```
1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using System.Linq;
5
6  namespace TG2_RFID
7  {
8
9      public class Curve
10     {
11         /// <summary>
12         /// Holds the curve's name.
13         /// </summary>
14         protected string curveName;
15
16         /// <summary>
17         /// Holds the curve's data separated as X, Y point.
18         /// </summary>
19         protected SortedList<double, double> curveData;
20
21         /// <summary>
22         /// Holds the maximum amount of points in this curve.
23         /// </summary>
24         static protected int maxNumberOfPoints = 12;
25
26         static protected int averageFilterWindow = 3;
27
28         protected double maximumPointX;
29         protected double maximumPointY;
30
31         protected double minimumPointX;
32         protected double minimumPointY;
33
34         public static void PopulateCurveTest(Curve curve)
35         {
36             //curve.AddPoint(0, .5);
37             //curve.AddPoint(0.1, 0.1);
38             //curve.AddPoint(0.2, 0.2);
39             //curve.AddPoint(0.25, 0.25);
40             //curve.AddPoint(0.4, 0.4);
41             //curve.AddPoint(0.5, 0.5);
42             //curve.AddPoint(0.6, 0.6);
43             //curve.AddPoint(0.9, 0.9);
44             //curve.AddPoint(0.99, .95);
45             //curve.AddPoint(.33, .33);
46             //curve.AddPoint(.7, .8);
47             //curve.AddPoint(.75, .8);
48             //curve.AddPoint(.8, .8);
49             //curve.AddPoint(1.3, .6);
50             //curve.AddPoint(1.4, .4);
51             //curve.AddPoint(1.5, .3);
52             //curve.AddPoint(1.6, .2);
53             //curve.AddPoint(1.7, .1);
54             //curve.AddPoint(1.8, -.1);
55             //curve.AddPoint(1.9, -.1);
56             //curve.AddPoint(2.1, -.2);
```

```
57              //curve.AddPoint(2.15, -.3);
58              //curve.AddPoint(2.2, -.4);
59              //curve.AddPoint(2.225, -.5);
60              //curve.AddPoint(2.25, -.6);
61              //curve.AddPoint(2.275, -.7);
62              //curve.AddPoint(2.3, -.8);
63              //curve.AddPoint(2.4, -.9);
64              //curve.AddPoint(2.5, -.95);
65              //curve.AddPoint(2.7, -.95);
66              ////////////////////////////////////
67              ///
68              curve.AddPointWithAvgFilter(-3.14159265358979, 1.39635420686754);
69              curve.AddPointWithAvgFilter(-3.07812613533545, 1.89995829146925);
70              curve.AddPointWithAvgFilter(-3.01465961708111, 1.48854286693781);
71              curve.AddPointWithAvgFilter(-2.95119309882678, 1.0860811022767);
72              curve.AddPointWithAvgFilter(-2.88772658057244, 0.920492037729703);
73              curve.AddPointWithAvgFilter(-2.8242600623181, 1.3838628891516);
74              curve.AddPointWithAvgFilter(-2.76079354406376, 0.933474681122092);
75              curve.AddPointWithAvgFilter(-2.69732702580942, 0.35450756315884);
76              curve.AddPointWithAvgFilter(-2.63386050755508,
                  0.0534229643943726);
77              curve.AddPointWithAvgFilter(-2.57039398930074, 0.838723714241578);
78              curve.AddPointWithAvgFilter(-2.5069274710464, 0.217962442212655);
79              curve.AddPointWithAvgFilter(-2.44346095279206, 0.410322568792965);
80              curve.AddPointWithAvgFilter(-2.37999443453772, 0.33462340878903);
81              curve.AddPointWithAvgFilter(-2.31652791628338,
                  -0.674969032252129);
82              curve.AddPointWithAvgFilter(-2.25306139802904,
                  -0.804442376513671);
83              curve.AddPointWithAvgFilter(-2.1895948797747, -0.997707724013973);
84              curve.AddPointWithAvgFilter(-2.12612836152037, -1.0413203658205);
85              curve.AddPointWithAvgFilter(-2.06266184326603, -1.25870522517678);
86              curve.AddPointWithAvgFilter(-1.99919532501169, -1.35944318507287);
87              curve.AddPointWithAvgFilter(-1.93572880675735, -1.42452521176546);
88              curve.AddPointWithAvgFilter(-1.87226228850301, -1.30958887273105);
89              curve.AddPointWithAvgFilter(-1.80879577024867, -1.15584663542153);
90              curve.AddPointWithAvgFilter(-1.74532925199433, -1.67184080123879);
91              curve.AddPointWithAvgFilter(-1.68186273373999, -1.65681005379094);
92              curve.AddPointWithAvgFilter(-1.61839621548565, -1.64392135469107);
93              curve.AddPointWithAvgFilter(-1.55492969723131, -1.37223601492581);
94              curve.AddPointWithAvgFilter(-1.49146317897697, -1.92358823367669);
95              curve.AddPointWithAvgFilter(-1.42799666072263, -1.19516369166389);
96              curve.AddPointWithAvgFilter(-1.36453014246829, -1.36492208827684);
97              curve.AddPointWithAvgFilter(-1.30106362421395, -1.08783537085571);
98              curve.AddPointWithAvgFilter(-1.23759710595962, -1.26508751441717);
99              curve.AddPointWithAvgFilter(-1.17413058770528,
                  -0.625362236401926);
100             curve.AddPointWithAvgFilter(-1.11066406945094, -0.56684144603304);
101             curve.AddPointWithAvgFilter(-1.0471975511966, -0.856734378351539);
102             curve.AddPointWithAvgFilter(-0.983731032942258,
                  -0.711052078905264);
103             curve.AddPointWithAvgFilter(-0.920264514687919,
                  -0.282403717588411);
104             curve.AddPointWithAvgFilter(-0.85679799643358,
                  -0.645852139012745);
105             curve.AddPointWithAvgFilter(-0.79333147817924, 0.186706066961859);
```

```
106             curve.AddPointWithAvgFilter(-0.729864959924901,
                  -0.466745932262059);
107             curve.AddPointWithAvgFilter(-0.666398441670562,
                  0.176330072978658);
108             curve.AddPointWithAvgFilter(-0.602931923416223,
                  0.0677179747872264);
109             curve.AddPointWithAvgFilter(-0.539465405161884,
                  0.613412607652294);
110             curve.AddPointWithAvgFilter(-0.475998886907544,
                  0.502621087466802);
111             curve.AddPointWithAvgFilter(-0.412532368653205,
                  0.716935659109958);
112             curve.AddPointWithAvgFilter(-0.349065850398866,
                  0.470792234094891);
113             curve.AddPointWithAvgFilter(-0.285599332144526, 1.09232955297835);
114             curve.AddPointWithAvgFilter(-0.222132813890187, 1.41011741790095);
115             curve.AddPointWithAvgFilter(-0.158666295635848, 1.33875190579228);
116             curve.AddPointWithAvgFilter(-0.095199777381509, 1.00794417153841);
117             curve.AddPointWithAvgFilter(-0.03173325912717, 1.91460655432265);
118             curve.AddPointWithAvgFilter(0.03173325912717, 1.7816862048228);
119             curve.AddPointWithAvgFilter(0.095199777381509, 1.37592051147737);
120             curve.AddPointWithAvgFilter(0.158666295635848, 1.37860742057852);
121             curve.AddPointWithAvgFilter(0.222132813890187, 1.2872818200005);
122             curve.AddPointWithAvgFilter(0.285599332144526, 1.98959707400003);
123             curve.AddPointWithAvgFilter(0.349065850398866, 1.18834528115876);
124             curve.AddPointWithAvgFilter(0.412532368653205, 1.99317242166135);
125             curve.AddPointWithAvgFilter(0.475998886907544, 1.60381272234308);
126             curve.AddPointWithAvgFilter(0.539465405161884, 1.80561808342577);
127             curve.AddPointWithAvgFilter(0.602931923416223, 1.72570961909859);
128             curve.AddPointWithAvgFilter(0.666398441670562, 1.6386525947751);
129             curve.AddPointWithAvgFilter(0.729864959924901, 1.75586085557358);
130             curve.AddPointWithAvgFilter(0.79333147817924, 1.49256828391793);
131             curve.AddPointWithAvgFilter(0.85679799643358, 1.12734384227661);
132             curve.AddPointWithAvgFilter(0.920264514687919, 0.719248621295077);
133             curve.AddPointWithAvgFilter(0.983731032942258, 0.65975286277186);
134             curve.AddPointWithAvgFilter(1.0471975511966, 0.591019662713436);
135             curve.AddPointWithAvgFilter(1.11066406945094, 1.28613295844699);
136             curve.AddPointWithAvgFilter(1.17413058770528, 1.12065310853309);
137             curve.AddPointWithAvgFilter(1.23759710595962, 1.00736033247674);
138             curve.AddPointWithAvgFilter(1.30106362421395, 0.624858768074646);
139             curve.AddPointWithAvgFilter(1.36453014246829, 0.0810994893070038);
140             curve.AddPointWithAvgFilter(1.42799666072263, 0.0881309104803278);
141             curve.AddPointWithAvgFilter(1.49146317897697, 0.0115093591309446);
142             curve.AddPointWithAvgFilter(1.55492969723131, 0.472983281472946);
143             curve.AddPointWithAvgFilter(1.61839621548565, 0.171282500319029);
144             curve.AddPointWithAvgFilter(1.68186273373999, 0.789575346431925);
145             curve.AddPointWithAvgFilter(1.74532925199433, 0.0455145603400934);
146             curve.AddPointWithAvgFilter(1.80879577024867, 0.414848567793334);
147             curve.AddPointWithAvgFilter(1.87226228850301, 0.976707108463735);
148             curve.AddPointWithAvgFilter(1.93572880675735, 0.409173496221809);
149             curve.AddPointWithAvgFilter(1.99919532501169, 0.461345089757619);
150             curve.AddPointWithAvgFilter(2.06266184326603, 1.2881157063007);
151             curve.AddPointWithAvgFilter(2.12612836152037, 1.15891840436669);
152             curve.AddPointWithAvgFilter(2.1895948797747, 0.953258370681676);
153             curve.AddPointWithAvgFilter(2.25306139802904, 0.879173622417986);
154             curve.AddPointWithAvgFilter(2.31652791628338, 1.48330907109987);
```

```
155             curve.AddPointWithAvgFilter(2.37999443453772, 1.21653245695899);
156             curve.AddPointWithAvgFilter(2.44346095279206, 1.7273615984108);
157             curve.AddPointWithAvgFilter(2.5069274710464, 1.8059154068297);
158             curve.AddPointWithAvgFilter(2.57039398930074, 1.91125743207817);
159             curve.AddPointWithAvgFilter(2.63386050755508, 1.89979940409336);
160             curve.AddPointWithAvgFilter(2.69732702580942, 1.21360887389898);
161             curve.AddPointWithAvgFilter(2.76079354406376, 1.47324949905646);
162             curve.AddPointWithAvgFilter(2.8242600623181, 2.09487483004092);
163             curve.AddPointWithAvgFilter(2.88772658057244, 2.11566891751715);
164             curve.AddPointWithAvgFilter(2.95119309882678, 1.34972184058067);
165             curve.AddPointWithAvgFilter(3.01465961708111, 1.31136365213583);
166             curve.AddPointWithAvgFilter(3.07812613533545, 1.68187849713421);
167             curve.AddPointWithAvgFilter(3.14159265358979, 1.15568185651978);
168         }
169
170         /// <summary>
171         /// Adds a point for the curve.
172         /// </summary>
173         /// <param name="x">The x coordinate.</param>
174         /// <param name="y">The y coordinate.</param>
175         public void AddPoint(double x, double y)
176         {
177             //var watch = System.Diagnostics.Stopwatch.StartNew();
178             try
179             {
180                 curveData.Add(x, y);
181                 if (maximumPointY < y)
182                 {
183                     maximumPointY = y;
184                     maximumPointX = x;
185                 }
186                 if (minimumPointY > y)
187                 {
188                     minimumPointY = y;
189                     minimumPointX = x;
190                 }
191                 while (curveData.Count >= maxNumberOfPoints)
192                 {
193                     var removedX = curveData.Keys[0];
194                     curveData.Remove(curveData.Keys[0]);
195                     if (Double.Equals(removedX, minimumPointX))
196                     {
197                         var XY = CalculateCurveMinPoint();
198                         minimumPointX = XY.Item1;
199                         minimumPointY = XY.Item2;
200                     }
201                     if (Double.Equals(removedX, maximumPointX))
202                     {
203                         var XY = CalculateCurveMaxPoint();
204                         maximumPointX = XY.Item1;
205                         maximumPointY = XY.Item2;
206                     }
207                 }
208             }
209             catch(Exception e)
210             {
```

```
211                 // Handle .NET errors.
212                 Console.WriteLine("Exception : {0}", e.Message);
213             }
214             //watch.Stop();
215             //var elapsedMs = watch.ElapsedMilliseconds;
216             //Console.WriteLine("AddPointTIme in ms : {0}", elapsedMs);
217         }
218 
219         public void AddPointWithAvgFilter(double x, double y)
220         {
221             if (curveData.Count >= averageFilterWindow)
222             {
223                 var avg = y + curveData.Values[curveData.Count - 1] +
                      curveData.Values[curveData.Count - (averageFilterWindow -
                      1)];
224                 avg /= averageFilterWindow;
225                 AddPoint(x, avg);
226             }
227             else
228             {
229                 AddPoint(x, y);
230             }
231         }
232 
233         /// <summary>
234         /// Prints the curve in console.
235         /// </summary>
236         public void PrintCurveInConsole()
237         {
238             const char BLANK = ' ';
239             const char DOT = '.';
240             const char X = 'x';
241             const int cMaxLineChars = 79;
242             const int cHalf = cMaxLineChars / 2;
243             char[] LINE = new char[cMaxLineChars];
244 
245             for (int i = 0; i < LINE.Length; i++)
246             {
247                 LINE[i] = DOT;
248             }
249             Console.WriteLine(LINE);
250             for (int i = 0; i < LINE.Length; i++)
251             {
252                 LINE[i] = BLANK;
253             }
254 
255             int loc;
256             //var maxY = getCurveMaxAbsY();
257             var maxY = 2.2;
258             LINE[cHalf] = DOT;
259             foreach (var pair in curveData)
260             {
261                 loc = (int)Math.Round(cMaxLineChars * (pair.Value + maxY) / (2
                        * maxY));
262                 //loc = (int)Math.Round(cMaxLineChars * (pair.Value + maxY) /
                      (2 * maxY));
```

```
263                 if (loc == LINE.Length)
264                 {
265                     LINE[loc - 1] = X;
266                 } else
267                 {
268                     LINE[loc] = X;
269                 }
270                 Console.WriteLine(LINE);
271                 for (int i = 0; i < LINE.Length; i++)
272                 {
273                     LINE[i] = BLANK;
274                 }
275                 LINE[cHalf] = DOT;
276             }
277         }
278 
279         public void PrintCurveLastValue()
280         {
281             Console.WriteLine(curveData[curveData.Keys[0]]);
282         }
283 
284         public double GetCurveLastValue()
285         {
286             if (curveData.Count != 0)
287             {
288                 var x = curveData.Keys[curveData.Keys.Count - 1];
289                 return curveData[x];
290             }
291             else
292             {
293                 return 0;
294             }
295 
296         }
297 
298         /*!
299          * Writes info about cardholder: name, EPC, Ambient and time-entered-
              ambient
300          */
301         public void WriteDataToFile()
302         {
303 
304         }
305 
306         /// <summary>
307         /// Getter the minimum value for x coordinate.
308         /// </summary>
309         /// <returns>The curve minimum x.</returns>
310         public double GetCurveMinX()
311         {
312             return curveData.Keys[0];
313         }
314 
315         /// <summary>
316         /// Getter for the index's value for Y
317         /// </summary>
```

```
318         public double GetCurveIndexY(int i)
319         {
320             if (curveData.Values.Count() > i)
321                 return curveData.Values[i];
322             else
323                 return 0;
324         }
325 
326         /// <summary>
327         /// Getter for the index's value for X
328         /// </summary>
329         public double GetCurveIndexX(int i)
330         {
331             if (curveData.Values.Count() > i)
332                 return curveData.Keys[i];
333             else
334                 return 0;
335         }
336 
337         /// <summary>
338         /// Getter the maximum value for x coordinate.
339         /// </summary>
340         public double GetCurveMaxX()
341         {
342             return curveData.Keys[curveData.Count - 1];
343         }
344 
345         /// <summary>
346         /// Getter the curve coordinate for the y maximum value.
347         /// </summary>
348         public Tuple<double, double> CalculateCurveMaxPoint()
349         {
350             maximumPointX = Double.NegativeInfinity;
351             maximumPointY = Double.NegativeInfinity;
352             if (curveData.Count != 0)
353             {
354                 foreach (var pair in curveData)
355                 {
356                     if (pair.Value > maximumPointY)
357                     {
358                         maximumPointX = pair.Key;
359                         maximumPointY = pair.Value;
360                     }
361                 }
362             }
363 
364             return Tuple.Create<double, double>(maximumPointX, maximumPointY);
365         }
366 
367         public Tuple<double, double> GetCurveMaxPoint()
368         {
369             return Tuple.Create<double, double>(maximumPointX, maximumPointY);
370         }
371 
372         /// <summary>
373         /// Getter the curve coordinate for the y minimum value.
```

```
374         /// </summary>
375         public Tuple<double, double> CalculateCurveMinPoint()
376         {
377             minimumPointX = Double.PositiveInfinity;
378             minimumPointY = Double.PositiveInfinity;
379             foreach (var pair in curveData)
380             {
381                 if (pair.Value < minimumPointY)
382                 {
383                     minimumPointX = pair.Key;
384                     minimumPointY = pair.Value;
385                 }
386             }
387             return Tuple.Create<double, double>(minimumPointX, minimumPointY);
388         }
389 
390         public Tuple<double, double> GetCurveMinPoint()
391         {
392             return Tuple.Create<double, double>(minimumPointX, minimumPointY);
393         }
394 
395         /// <summary>
396         /// Getter the curve median y coordinates.
397         /// </summary>
398         public double GetMedianY()
399         {
400             double medianY = 0;
401             if (curveData.Count > 2)
402             {
403                 var values = curveData.Values;
404                 List<double> list = new List<double>(values);
405                 list.Sort();
406                 if (list.Count % 2 != 0)
407                 {
408                     medianY = list[list.Count / 2];
409                 }
410                 else
411                 {
412                     medianY = list[1 + list.Count / 2];
413                 }
414             }
415             return medianY;
416         }
417 
418         /// <summary>
419         /// Getter the curve mean y coordinates.
420         /// </summary>
421         public double CalculateMeanY()
422         {
423             double sumY = 0;
424             double medianY = 0;
425             foreach (var pair in curveData)
426             {
427                 sumY += pair.Value;
428             }
429             if (curveData.Count != 0)
```

```
430             {
431                 medianY = sumY / curveData.Count;
432             }
433             return medianY;
434         }
435 
436         /// <summary>
437         /// Getter for the crossing treshold between positive and negative
              values.
438         /// </summary>
439         public Tuple<double, double> CalculateCrossingPoint()
440         {
441             //var watch = System.Diagnostics.Stopwatch.StartNew();
442             double crossingPointX = Double.NaN;
443             double crossingPointY = Double.NaN;
444             bool isFirstSignPositive = false;
445             for(int i = curveData.Count - 1; i >= 0; i--)
446             {
447                 var x = curveData.Keys[i];
448                 var y = curveData[x];
449                 if (i == curveData.Count - 1)
450                 {
451                     isFirstSignPositive = y > 0 ? true : false;
452                     continue;
453                 }
454                 //if ((y < 0 && isFirstSignPositive) || (y > 0 && !
                      isFirstSignPositive))
455                 if ((y > 0 && !isFirstSignPositive))
456                 {
457                     if (i < curveData.Count - 2)
458                     {
459                         x += curveData.Keys[i + 1];
460                         x *= 0.5;
461                         y += curveData[curveData.Keys[i + 1]];
462                         y *= 0.5;
463                     }
464                     crossingPointY = x;
465                     crossingPointX = y;
466                     break;
467                 }
468             }
469             //watch.Stop();
470             //var elapsedMs = watch.ElapsedMilliseconds;
471             //Console.WriteLine("CrossingPoint in ms : {0}", elapsedMs);
472             return Tuple.Create(crossingPointX, crossingPointY);
473         }
474 
475         /// <summary>
476         /// Constructor for a curve.
477         /// </summary>
478         public Curve()
479         {
480             curveData = new SortedList<double, double>();
481             maximumPointX = Double.NegativeInfinity;
482             maximumPointY = Double.NegativeInfinity;
483             minimumPointX = Double.PositiveInfinity;
```

```
484             minimumPointY = Double.PositiveInfinity;
485         }
486
487         public int GetSize()
488         {
489             return curveData.Count;
490         }
491
492         public IList<Tuple<double, double>> CalculatePeaks()
493         {
494             //var watch = System.Diagnostics.Stopwatch.StartNew();
495             IList<double> values = curveData.Values;
496             int rangeOfPeaks = (int)(Math.Round(.3 * values.Count));
497
498             List<Tuple<double, double>> peaks = new List<Tuple<double,
                  double>>();
499
500             double current;
501             IEnumerable<double> range;
502
503             int checksOnEachSide = rangeOfPeaks / 2;
504             if (values.Count == 0)
505             {
506                 return peaks;
507             }
508             else if (values.Count < checksOnEachSide)
509             {
510                 peaks.Add(Tuple.Create(maximumPointX, maximumPointY));
511             }
512             else
513             {
514                 for (int i = 0; i < values.Count; i++)
515                 {
516                     current = values[i];
517                     range = values;
518
519                     if (i > checksOnEachSide)
520                     {
521                         range = range.Skip(i - checksOnEachSide);
522                     }
523
524                     range = range.Take(rangeOfPeaks);
525                     if ((range.Count() > 0) && (current == range.Max()))
526                     {
527                         peaks.Add(Tuple.Create(curveData.Keys[i],
                        curveData.Values[i]));
528                     }
529                 }
530             }
531             //watch.Stop();
532             //var elapsedMs = watch.ElapsedMilliseconds;
533             //Console.WriteLine("PeaksPoint in ms : {0}", elapsedMs);
534             return peaks;
535         }
536
537         public bool CompareCurveMaximums(Curve otherCurve)
```

```
538         {
539             return GetCurveMaxPoint().Item2 > otherCurve.GetCurveMaxPoint
                ().Item2;
540         }
541
542         public bool CompareCurveLastValues(Curve otherCurve)
543         {
544             return GetCurveLastValue() > otherCurve.GetCurveLastValue();
545         }
546
547         public bool CompareCurveMeans(Curve otherCurve)
548         {
549             return CalculateMeanY() >= otherCurve.CalculateMeanY();
550         }
551
552         public bool CompareCurveMedians(Curve otherCurve)
553         {
554             return GetMedianY() >= otherCurve.GetMedianY();
555         }
556
557         public bool CompareCurveLastPeak(Curve otherCurve)
558         {
559             var peakListLast = CalculatePeaks();
560             var peakListOther = otherCurve.CalculatePeaks();
561
562             return (peakListLast.Count > 0 && peakListOther.Count > 0 &&
                (peakListLast[peakListLast.Count - 1].Item1 > peakListOther
                [peakListOther.Count - 1].Item1))
563                    || (peakListLast.Count == 0 || peakListOther.Count == 0 &&
                    CompareCurveMaximums(otherCurve));
564         }
565     }
566 }
567
568
```