Finding local extrema in noisy data using Quantile Regression

Anton Antonov
MathematicaForPrediction at GitHub
MathematicaForPrediction at WordPress
September 2015

Introduction

This article describes an algorithm for finding local extrema in noisy data using Quantile Regression. The problem formulation and a solution for it using polynomial model fitting (through LinearModelFit) were taken from *Mathematica* StackExchange -- see “Finding Local Minima / Maxima in Noisy Data”, [1].

The proposed Quantile Regression algorithm is a version of the polynomial fitting solution (proposed by Leonid Shifrin in [1]) and has the following advantages: (i) requires less parameter tweaking, and (ii) more importantly it is more robust with very noisy and oscillating data. That robustness is achieved by using two regression fitted curves: one close to the local minima and another close to the local maxima, computed for low and high quantiles respectively. (Quantile Regression is uniquely able to do that.)

The code for this article is available at [6].

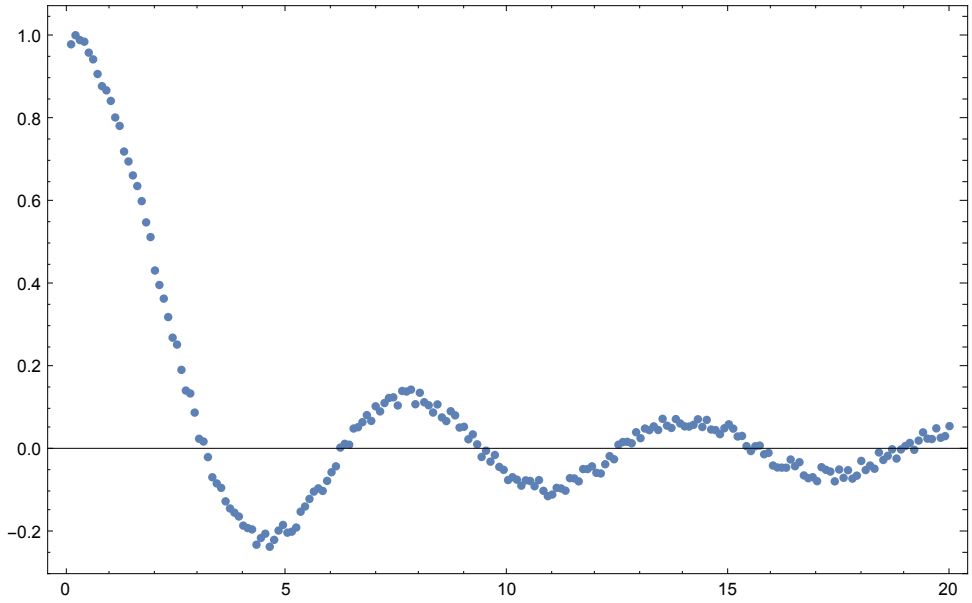
The problem

Here is the problem formulation from [1].

Problem 1: We have a list of pairs of numbers representing measurements of a scalar variable on a regular time grid. The measurements have noise in them.

As a data example for this problem the author of the question in [1] has provided the following code:

```
temptimelist = Range[200] / 10;
tempvaluelist = Sinc[#] &@temptimelist + RandomReal[{-1, 1}, 200] * 0.02;
data1 = Transpose[{temptimelist, tempvaluelist}];
ListPlot[data1, PlotRange → All, Frame → True, ImageSize → 500]
```

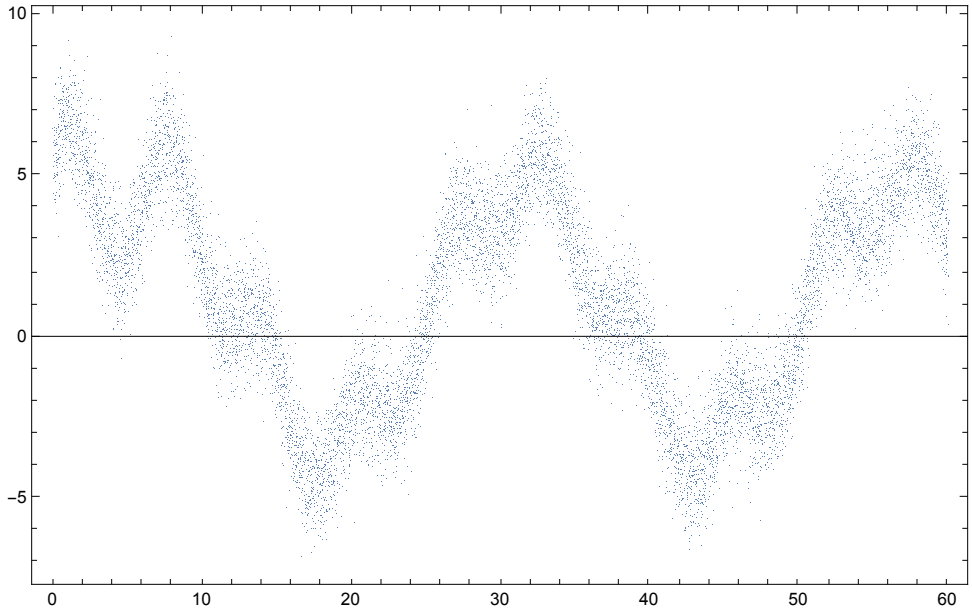


In this article we are going to consider a more general problem hinted in the discussion of the solutions in [1].

Problem 2: Assume that the data in Problem 1 is collected several times and the noise is present in both the measured values and the time of measurement. Also the data can be highly oscillatory in nature.

Consider this data generation as an example for Problem 2:

```
n = 1000;
xs = N@Rescale[Range[n], {1, n}, {0, 60}];
data2 = Flatten[Table[Transpose[{xs + 0.1 RandomReal[{-1, 1}, Length[xs]],
       Map[5 Sinc[#] + Sin[#] + 4 Sin[1 / 4 #] &, xs] + 1.2
         RandomVariate[SkewNormalDistribution[0, 1, 0.9], Length[xs]]}], {10}], 1];
ListPlot[data2, PlotRange → All, Frame → True, ImageSize → 500]
```



Note that this data has 10 000 points and it is much larger than the data for Problem 1.

Extrema location approximation by model fitting followed by nearest neighbors search

Several solutions are given in [1]. A couple are using wavelets or Gaussian filtering for de-noising. The one we are going to focus on in this article is using polynomial fitting for extrema location approximation and then finding the actual data extrema by Nearest Neighbors (NN's) search. It is provided by Leonid Shifrin.

Let us list the steps of that algorithm:

1. Fit a polynomial through the data (using `LinearModelFit`).
2. Find the local extrema of the fitted polynomial. (We will call them ***fit estimated extrema***.)
3. Around each of the fit estimated extrema find the most extreme point in the data by nearest neighbors search (by using `Nearest`).

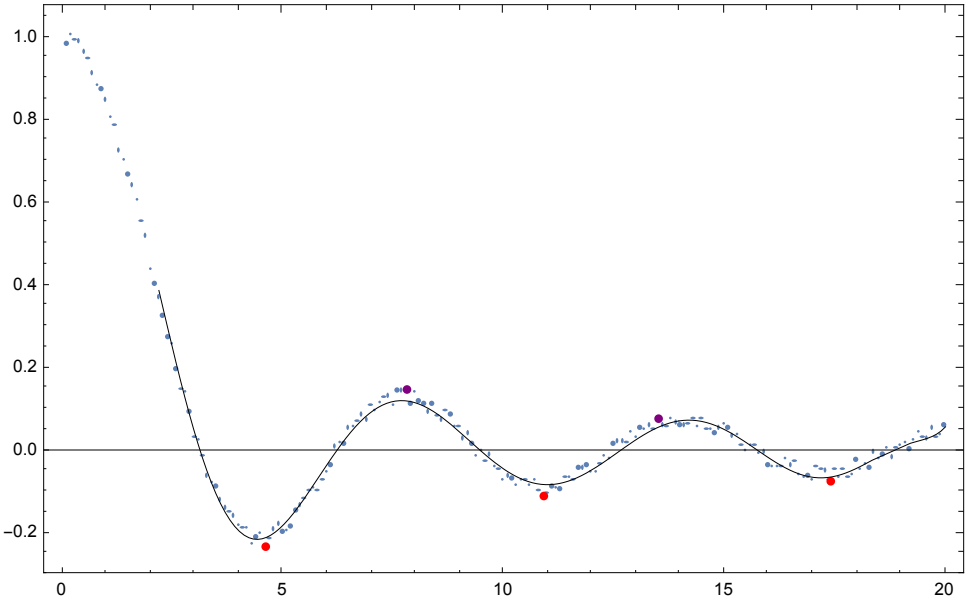
Here is a slightly modified implementation in *Mathematica* (I have added the use of Chebyshev polynomials):

```
Clear[LMFFindExtrema]
LMFFindExtrema[points_List, fitOrder_Integer,
    around_Integer: 5, fitFunctionType_: "Polynomial"] :=
  Module[{fit, fn, extrema, x, signs, extPoints},
   (* Step 1 *)
   Which[
    fitFunctionType == "ChebyshevT",
    fit = LinearModelFit[points, ChebyshevT[#, x] & /@ Range[0, fitOrder], x],
    True, (*fitFunctionType=="Polynomial",*)
    fit = LinearModelFit[points, x^Range[0, fitOrder], x]
   ];
   fn = fit["Function"];
   (* Step 2 *)
   extrema = NSolve[fn'[x] == 0, x, Reals];
   signs = Sign[fn''[x] /. extrema];
   (* Step 3 *)
   extPoints = MapThread[First@SortBy[Nearest[points, #, around], #2] &,
     {{x, fn[x]} /. extrema, signs /. {1 → Last, -1 → (-Last[#] &)}}];
   {fn, Map[Pick[extPoints, signs, #] &, {1, -1}]}
  ];
```

We are going to refer to this algorithm as ***LMFFindExtrema***.

Here is the application of the algorithm to the data example of Problem 1:

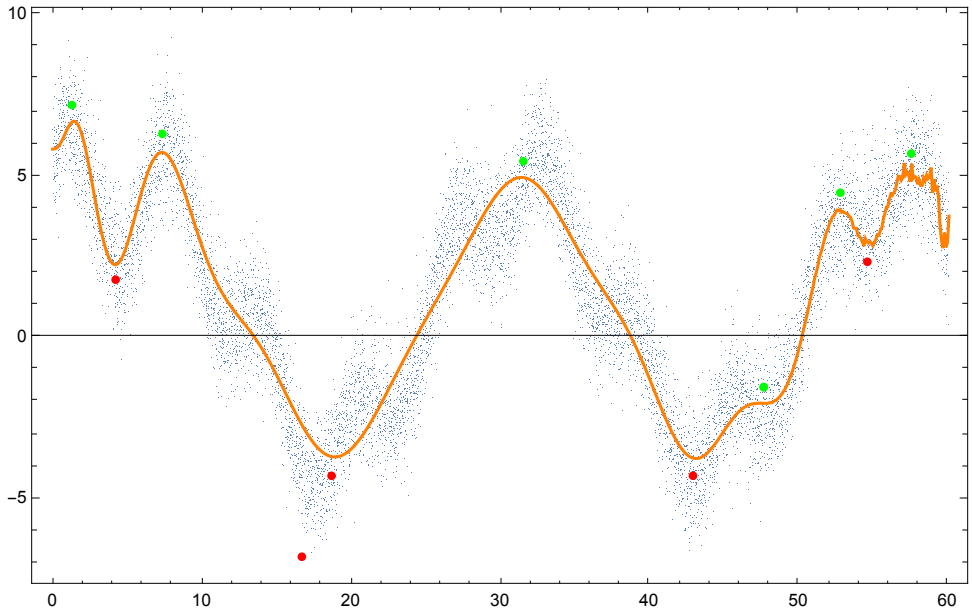
```
{fit1, extrema1} = LMFFindExtrema[data1, 10, 20, "ChebyshevT"];
```



(The continuous line shows the fitted curve.)

Here is the application of the algorithm to the data example of Problem 2:

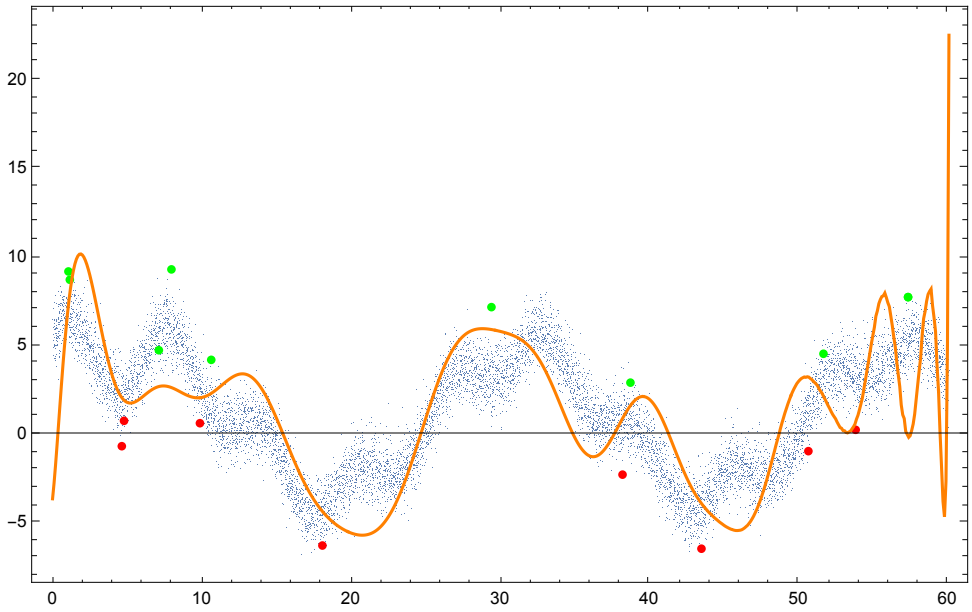
```
{fit2, extrema2} = LMFFindExtrema[data2, 30, 60, "ChebyshevT"];
```



We can see from the plot that there are two reasons for the estimated local extrema to be displaced: (1) the curve fitted by the model does not follow the data well, (2) too few points are examined by the NN' search around the fit estimated extrema.

It does not help to just increase the number of basis polynomials and the number of NN's examined points:

```
{fit2, extrema2} = LMFFindExtrema[data2, 60, 200, "ChebyshevT"];
```



We can also see that increasing the number of examined NN's for each of the fit estimated extrema would make some the final result points to be "borrowed" from neighboring peaks in the data.

Experiments with fitting trigonometric basis functions showed very good fit to the data but the calculations of the fit estimated extrema were very slow.

Using Quantile Regression

It is trivial to rewrite the algorithm LFMFindExtrema to use Quantile Regression instead of linear model fitting of polynomials. We are going to use the *Mathematica* package [2] implementation described in [3,4]. The function QuantileRegression provided by [2] uses a B-spline basis [5] to find the quantile regression curves (also known as ***regression quantiles***).

```
Import[
 "https://raw.githubusercontent.com/antononcube/MathematicaForPrediction/master/
   QuantileRegression.m"]
```

```
QRFindExtrema[points_List, nknots_Integer,
   nfitOrder_Integer, around_Integer, quantiles_: {0.5}] :=
  Module[{fit, fn, extrema1, extrema2, extrema, x, signs1, signs2, signs, extPoints},
   (* Step 1 *)
   fn = Simplify[QuantileRegression[points, nknots, quantiles, InterpolationOrder →
        nfitOrder, Method → {LinearProgramming, Method → "CLP"}]];
   (* Step 2 *)
   extrema1 = Reduce[fn[[1]]'[x] == 0, x, Reals];
   extrema1 = Cases[{ToRules[extrema1]}, _Rule, ∞];
   signs1 = Sign[fn[[1]]''[#] & /@ extrema1[[All, 2]]];
   extrema2 = Reduce[fn[[-1]]'[x] == 0, x, Reals];
   extrema2 = Cases[{ToRules[extrema2]}, _Rule, ∞];
   signs2 = Sign[fn[[-1]]''[#] & /@ extrema2[[All, 2]]];
   (* Step 3 *)
   extrema =
    Join[
     Map[{#, fn[[1]][#]} &, Pick[extrema1[[All, 2]], # > 0 & /@ signs1]],
     Map[{#, fn[[-1]][#]} &, Pick[extrema2[[All, 2]], # < 0 & /@ signs2]]];
   signs = Join[Pick[signs1, # > 0 & /@ signs1], Pick[signs2, # < 0 & /@ signs2]];
   extPoints = MapThread[First@SortBy[Nearest[points, #, around], #2] &,
     {extrema, signs /. {1 → Last, -1 → (-Last[#] &)}}];
   {fn, Map[Pick[extPoints, signs, #] &, {1, -1}]}
  ];
```

We are going to call this algorithm ***QRFindExtrema***.

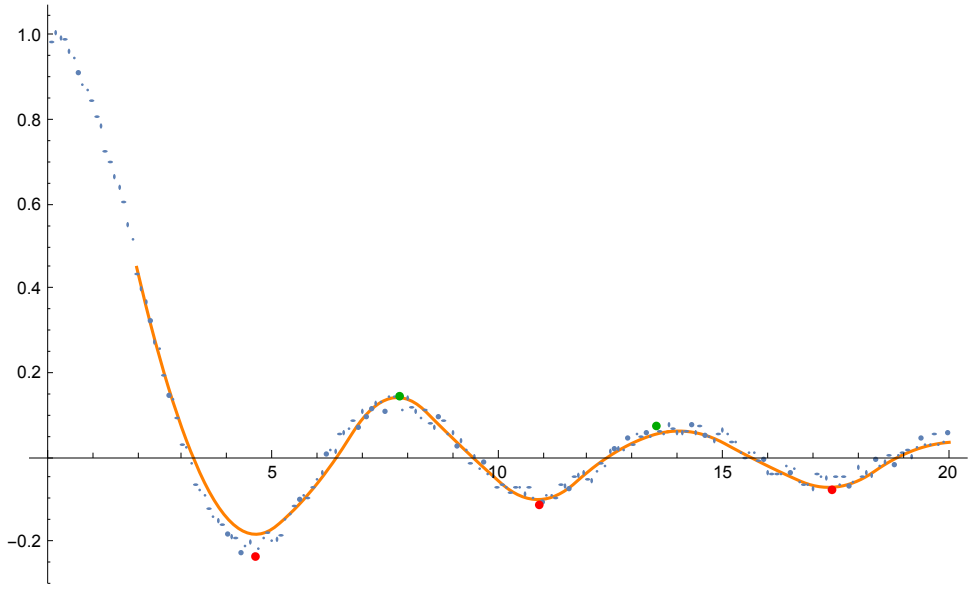
The algorithm QRFindExtrema has the following parameters: the data, number of B-spline knots, interpolation order, quantiles (corresponding to the curves to be fitted).

QRFindExtrema returns a list of regression quantile functions and a list of lists with extrema estimates.

The implementation of QRFindExtrema uses Reduce instead of NSolve because Reduce deals better with the Piecewise functions found by QuantileRegression. More importantly though, QRFindExtrema can use two curves for finding the local extrema: one for local minima, and one for local maxima. (This feature is justified below.)

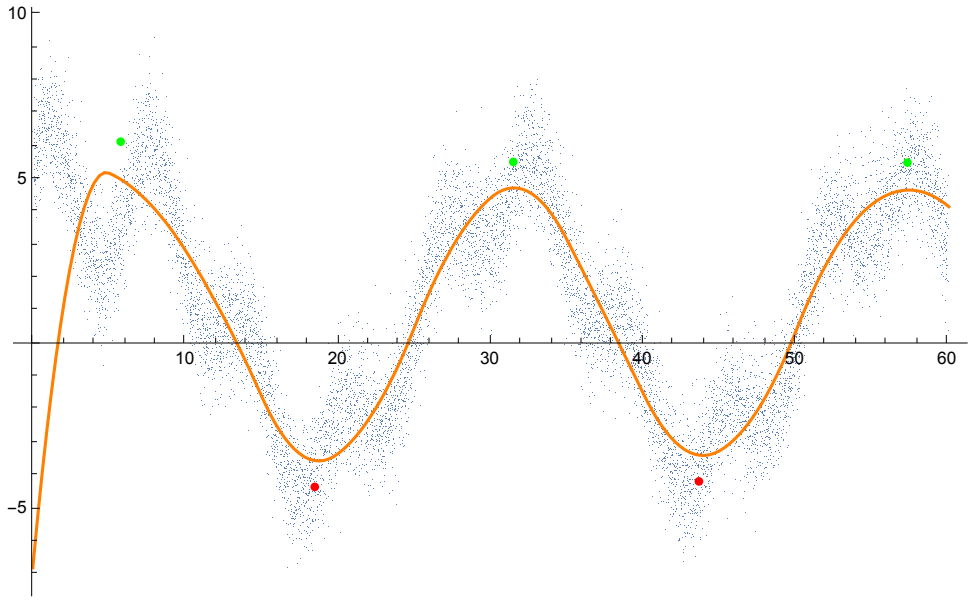
Here is the application of QRFindExtrema to the example data of Problem 1:

```
{qfuncs1, extrema1} = QRFindExtrema[data1, 12, 2, 10, {0.5}];
```



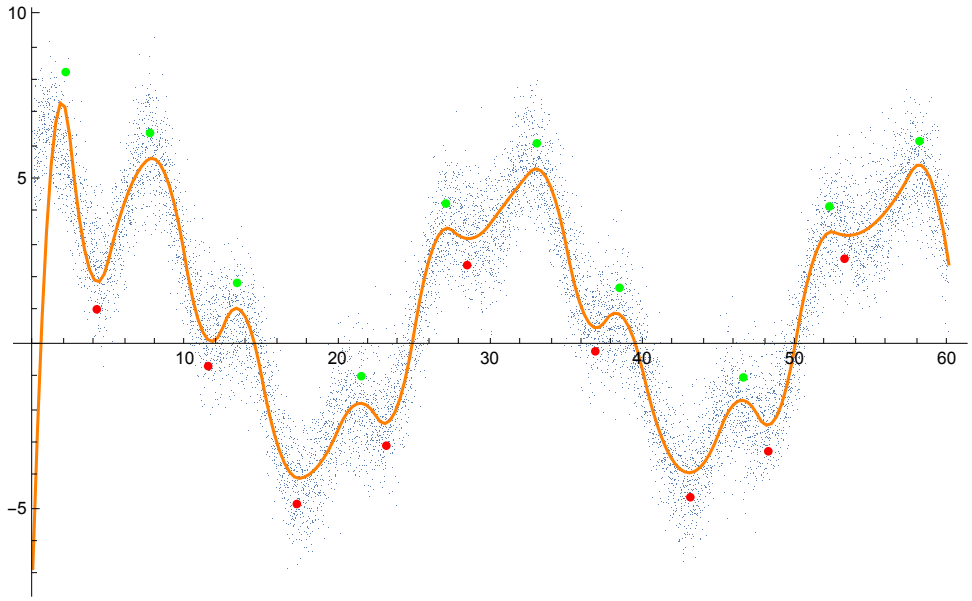
Here is application of QRFindExtrema to the data of Problem 2:

```
{qfuncs2, extrema2} = QRFindExtrema[data2, 12, 2, 120, {0.5}];
```



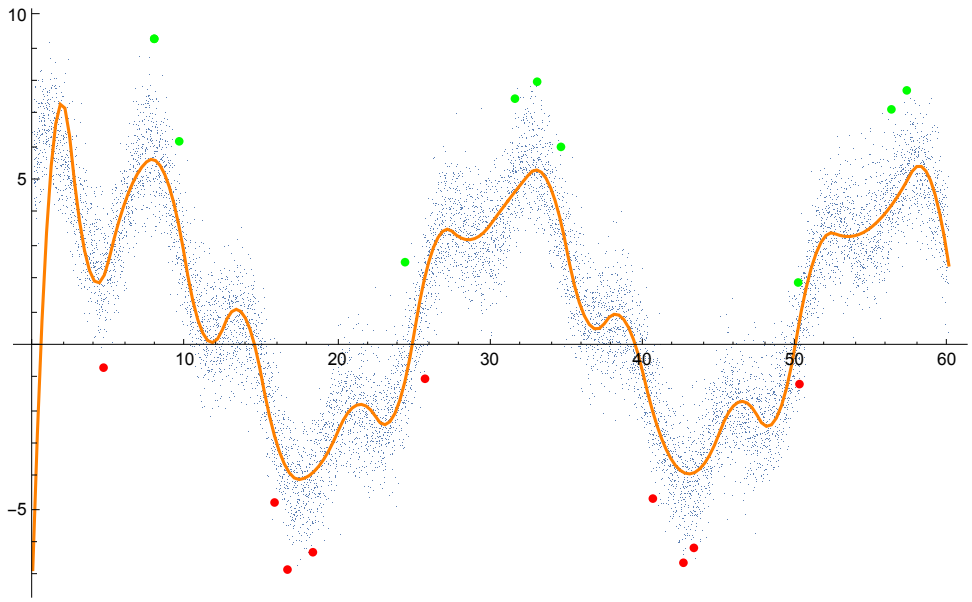
We can see that the regression quantile for 0.5 is too flat to get good judgment of the local extrema. We can get better results if we increase the number of knots for the B-spline basis built by `QuantileRegression`.

```
{qfuncs2, extrema2} = QRFindExtrema[data2, 24, 2, 120, {0.5}];
```



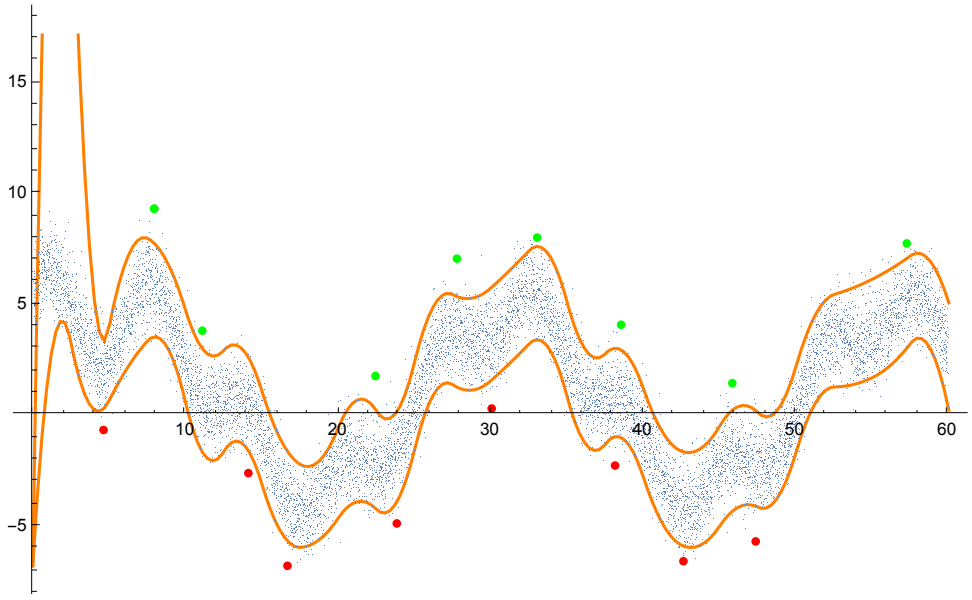
We can see that results look “almost right”, the horizontal locations of the peaks are apparently more-or-less correctly identified, but the result extrema are too close to the fitted curve. Just increasing the number of examined NN’s for the fit estimated extrema does not produce good results because points from neighboring peaks are being chosen as final extrema estimates.

```
{qfuncs2, extrema2} = QRFindExtrema[data2, 24, 2, 1500, {0.5}];
```



In order to solve this problem we use two regression quantiles. For local minima we use a regression quantile for a low quantile number, say, 0.02; for the local maxima we use a regression quantile for a large quantile number, say, 0.98 .

```
{qfuncs2, extrema2} = QRFindExtrema[data2, 24, 2, 200, {0.02, 0.98}];
```



The use of two (or more) curves to be fitted is an unique capability of Quantile Regression. With this algorithm feature by construction the lower regression quantile is close to the local minima and the higher regression quantile is close to the local maxima.

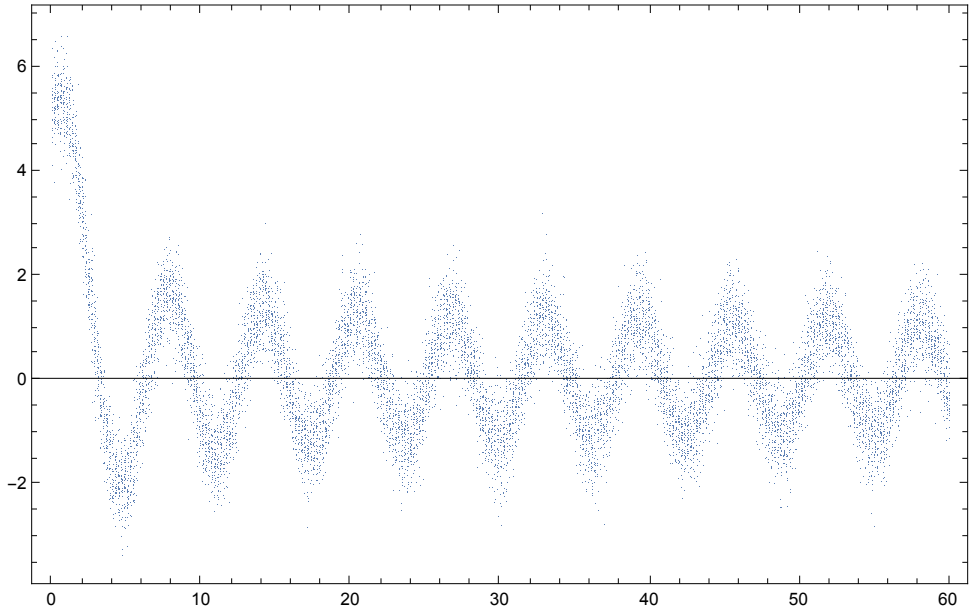
Also, since we find two regression quantile curves we can use two nearest neighbors finding functions: one with the points below the low regression quantile, and one with the points above the high regression quantile. The implementation in [6] takes an option specification for should the nearest neighbor functions for finding the extrema be constructed using all data points or just the outliers (the points outside of the found regression quantiles).

Using more timid data

With more timid data for Problem 2 the algorithm QRFindExtrema gives again (much) better results.

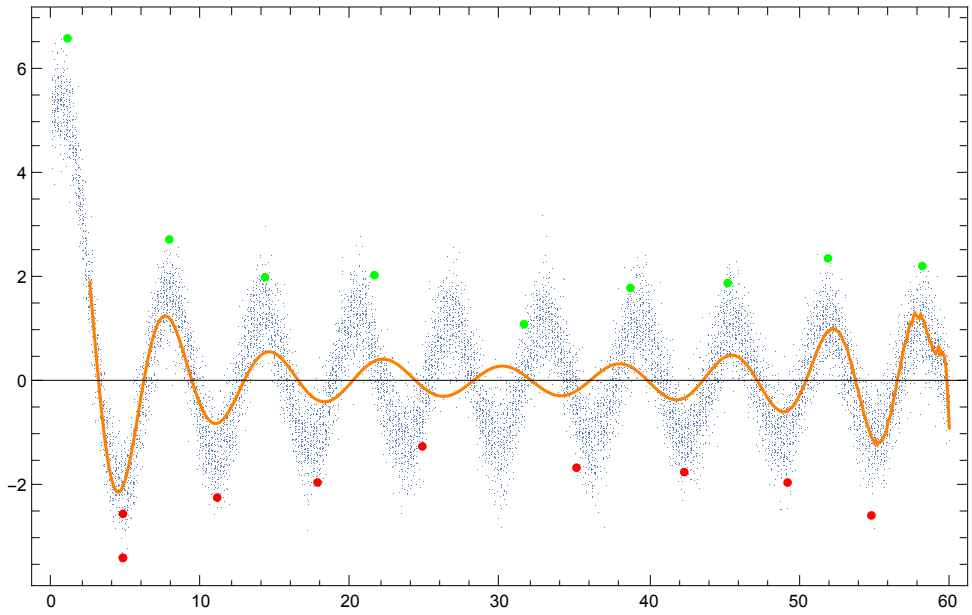
Consider this example data:

```
n = 600;
xs = N@Rescale[Range[n], {1, n}, {0, n / 10}];
data3 =
  Flatten[Table[Transpose[{xs, Map[5 Sinc[#] + Sin[#] &, xs] + 0.5 RandomVariate[
        SkewNormalDistribution[0, 1, 0.1], Length[xs]]}], {20}], 1];
ListPlot[data3, Frame → True, ImageSize → 500]
```



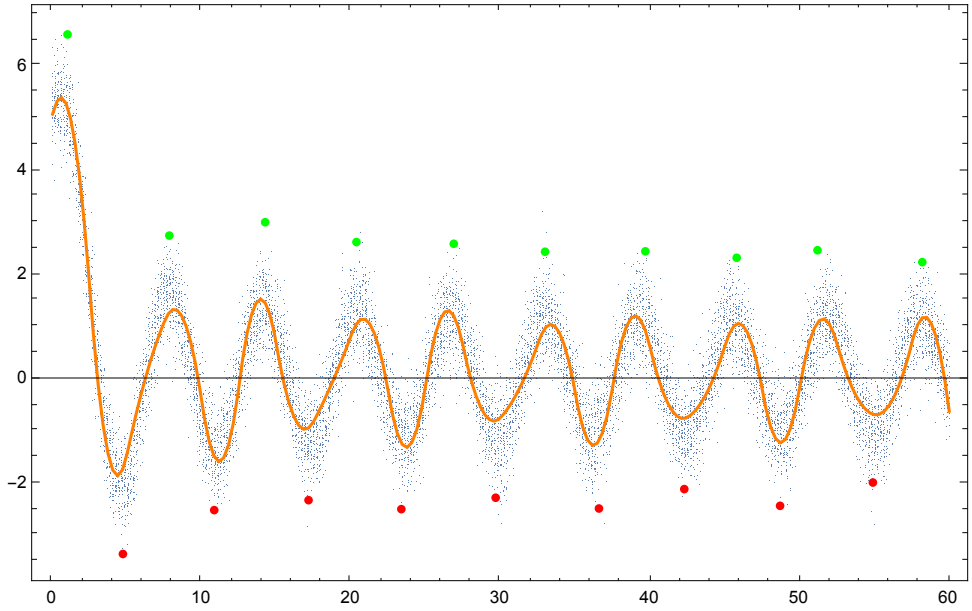
Here is a plot with the results of LMFFindExtrema:

```
{fit3, extrema3} = LMFFindExtrema[data3, 26, 500, "ChebyshevT"];
```



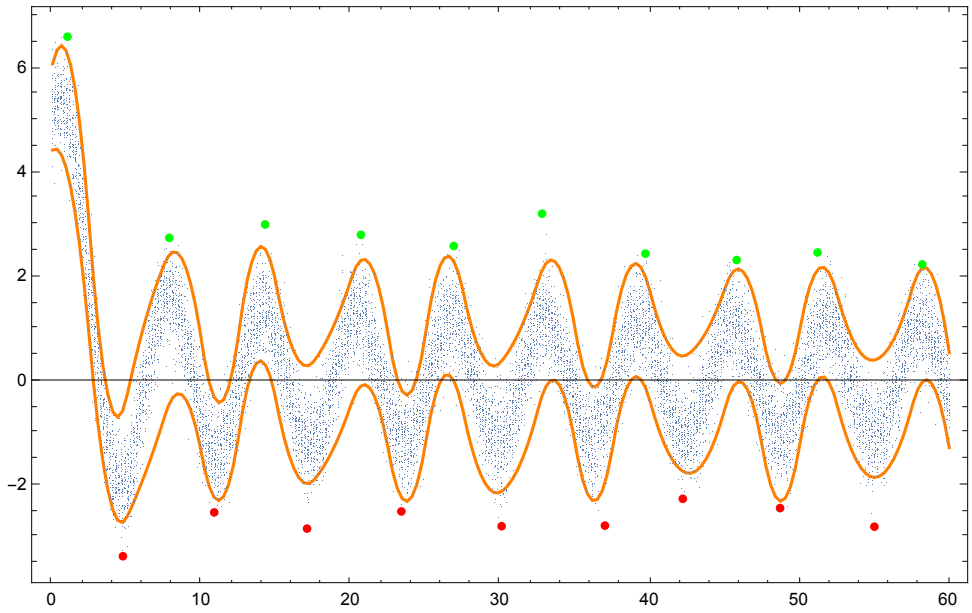
Here is a plot with the results of QRFindExtrema using only one regression quantile for 0.5:

```
{qfuncs3, extrema3} = QRFindExtrema[data3, 24, 2, 500, {0.5}];
```



Here is a plot with the results of QRFindExtrema using regression quantiles for 0.02 and 0.98:

```
{qfuncs3, extrema3} = QRFindExtrema[data3, 24, 2, 200, {0.02, 0.98}];
```



Again, it would be required less experimentation with the parameters values in order to get good results with Quantile Regression that fits two, low and high, regression quantiles.

Again, if we use Quantile Regression that fits two, low and high, regression quantiles, then we will get good results with less experimentation.

References

[1] *Mathematica* StackExchange discussion. “Finding Local Minima / Maxima in Noisy Data”, URL: http://mathematica.stackexchange.com/questions/23828/finding-local-minima-maxima-in-noisy-

data/ .

[2] Anton Antonov, Quantile regression *Mathematica* package, source code at GitHub, https://github.-com/antononcube/MathematicaForPrediction, package QuantileRegression.m, (2013).

[3] Anton Antonov, Quantile regression through linear programming, usage guide at GitHub, http-s://github.com/antononcube/MathematicaForPrediction, in the directory "Documentation", (2013).

[4] Anton Antonov, Quantile regression through linear programming, “Mathematica for prediction algo-rithms” blog at WordPress.com, 12/16/2013.

[5] Anton Antonov, Quantile regression with B-splines, “Mathematica for prediction algorithms” blog at WordPress.com, 1/1/2014.

[6] Anton Antonov, QuantileRegressionForLocalExtrema *Mathematica* package, source code at GitHub, https://github.com/antononcube/MathematicaForPrediction, package QuantileRegressionForLocalEx-trema.m, (2015).
The package is in the directory “Applications”.