throws IllegalArgumentException {
293
          if (set.hasConversionFunction() == false) {
294
            throw new IllegalArgumentException(
                "ValueDataSet must have a convertFromDouble function defined!");
295
296
297
          if (this.step != set.getStep()) {
298
            throw new IllegalArgumentException(
299
                String.format("%s. addToDoubleDataSet() does not match preset step! %s!=%s",
300
                    GenSegment.class, this.step, set.getStep()));
301
          }
302
          set.ensureCapacity(set.size() + N);
303
          Integer temp;
304
          for (int i = 0; i < N; i++) {
305
            temp = (rand.nextInt(10000) - 5000);
306
            set.add(temp.doubleValue() / 100);
307
          }
308
309
        }
310
311
      }
312
```