MARS/ARES Modeling of Race Line Data

http://github.com/charlestucker3/ARES-race-line-data

Charles L. Tucker III
Department of Mechanical Science and Engineering
University of Illinois at Urbana-Champaign
1206 W. Green St.
Urbana, IL 61801
ctucker@illinois.edu

July 5, 2022

Abstract

The MARS (Multivariate Adaptive Regression Splines) algorithm can be used to model the path of a race car around a circuit, using information from typical data acquisition systems. Specifically, a model for curvature as a function of path length is fitted to logged data for speed and lateral acceleration. This type of information is useful as input to path-based lap time simulators. An example in MATLAB, using the `ARESLab` package, is shown.

1 Introduction

In a recent post on the *Vehicle Dynamics Professionals* group on Facebook, Andrea Quintarelli asked about methods to extract curvature vs. path length information from typical logged data for race cars, in a form suitable for use in a quasi-static lap time simulation. From his post, the main issue is how to smooth the data:

> No matter what the source data looks like (deriving curvature from speed and Ay, from GPS, etc.), the data always includes some noise that needs somehow to be filtered out to get a proper, smooth trace of distance vs cornering radius (or curvature) . . .
>
> Working in Matlab, I tried to both smooth the data and filter them, but in both cases the results were not that good: the radius vs distance trace either does not follow the original one (produced using logged data) at all, or is not smooth enough.

The problem is illustrated in Fig. 1. This data for curvature vs. distance is modeled using 40 equally-spaced cubic spline segments[1]. The model is not bad in some locations, but the data is over-modeled where the track is relatively straight (e.g., from 0 to 200 m and from 1100 to 1500 m) and badly under-modeled in other locations. The fit is particularly poor between 2400 and 2800 m, and a feature around 3200 m is missed as well. One could improve the fit in the under-modeled regions by using a larger number of spline segments, but that would make the over-modeling problems worse in other locations.

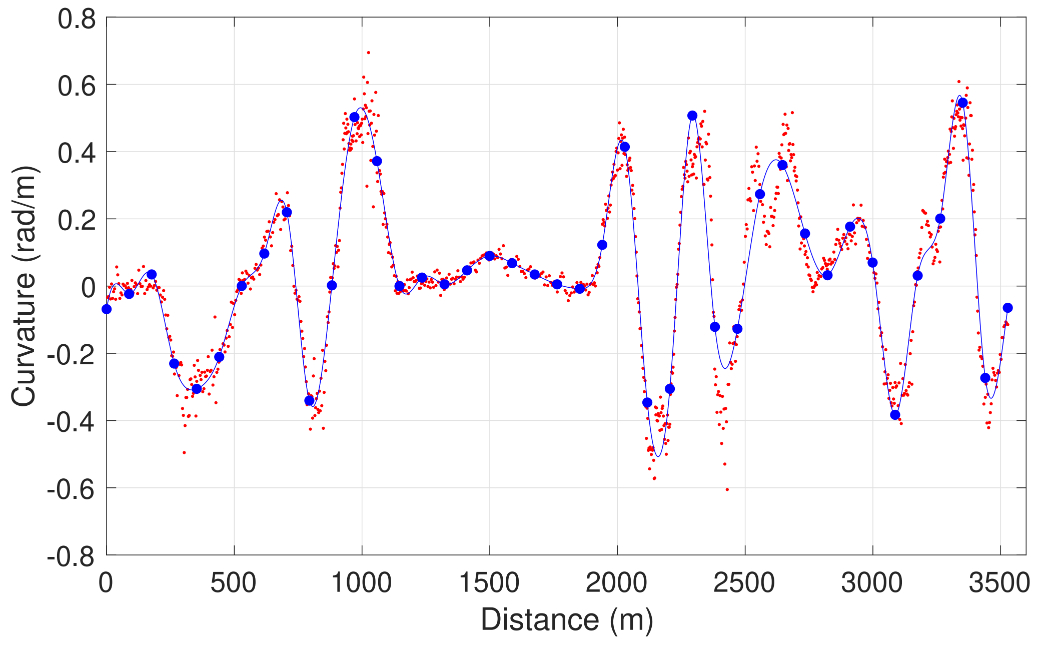


Figure 1: One lap of curvature data for Mid-Ohio (red points), modeled using 40 equally-spaced cubic spline segments. Some portions of the data are under-modeled, while other portions are over-modeled.

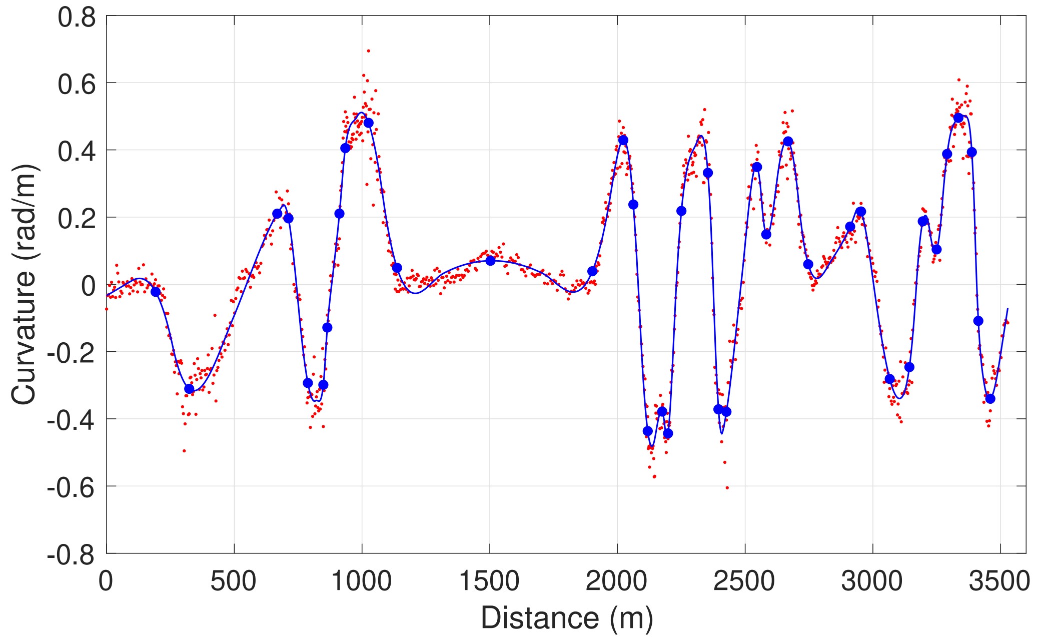


Figure 2: Curvature data for Mid-Ohio and a MARS-type cubic spline model built using `ARESLab`. Blue dots are knots. `maxFuncs = 100`, `c = 6`, `cubic = true`, and other parameters have default values.

The challenge here is to smooth the noise that is typical of logged data but not remove features that indicate the path the driver actually took.

This note demonstrates one approach to solving this problem, using the MARS algorithm. MARS stands for Multivariate Adaptive Regression Splines. In the simplest sense, MARS performs a piecewise fit similar to Fig. 1, but automatically finds optimal locations for the breakpoints between segments. The result is a model like Fig. 2.

We first provide a brief explanation of the MARS algorithm, then show how Fig. 2 is created using MATLAB and the `ARESLab` package (Jekabsons, 2016). The note closes with a discussion of limitations and future possibilities.

2 The MARS Algorithm

To understand MARS, suppose that we want to fit a piecewise linear model to the data points shown in Fig. 3(a). The MARS algorithm develops this model in two passes:

The forward pass starts with a flat-line approximation of the data (the average value, as in Fig. 3(a)), and progressively adds locations where the model can change slope. These points are called *knots*. Initially, each knot has two *basis functions*, one that affects the model slope to the right of the knot, and one that affects the slope to the left.

Each new knot is placed at the location that gives maximum reduction in the mean square error between the model and the data, and the slopes of the basis functions are adjusted to minimize the mean square error. Figure 3(b) shows the model with the first knot added. This is the best two-segment linear model for this data.

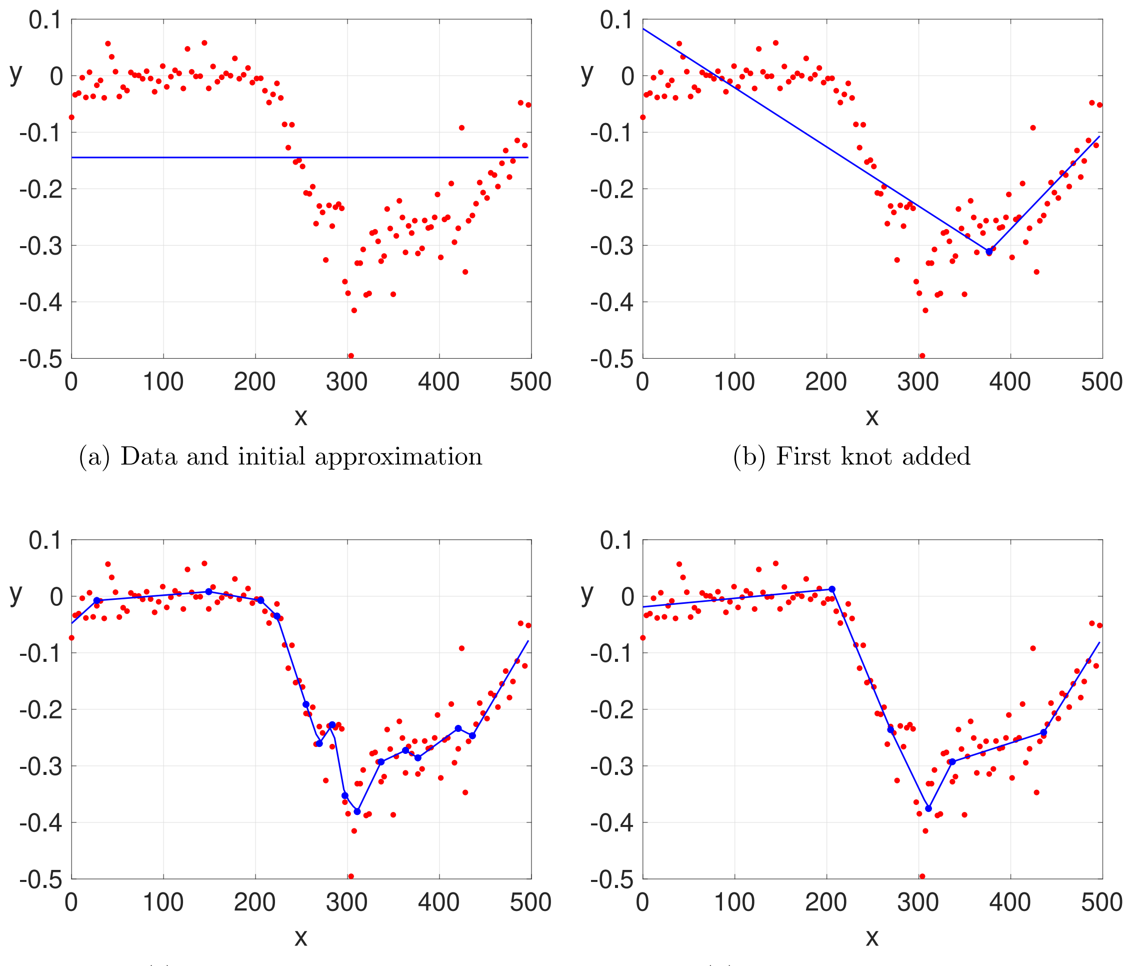
More knots and basis functions are added, each one providing the maximum reduction in mean square error given the previous knot locations. This continues until either a pre-set maximum number of basis functions is reached or the mean square error becomes very small.

Figure 3(c) shows the model at the end of the forward pass. Typical of this stage, the data is over-fitted, with the model following small wiggles in the data that are probably not meaningful.

The backward pass reduces this over-fitting by removing (pruning) basis functions and knots that contribute little to the data fit. This pruning is also done step by step, removing the least important basis function first, refitting the model, then removing the next least important basis function, and so on.

A statistic called the *generalized cross validation* or GCV is used to decide which basis function to remove next. The GCV is proportional to the residual sum of squares of the model, but it also depends on in the current number of basis functions. GCV is a "bang for the buck" metric: a model with many basis functions can follow the wiggles in the data and have a small mean square error, but if another model with fewer basis functions has only a slightly larger mean square error, it can have a smaller GCV. The final result of the backward pass is the pruned model with the lowest GCV score. This model offers the best combination of a good fit to the data and a small number of basis functions.

[1]The model in Fig. 1 was created using the `ppfit` package by Karl Skretting for fitting piecewise polynomials, `https://www.mathworks.com/matlabcentral/fileexchange/55024-ppfit-varargin`.



(a) Data and initial approximation

(b) First knot added

(c) End of forward pass

(d) End of backward pass

Figure 3: Example of piecewise linear MARS-type fit to noisy data. Red dots are data, blue lines are the model, and blue circles are the knot locations.

> Figure 3(d) shows our example at the end of the backward pass. The number of knots has been reduced, but the model still fits the data well. The final knot locations have been are strategically placed to model the data accurately and efficiently.

MARS was initially developed by Friedman (1991). The article in Wikipedia (2022) is a good place to start reading more about the method. The MARS name for software is a trademark of Salford Systems, so other software packages that use the method have different names. The examples in this note use `ARESLab`, which is a toolbox for MATLAB. Information about implementations in R and Python, and a list of additional papers on the MARS algorithm, can be found in the `ARESLab` manual (Jekabsons, 2016).

While this example used a piecewise linear model, `ARESLab` can also produce cubic spline models. Also, while all of our examples here have a single independent variable and a single dependent variable, MARS-type models can have dozens or even hundreds of independent variables, as well as multiple dependent variables.

3 Example

For a realistic example, we model data from the author's 2016 Porsche GT4 street car on the Mid-Ohio Sports Car Course[2]. The data file and a MATLAB script to produce Fig. 2 can be downloaded from https://github.com/charlestucker3/ARES-race-line-data. We neglect slope and camber of the track for this analysis.

The data was acquired using an APEX Pro device, and probably has more noise than professional-grade data loggers. The data set includes GPS speed v and lateral acceleration a_y, logged approximately every 0.1 s. The original data file indicates the lap number for each data point, and one lap of data was extracted using that information and saved in the file `MidOhioLap3.csv`. This may not correspond precisely to a full lap, but the data still provides a useful illustration of the method.

We start by reading this data. Here is the MATLAB code:

```
% Read sample data.  Time in seconds, distance in meters,
% accelerations in gravities, heading is in degrees.
% gz is the yaw rate in degrees/s.
% Only time, ay and speed will be used here.
lap3 = readtable('MidOhioLap3.csv');
```

Distance traveled s is computed using the average velocity over each time step,

$$s(i) = s(i-1) + \frac{1}{2}\Big(v(i-1) + v(i)\Big)\Big(t(i) - t(i-1)\Big) \tag{1}$$

with $s(1) = 0$. Path curvature κ is then estimated using

$$\kappa(i) = \frac{a_y(i)}{v^2(i)} \tag{2}$$

Recall that $\kappa = 1/R$, where R is the local radius of curvature. For this data set, right-hand turns have positive curvature and left-hand turns have negative curvature.

```
% Curvature based on ay
kappa = lap3.ay * 9.82 ./ lap3.speed;

% Path length s calculated from speed and time
s = zeros(size(lap3.time));
s(2:end) = (lap3.time(2:end) - lap3.time(1:end-1)) ...
        .* (lap3.speed(1:end-1) + lap3.speed(2:end))/2;
s = cumsum(s);
```

To fit a MARS-type model, the `ARESLab` package must be downloaded from http://www.cs.rtu.lv/jekabsons/regression.html and installed in a suitable location. This toolbox has many functions and options, but only a few of those are important for our current problem.

The first step uses the function `aresparams2` to create a data structure `params` that will control the model-building step. The key parameters and their suggested values are discussed below. This data structure is passed to `aresbuild`, along with the distance and curvature data, to build the ARES model.

[2]This course for this event included the chicane between Turn 1 and the Keyhole.

```
% Set parameters for ARES model building.
params = aresparams2('maxFuncs', 100, 'c', 6, 'cubic', true);

% Build the ARES model
[model, time, resultsEval] = aresbuild(dist, kappa, params);
```

Building the model may take several seconds. The complete, fitted model is stored in the variable `model`. The `time` variable contains the execution time, and `resultsEval` contains information about the model and the fitting process. We will only use `model` here.

Once the `model` has been built, values of the curvature at any distance from the start can be produced as often as desired (and very quickly) using `arespredict`. For the plot in Fig. 2, we use

```
% Generate a set of points for plotting the model
splot = linspace(min(dist), max(dist), 1000)';   % Must be a column vector
kplot = arespredict(model, splot);
```

The results are then plotted in the usual Matlab way:

```
% Plot the data, the fitted model, and the knot points
figure(1); clf; hold on
plot(dist, kappa, 'r.')  % Raw data points
plot(splot, kplot, 'b-', 'LineWidth', 1)  % Model
knots = sort(cell2mat(model.knotsites));  % Knot points
plot(knots, arespredict(model, knots), 'bo', 'MarkerFaceColor', 'b')

set(gca, 'FontSize', 18)
xlabel('Distance (m)')
ylabel('Curvature (rad/m)')
grid on
box on
axis([0 3600 -0.8 0.8])
```

The data and the `ARESLab` model are shown in Fig. 2. The final model has 39 basis functions: one at each of the 38 knots[3], plus the mean value.

Notice how the algorithm has chosen few knots where the track is relatively straight, e.g., between 1200 and 1800 m, but spaced the knots closely where the curvature changes rapidly. The dip-and-rise in curvature near 3200 m appears to be real driver input, and the algorithm has modeled that area nicely. The wiggle in curvature at the apex near 2200 m initially looks like over-modeling, though that turn has a sharp crest and the model may have captured the driver easing the steering as the car goes over the crest.

4 ARES Parameter Selection

In the `ARESLab` package there are many parameters that can be adjusted to control the model-building process. Many of these are used to tune models with multiple independent variables, or

[3]Each knot starts with two basis functions in the forward pass, but most knots have at least one of those basis functions pruned in the backward pass.

to speed up the computation of those models. Our example, with a single output depending on a single input, is easy as ARES problems go, and can be calculated using just a few parameters.

The important parameters and their suggested values are as follows:

`maxFuncs` sets the maximum number of basis functions for the forward pass. This is a key parameter that must be chosen carefully.

Figure 4 shows a model where the value of `maxFuncs` is too small for the data (`maxFuncs = 40`). The algorithm has not been able to place enough knots to model all the important features in the data on the forward pass, and the corners between 2500 and 3000 m are missed entirely. The cure for this problem is to increase the value of `maxFuncs`.

If the value of `maxFuncs` is set too large, the forward pass will greatly over-model the data and the backward pass may not remove all the unnecessary knots. An example is shown in Fig. 5 using `maxFuncs = 200`. Now the data appears to be over-modeled around 300 m and 1000 m. To fix this problem, either reduce `maxFuncs` or increase `c` (see below).

The fit in Fig. 2 used `maxFuncs = 100`. Based on this, I suggest setting `maxFuncs` to 6 or 7 times the number of turns on the circuit on the first try, and adjusting from there.

`c` determines how much the GCV is affected by the number of basis functions. This parameter can be used to fine-tune the amount of pruning in the backward pass. The default value of 2 for this example is not bad, and the usual recommendation is to set `c` between 2 and 4. Larger values will make the pruning process more aggressive, and can help to remove a few knots that seem to be over-modeling the data. A value of `c = 6` was used in Fig. 2 to do just this. A value like `c = 6` also tends to reduce over-modeling if `maxFuncs` has been set a bit large.

`cubic` can be set to `true` to generate a cubic spline model, as in Fig. 2, or `false` to create a piecewise linear model as in Fig. 3.

Additional parameters can be specified; see the `ARESLab` manual (Jekabsons, 2016) for details. Any parameters that are omitted from the call to `aresparams2` (or whose values are specified as empty using `[ ]`) receive their default values.

5 Limitations and Future Possibilities

The example here used one particular approach for extracting curvature from logged data, but there are other ways to do this. The MARS/ARES approach works equally well whatever the source of data.

One limitation of the `ARESLab` package is that it does not support periodic models. This means that the curvature at the end of the lap will not match the curvature at the start of the lap. This may not be an issue for some applications.

Another limitation is that there is no guarantee that the fitted model for $\kappa(s)$ represents a closed path. For motion in a plane, the vehicle path $x(s)$, $y(s)$ and its heading $\psi(s)$ can be calculated from the fitted model by integrating the following system of different equations:

$$\frac{dx}{ds} = \cos\psi \qquad \frac{dx}{ds} = \sin\psi \qquad \frac{d\psi}{ds} = \kappa(s) \tag{3}$$

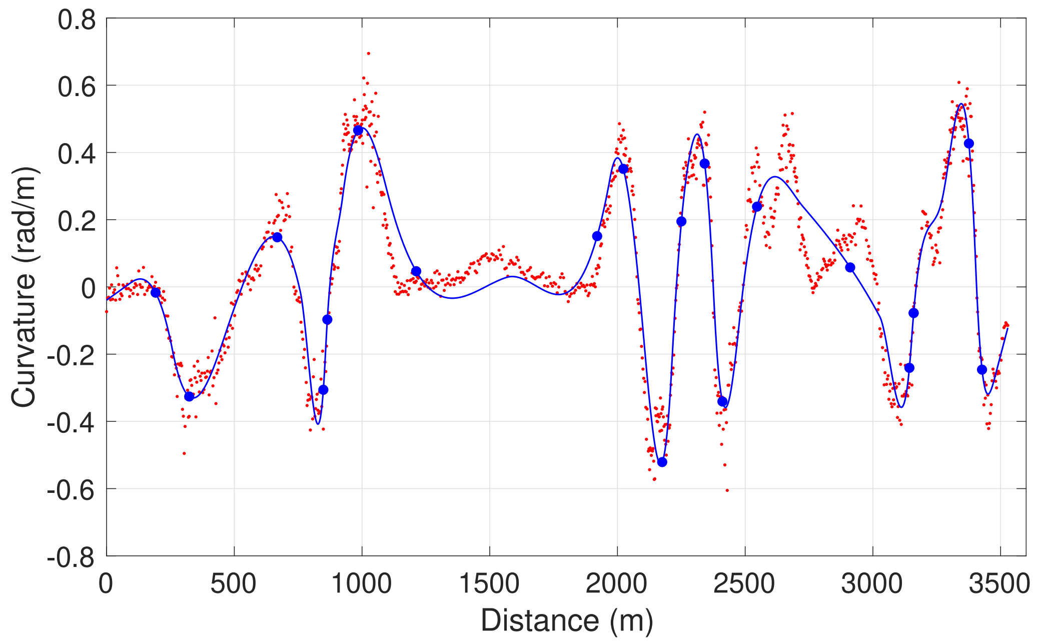


Figure 4: ARES model using `maxFuncs = 40`, which is too small for the data.

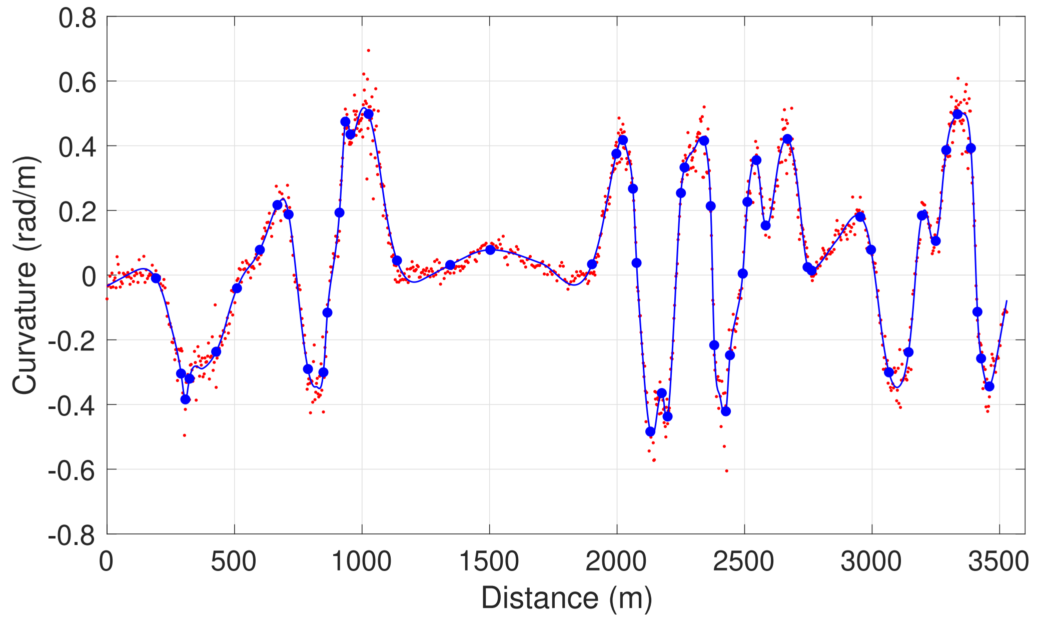


Figure 5: ARES model using `maxFuncs = 200` and `c = 2`. The large value of `maxFuncs` and small value of `c` over-models some locations.

with suitable initial conditions at the start of the lap. However, the position and heading at the end of the lap may not join to the start of the lap. In principle, one could enforce this as a constraint in the modeling process, but there is no way to do this in `ARESLab`.

Some users may prefer to be able to remove certain knots and their basis functions interactively. The `ARESLab` package does have a function `aresdel` that can delete a basis function. It should be possible to build a MATLAB GUI that uses this function and allows user-directed editing of the basis functions.

The MARS algorithm is a powerful tool for data modeling, and works well for the curvature data shown here. The `ARESLab` package is a useful implementation for MATLAB users. A few parameters need to be adjusted to fit each data set, especially `maxFuncs`, but this can be done with a modest amount of trial and error. `ARESLab` and other MARS-type software packages are also capable of modeling functions that depend on multiple input variables, so there may be many other uses for this tool in motorsports.

Comments and Suggestions

Comments and suggestions may be sent to the author at ctucker@illinois.edu.

References

J. H. Friedman. Multivariate adaptive regression splines. *The Annals of Statistics*, 19(1):1 – 67, 1991. doi: 10.1214/aos/1176347963. URL `https://doi.org/10.1214/aos/1176347963`.

G. Jekabsons. ARESLab: Adaptive Regression Splines toolbox for Matlab/Octave, 2016. URL `http://www.cs.rtu.lv/jekabsons/regression.html`.

Wikipedia. Multivariate adaptive regression spline, 2022. URL `https://en.wikipedia.org/wiki/Multivariate_adaptive_regression_spline`.