Regression analysis using a blending type spline construction

Tatiana Kravetc, Børre Bang, Rune Dalmo

R&D group Simulations
Department of Computer Science and Computational Engineering
Faculty of Science and Technology
UiT - The Arctic University of Norway

9th International Conference on Mathematical Methods for Curves and Surfaces,
Tønsberg, June 23rd - 28th, 2016

Abstract

Regression analysis allows us to track the dynamics of change in measured data and to investigate their properties. A sufficiently good model allows us to predict the behavior of dependent variables with higher accuracy, and to propose a more precise data generation hypothesis.
By using polynomial approximation for big data sets with complex dependencies we get *piecewise* smooth functions. One way to obtain a smooth spline representation of an entire data set is to use local curves and to blend them using smooth basis functions. This construction allows the computation of derivatives at any point on the spline. Properties such as tangent, velocity, acceleration, curvature and torsion can be computed, which gives us the opportunity to exploit these data in the subsequent analysis.
We can adjust the accuracy of the approximation on the different segments of the data set by choosing a suitable knot vector. This article describes a new method for determining the number and location of the knot-points, based on changes in the Frenet frame.
We present a method of implementation using generalized expo-rational B-splines (GERBS) for regression problems (in two and three variables) and we evaluate the accuracy of the model using comparison of the residuals.

1 Introduction

1.1 The problem

We describe a general method suitable for use on stream based data. A wide range of applications is possible, trend analysis for correlation of trends between weather stations, and estimation of trends in online revenue streams are two examples that will be considered for further work at a later time. In this paper, the model implementation is based on weather data. Stages in the data form the basis for a segmentation intended to be used in further classification, which gives us the opportunity to identify state changes, or derived properties as for instance accumulations (mass/volume etc.). Identifying shifts in trends, i.e. stages in time-dependent open ended stream based data sets is considered to be of general interest in a wide area of real world applications.

1.2 Contribution

A natural way of treating possibly noise contaminated data which consists of more or less well defined stages, is to utilize local approximation in the relevant stages.

Our method changes the representation of the raw data to a form that accommodates both local approximation and adjustable criteria for identifying stages. Local piecewise Bézier curves are formed from the Least Squares Method of a limited number of data points that are close in time. Blending splines makes it possible to keep the original approximation and gives a gradual refinement that can be used to balance accuracy and computational effort. The Hermite properties of the interpolation method we use, where the local curves supply the additional information regarding derivatives, makes the resulting approximation work on localized intervals without keeping the complete data set (in memory).

1.3 Related work

1.3.1 Statistical methods

In [15], James solves the registration problem between curves by "equating the moments of the curve while also shrinking toward a common shape". This could also be used to recognize "stages" or states, if information about the shapes of the stages are available. Statistical methods like the use of first and second moments of data for feature extraction, have been treated in [7]. For several application specific problems, an expansion in this direction of the approximation theory related to a blending type spline construction, would be appropriate.

1.3.2 Geometrical methods

Real-time approximation and smoothing of geometry data, like position and orientation under constraints, arises frequently in path planning and control theory for self-guided vehicles. In [6], Brezak and Petrovic consider locally smoothing of paths by clothoids, using table lookup to implement a numerical integration that otherwise could be prohibitive in computational cost.

The problem of finding suitable knot vectors has been studied in the literature [13, 5, 8]. It seems to be an active area of research, however, the methods are depending on the data. In [3], Bittner and Brachtendorf utilize the Oslo algorithm for knot insertion in a wavelet context and provides an adaptive approximation method.

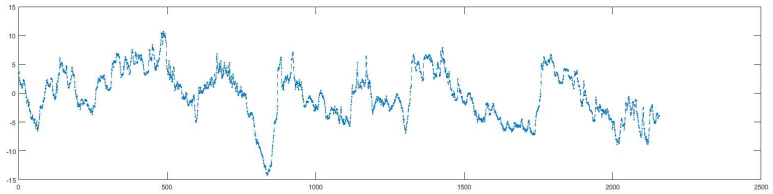
In [19] a locally supported spline quasi-interpolant is developed, with the assumption that the B-spline knots are chosen to lie midway between consecutive sampling points, or chosen to coincide with the sampling points. Then by blending of this and a local Hermite interpolation scheme, a method for fast interpolation is achieved, which has some resemblance to our method. Since our global setting is open ended, and we choose to approximate our data with local curves before blending, we also gain a data reduction for the current stage.

The implemented approximation method in [16], called MARS, provides searching for a suitable knot vector and approximates the data using the B-spline. It has parameters such us maximum number of basis functions, and an accuracy of this approximation depends on the size of the data set. The method presented in this paper provides a flexible approximation curve, independent of the complete data set. We do not need to keep all stages. This feature shows the distinction between local curves and control points. In MARS, if we remove the first knots, we completely lose the connection with the previous data points.

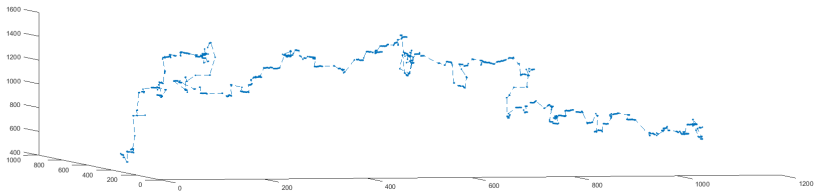
2 Method overview

We seek to obtain a real-time continuous smooth approximation of noisy data, automatization of searching for non-stochastic deviation, and approximation with sufficient accuracy near sudden changes in the data.

The main goal of the algorithm presented in this paper is searching for features of the data. In other words, the algorithm provides preparation for clusterization of the data. Based on that, we consider to use data with the following definition: the 2- or 3-dimensional point set, which is the time dependence of some value, and can be divided into "stages". For example, change of temperature caused by the weather can be divided by times of day or seasons, depending on the length of the data set. In this paper we have considered open data from the Norwegian weather service yr.no at a specific location [21]. For the 2D data set we have used the maximum temperature between 01.12.2016 and 01.03.2017, measured once per hour, as shown in Figure 1(a). For the 3D data set we have used the data from the weather radar, as the pixel's coordinates of the maximum amount of precipitation movement, as shown in Figure 1(b).



(a)



(b)

Figure 1: The data is taken from yr.no [21]. The dashed lines between the data points are generated to obtain a more clear view. Figure 1(a) shows time dependence of the temperature in Narvik during winter months, the timestep is one hour; figure 1(b) shows time dependence of the pixel's coordinate of the maximum amount of precipitation in Nordland taken from the weather radar.

In order to choose our approximation method we first notice some important properties: the data set has been collected throughout a long time and is has unpredictable dynamics. Additionally, the number of points in-

creases with time, i.e., we based our approximation on the amount of points available at the current time step.

From existing methods for building regression models we will take the Multivariate Adaptive Regression Splines method (also known as MARS) [11, 16] for comparison with our implemented method. The MARS method makes cubic B-spline approximation and provides optimization of knots positions and the number of knots. The maximum number of basis functions can be set as a parameter.

The approach is to recognize the changes between "stages" on the data action, when they occur, where the duration of one "stage" is unknown. Thus, we need to provide the approximation of each "stage". We can use polynomial approximation for this purpose because the dynamics within one "stage" does not change sharply. Then we blend these local polynomial curves together continuously and smoothly.

The algorithm considered in the present paper combines several approximation methods. We initiate the approximation with a rough estimate and improve it to obtain clustered sets of points. The motivation is to keep the accuracy independent of the length of on the point set, to provide stable real-time approximation, since the size of the data is unknown.

Consider the following short outline of the sequence of algorithms constituting the main algorithm proposed in the paper. Transition from one step to to the next occurs only if the step returned a value. Throughout this text, kv is an array of the elements of the knot vector, ℓ is an array of local curves, A is the blending spline approximation curve, i is the index of a local curve, and t is time.

(i) Statistical searching of a "stage" or, in other words, element of a knot vector, **return** $kv(i)$; (see Algorithm 1)

(ii) Compute the $(i-1)^{\text{th}}$ local curve, **return** ℓ_{i-1}; (using formula (1))

(iii) Blend ℓ_{i-1} together with ℓ_{i-2}, **return** parametric curve $A(t)$; (see Algorithm 2 and formula (3))

Extension of the knot vector:

(a) Find candidates for knot insertion from the knot intervals; (see Algorithm 3)

(b) Find the positions for new knots, **return** recomputed arrays kv, ℓ, and curve $A(t)$; (see Algorithm 4)

The sequence above is realized for each time step.

3 GERBS

In this section we shall briefly consider some of the theory of blending type spline constructions, which is relevant for this work. A comprehensive study of GERBS can be found in [9, 17].

In contrast to classical B-splines [4, 18], where the coefficients typically are control points in some Eucledian space $\mathbb{R}^n$, blending spline coefficients can be local geometry. Here we have opted to use Bézier curves [2] defined by

$$\ell_i(t) = \sum_{k=0}^{d} c_k b_{d,k}(t), \text{ if } t_{i-1} \leq t < t_{i+1}, \tag{1}$$

where $b_{d,k}$ are the Bernstein polynomials [1] of degree d, and $c_k \in \mathbb{R}^2$ are coefficients (or control points), t_{i-1} and t_{i+1} are blending spline interpolation knots, as described below.

We find the coefficients c_k by using the Least Squares Method [20]. According to this method, the vector of coefficients $c = (c_0, c_1, ..., c_d)^T$ is the solution of the common equation:

$$c = (W^T W)^{-1} W^T y,$$

where the vector y consist of values of the time dependent variable, and W is the matrix of basis functions.

The general formula for an expo-rational B-spline (ERBS) [9] curve over the knots $(t_i)_{i=0}^{n+1}$ is:

$$f(t) = \sum_{i=1}^{n} \ell_i(t) B_i(t), \text{ if } t_1 \leq t \leq t_n, \tag{2}$$

where the coefficients ℓ_i in our case are the local Bézier curves (1), and $B_i(t)$ are the expo-rational basis functions [17].

The domain of the local curves has to correspond with the domain of the respective ERBS basis function. This means that ℓ_i in (2) is defined on the domain (t_{i-1}, t_{i+1}), as shown in (1). Blending of local curves in the global domain $[0, 1]$ is performed via the formula

$$f(t) = \begin{pmatrix} 1 - B_{i-1} \circ \omega_{i-1}(t) & B_i \circ \omega_i(t) \end{pmatrix} \begin{pmatrix} \ell_{i-1}(t) \\ \ell_i(t) \end{pmatrix}, 0 \leq t \leq 1, \tag{3}$$

where $\omega_k(t) = \frac{t - t_k}{t_{k+1} - t_k}$, t_k and t_{k+1} are knots, and our choice of B is the logistic expo-rational B-function [22]:

$$B(t) = \frac{1}{e^{\frac{1}{t} - \frac{1}{1-t}} + 1}.$$

The choice of using expo-rational basis functions is founded on some important properties of these functions [17]. One main difference when compared to polynomial B-splines is the continuity properties. The continuity at the knots will increase with the order of the B-function. The expo-rational basis functions are C^{∞}-smooth at the respective exterior knot and all derivatives are zero at their interior knot. The blending splines in (2) possess a transfinite Hermite property, namely, $\ell_i(t)$ and all their existing derivatives are interpolated at the interior knots t_i. Figure 2 shows that the first derivative of the basis function is zero at the start and at the end of its support.

The speed of expo-rational B-splines occurs in a more "natural" way than the speed of polynomial B-splines, as shown in Figure 3. Its minimums are at the start and at the end of curve, local minimums at the turns and local maximums at the straights. We will use these speed properties further as criteria for inserting new knots and finding knot positions.

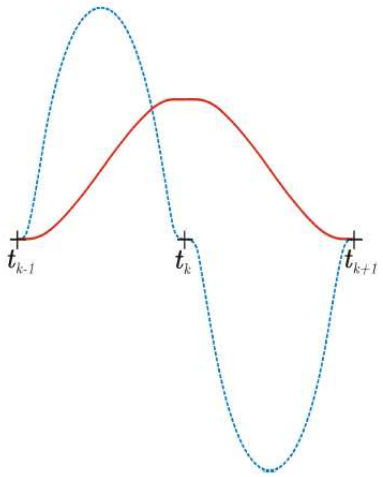


Figure 2: A graph of the expo-rational basis function (solid red) and its first derivative (dotted blue).

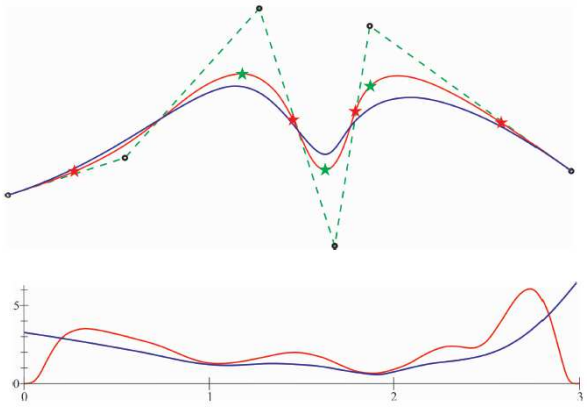


Figure 3: The top picture shows a plot of a 3^{rd} degree polynomial B-spline (blue). The dashed green lines are the control polygon. The red curve shows an expo-rational B-spline using the same knot vector and control points as the polynomial B-spline. The red and green stars denotes extreme values of the speed. The bottom picture shows the comparison between the speeds of these curves. Here the blue function shows the speed of the polynomial B-spline whereas the red function denotes the speed of the expo-rational B-splines.

4 Statistical method for real-time approximation

Our data are noisy, so we can divide them into noise and a non-stochastic part. We assume that the noise tends towards a normal distribution. We will construct probability distribution functions for some initial number of points, and successively add points to the point set. The deviation from the normal distribution yields a new stage of activity. Such a method facilitates the so-called *real-time*, that is, we run the algorithm while we receive new data.

Algorithm 1 provides searching for knots by comparing the normal distribution with the probability distribution function for an open ended stream of data.

A virtual example of Algorithm 1 is shown in Figure 4. Here, the green curves are normal distributions, where the red curves are probability distribution functions (PDF) for all points between each pair of blue lines. The blue lines illustrate the *stages* array, which corresponds to the knot vector.

Let us imagine that the set of points is the random translation of one point. Then for each time step it makes a displacement with an average value of 1. The initial value for displacement is 0 for both axes. Since the PDF can be unstable for small numbers of points, we have used a constant interval as a minimum distance between the knots.

Algorithm 1 Statistical searching of "stage"

procedure SEARCH STAGE
start = 1;
step = 4; *//take a* step *number of points for computation of the next PDF*

    for $t = start + 1 : step : T$ **do**

        compute $\sigma(start : t)$, $\mu(start : t)$; *//σ is the standard deviation and μ is the mean value*

        $nd = \frac{1}{\sigma\sqrt{2\Pi}} e^{-(x-\mu)^2/2\sigma^2}$; *//normal distribution*

        f =pdf($y(start : t)$); *//compute probability distribution function*

        if f has > 1 local maximums **then**

            $start = t$;

            append *start* to *stages*;

    return *stages*

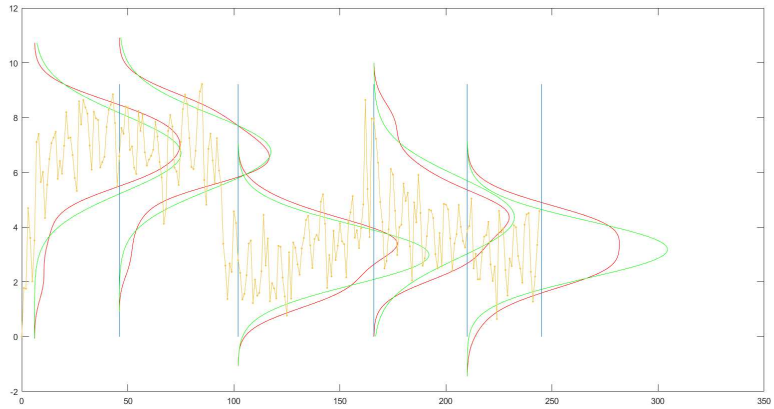


Figure 4: Illustration of statistical "stages", i.e., initial knots, searching by Algorithm 1 for 2D data (see Figure 1(a)). The blue lines show recognized knots, red lines show probability distribution functions of groups of data points separated by the blue lines, and the green curves are the normal distribution functions for those data points.

For the $\mathbb{R}^3$ case we define translation of the point as a continuous displacement of the $\mathbb{R}^3 \to \mathbb{R}^2$ projection.

The distance between two points is

$$|\mathbf{d}_j| = |\mathbf{p}_{j+1} - \mathbf{p}_j|. \tag{4}$$

The initial displacement D_0 is zero, and accumulates the distance (4) between the first and the second point. Then we choose the direction D_{t+1} of translation as the sign of the difference between the y coordinates as shown in formula (5).

$$D_{t+1} = D_t + \text{sign}(y_{t+1} - y_t)\sqrt{(x_{t+1} - x_t)^2 + (y_{t+1} - y_t)^2}. \tag{5}$$

By involving these steps we get the picture of a random walk, see Figure 5, to which we can apply Algorithm 1.

Figure 5 shows translation of the point as a projection from 3D to 2D, using formula (5), combined with an illustration of the searching for "stages".

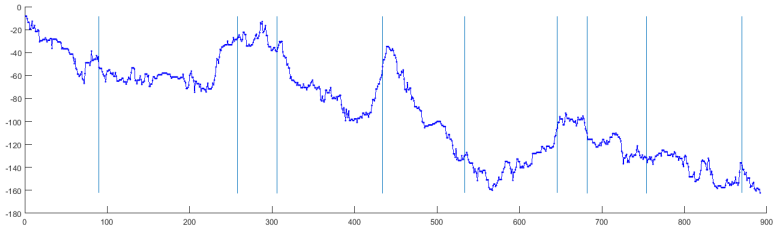


Figure 5: Illustration of searching for "stages" for 3D data (see Figure 1(b)). The distance between neighbor points in this figure is equal to the distance between neighbor points in 3D, and the direction corresponds to the projection on the plane determined by the t and y axes. The blue lines show the knots found by applying Algorithm 1.

Algorithm 2 provides construction of local curves, as Bézier curves of degree 3, and blending them together. A description of such a spline construction was addressed in Section 2 (see formula (2)). In Algorithm 2, the input value *stages* is adjusted by applying Algorithm 1. N denotes the number of stages, which changes with time, and $N + 2$ is the number of knots. We begin to compute the i^{th} local curve when we have $i + 2$ knots. The blending starts when we have a minimum of two local curves. Finally, by repeating the procedure, we obtain the blending spline $A(t)$. We make the observation that since our initial knot vector will increase with time, we can draw local curves and blend them in real-time.

Algorithm 2 Blending of local curves

procedure GERBS($stages$, N)
$kv = [stages(0), stages(0), stages(1), ...$
$..., stages(N-1), stages(N), stages(N)]$; //kv *is the knot vector*

for $i = 1 : N$ **do**

$dom = kv_{i+1} - kv_{i-1}$; //*define the domain of the* i^{th} *local curve*

$curve_i = \text{Bezier}(y(dom, 3))$; //*compute the Bézier local curve of degree* 3 *on the domain* dom *using formula (1)*

if $N > 2$ **then**

for $t = 0 : kv(N)$ **do**

$$A(t) = \begin{pmatrix} 1 - B_{i-1} \circ \omega_{i-1}(t) & B_i \circ \omega_i(t) \end{pmatrix} \begin{pmatrix} curve_{i-1}(t) \\ curve_i(t) \end{pmatrix};$$

//*blending using formula (3)*

return A

5 Adding knots

In the procedure above we defined the initial knot vector for a current time step. If this vector contains at least two knot intervals, we can improve the approximation of the data, i.e., increase the accuracy, by inserting new knots at certain positions.

Let $A(t)$ be a spline function. X is the discrete set of points, $\mathbf{x}_t \in X$, $\mathbf{x}_t = (t, y)$ for the 2D case and $\mathbf{x}_t = (t, x, y)$ for the 3D case, where t is the time variable. Thus, we have a point for each t and a continuous approximation of this set of points. The knot vector is denoted by $\boldsymbol{\tau} = (t_i)_{i=0}^{N}$, where N is number of knots.

We introduce *a moving frame*, denoted by its tangent and normal vectors, η and ξ for the 2D case, and tangent, normal and binormal vectors η, τ, ξ, for the 3D case, respectively. Such a frame represents a moving local coordinate system.

Definition 1. *The "scope" is the interior of the geometric boundary outlined by the pair of straight lines for the 2D case, or planes for the 3D case, which are defined via the knot interval. Each knot interval on* $\boldsymbol{\tau}$ *yields the top and the bottom borders of the "scope", which go through the points from the set* $X^i \subset X$, $X^i = [\mathbf{x}_{t_i}, \mathbf{x}_{t_{i+1}}]$, *which have the maximum distance from the local origin along the axis* ξ *of the moving frame, and are parallel to the axis* η,

for the 2D case, or rectifying plane denoted by the axes η and τ, for the 3D case.

The process of inserting new knots consists of two steps: finding candidates for knot insertion from intervals in the knot vector, and defining the position for knot insertion.

a) The first step is based on the detection of points which are outside of the "scope". The "scope" should contain all of the points. Algorithm 3 and Figures 6 and 8 describe a process for finding the indices of intervals where new knots should be inserted.

   We shall now consider separately line 15 from Algorithm 3 for the 3D case and describe how to find the points which are outside of the "scope".

   The parametric expression of the plane which belongs to the "scope" is:

$$\mathbf{q}(u, v) = \mathbf{p}_0 + u\mathbf{t} + v\mathbf{b},$$

   where $\mathbf{t}$ is the tangent vector, $\mathbf{b}$ is the binormal vector, and $\mathbf{p}_0$ is the local origin.

   Let the vector between $\mathbf{p}_0$ and the point to check be $\mathbf{d}_{point}$, the vector between $\mathbf{p}_0$ and A be $\mathbf{d}_{curve}$, and $\mathbf{n}$ be the normal vector of the curve at $\mathbf{p}_0$ (see Figure 8(c)). If the inner product between $\mathbf{n}$ and $\mathbf{d}_{point}$ has the same sign as the inner product between $\mathbf{n}$ and $\mathbf{d}_{curve}$, then the point and the curve lie on one side of the plane:

$$\textbf{if}$$

$$\langle \mathbf{d}_{point}, \mathbf{n} \rangle \langle \mathbf{d}_{curve}, \mathbf{n} \rangle < 0, \tag{6}$$

$$\textbf{then add a new knot}$$

b) The second step is based on the properties of ERBS curve, as considered in Section 2 (see Figure 3). We seek to divide the knot interval by inserting a new knot at the position where we can measure the largest change in the point set. We define this to be the position with the highest speed, thus, we need to find local maximums of the first derivative of the curve. Algorithm 4 and Figure 7 describe the process of inserting new knots. x and y are point coordinates, kv is the knot vector, and ms is an array of local maximums of the first derivative of the curve A. in is an array of indices of knot intervals, and there are two types of indexation, i is the index of the knot interval and t is an index along the time axis.

Algorithm 3 Finding candidates for knot insertion from knot vector

procedure FINDINTERVALS(kv, A) //kv *is the knot vector*, A *is the curve*
$\rho = []$; //*an array for extremal points, which denote the border of the "scope"*

Define the moving frame to $A(t)$; //Fig. 6(a), 8(a)

    for i = 0 : length(kv)-1 **do**
        //*add to* ρ *extremal points along* ξ *on the interval* $[kv_i, kv_{i+1})$
        ρ.add($X(\max(\xi_t))$);
        ρ.add($X(\min(\xi_t))$);

Construct lines through points from ρ*, parallel to the* η *axis, and we get the "scope" for the 2D case*; //Fig. 6(b)
OR
Construct planes through points from ρ*, parallel to the rectifying plane denoted by the* η *and* τ *axes, and we get the "scope" for the 3D case*; //Fig. 8(b)

    if $\exists$ points from X on the interval $[kv_i, kv_{i+1})$ which lie below of the bottom border or above of the upper border **then**
        insert a new knot using Algorithm 4;
        recompute the knot vector and the curve and **goto 2**; // (Fig. 6(c), 8(d))

Algorithm 4 Inserting new knots

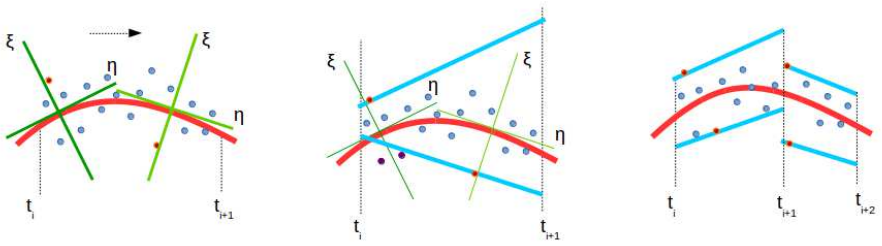
procedure NEWKV(kv, A, N)
$in = []$, $ms = []$;

//*find knot intervals where we need to add new knots, using Algorithm 3*
    in = FINDINTERVALS(kv, A); //in *consists of indices of knot intervals*

ms = LOCALMAXIMUMS(A'); //*populate the array* ms *with indices of local maximums of the* $A'(t)$

Add to kv the indices (t) from ms, which lie on knot intervals with indices (i) from in; //*in the case when values of* ms *or* in *does not lead to add new knot, then we* ***do not*** *add any indices*

return kv



(a) (b) (c)

Figure 6: Describes algorithm 3. (a) illustrates the detection of a moving frame on the curve (Algorithm 3, line 4). The blue lines in (b) are the tangent lines of the curve through the red points, corresponding to maximum and minimum distances along the ξ axis on the considered knot interval, which are used to obtain the "scope" (Algorithm 3, line 11). Since there are points outside of the "scope" (Algorithm 3, line 15), we insert a new knot and recompute the curve as shown in (c) (Algorithm 3, lines 16-17).

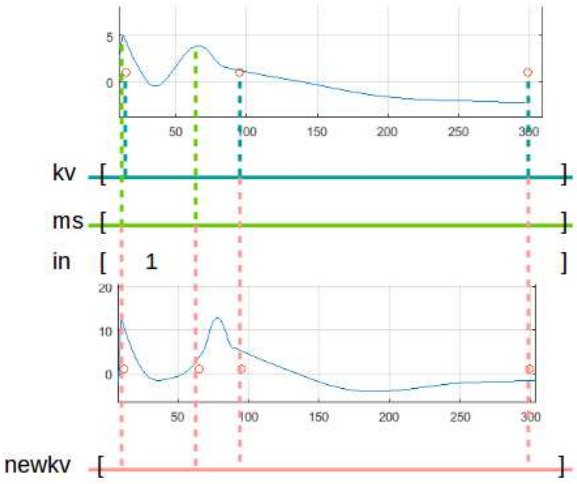


Figure 7: Relation between the arrays, described in Algorithm 4, which uses Algorithm 3. The blue curves show the derivative of the curve $A(t)$ before (top) and after (bottom) knot insertion. The red circles illustrate the positions of the knots. kv is the knot vector before knot insertion, ms is the array of positions on the time-axis of the maximums of the curve's first derivative, in is the array of indices of the knot intervals where we need to insert new knots found by using Algorithm 3, and $newkv$ is the new knot vector, found via comparing the kv, ms and in arrays, as described in Algorithm 4.

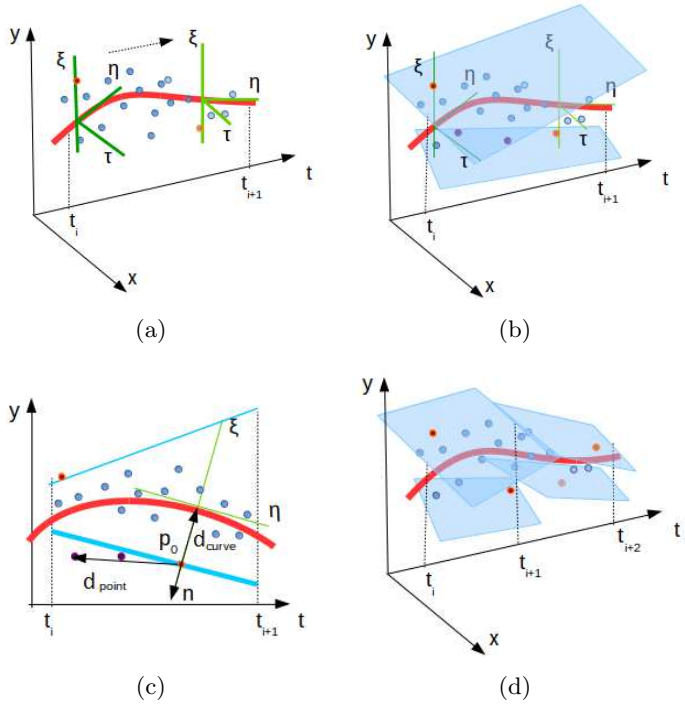


Figure 8: Description of Algorithm 3 for the 3D case. (a) illustrates the detection of a moving frame on the curve (Algorithm 3, line 4). The blue planes in (b) are the tangent planes to the curve through the red points, corresponding to the maximum and minimum values along the ξ axis on the considered knot interval, constituting the "scope" (Algorithm 3, line 13). (c) illustrates how we recognize the points which are outside of the "scope" in the projection on a plane denoted by the t and y axes: $\mathbf{n}$ is the normal to the curve at $\mathbf{p}_0$. We are checking the position of $\mathbf{d}_{point}$ relative to the plane by applying formula (6). Since there are points which are outside of the "scope" (Algorithm 3, line 15), we insert a new knot and recompute the curve as shown in (d) (Algorithm 3, lines 16-17).

6 Results and concluding remarks

Figures 9 and 11 show some steps of the proposed algorithm applied to representative examples in $\mathbb{R}^2$ and $\mathbb{R}^3$, respectively.

The approximation is a continuous smooth function which is generated by the automated algorithm. We can use intrinsic parameters of the resulting curve for further analysis. The settings of the algorithm are implicit: sensitivity to the detection of stages by changing the *step*, tolerance to the points which are outside of the "scope" (see Algorithm 3, line 15), and adjusting the degree of the local curves (in our case $d = 3$). We do not need to set the initial number of knots, or initial length of knot-intervals, or number of iterations.

For comparison, we consider the MARS algorithm [16]. Parameters for the MARS model have been set as

```
params = aresparams2('maxFuncs', 70);
```

which limits of the maximum number of basis functions used. Note that the implementation is not a real-time version, so, assuming that we can realize a comparable approach, we simply run MARS for each time step.

By making a visual comparison of Figures 9 and 10 one can see the differences and similarities between the results of the two algorithms. We note that the length of the resulting knot vectors for both methods are equal, but the values are very different. For example, the accuracy of the approximation changes within the range $300 - 600$. Also one can compare the range $800 - 1200$ for both algorithms.

One can not conclude which one is the better, based on the curves, since we do not have an original curve. However, we can discuss which method is more fit for our task and for our data.

The method presented in this paper provides flexible approximation of curves, independent of the complete data set. The curve is an affine combination of two and only two local functions on each knot interval. We do not need to keep all local curves or all "stages". This feature shows the distinction between local curves and control points. For comparison, with MARS, if we remove the first knots, we completely lose the connection with the "earlier" data points. But by using local curves, we keep the previous "stage" only as long as we need it.

We observe a constant (but not established) accuracy in our method for any time step. Conversely, with MARS, if $maxFuncs$ is large, then we obtain one accuracy at first, which will decrease when increasing the number of data points. This can be addressed by changing the maximum number of basis functions, but requires an extension of the algorithm.

Thus, we conclude that the presented algorithm is suitable for data possessing similar properties as our model data, whereas MARS is an established method with flexible settings for specific tasks.

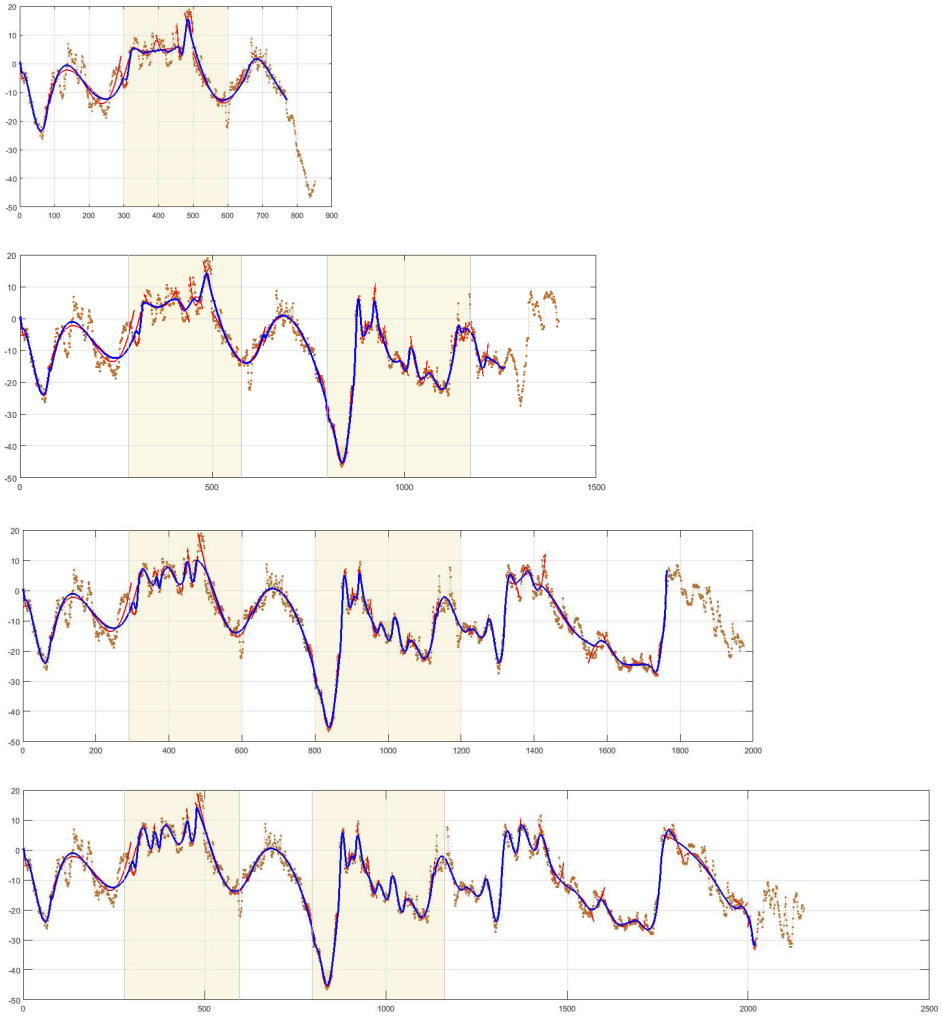


Figure 9: The process of approximation for 2D data (see Figure 1(a)). The red curves are local curves, the blue curve is the approximation curve.

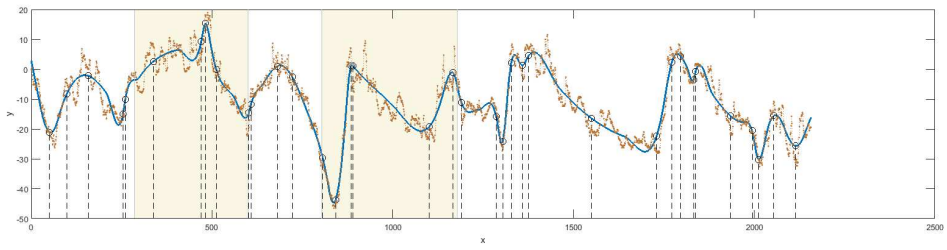


Figure 10: Result of the MARS [16] algorithm.The blue curve is the resulting curve, black circles and dashed lines show knots.

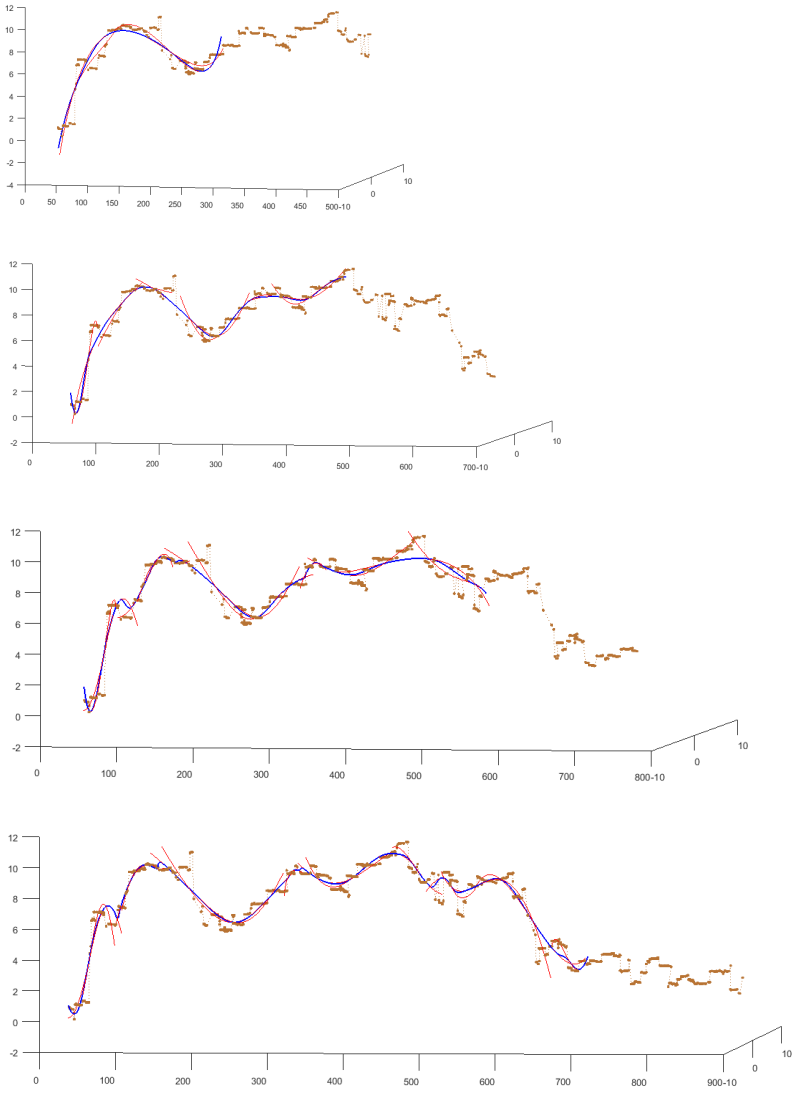


Figure 11: The process of approximation for 3D data (see Figure 1(b)). The red curves are local curves, the blue curve is the approximation curve.

References

[1] S. Bernstein, *Démonstration du théoréme de Weierstrass fondée sur le calcul des probabilités*, Communications de la Société Mathématique de Kharkow, 2-ée série, volume: 13(1), pages:1-2, 1912.

[2] P. Bézier, *Numerical control: Mathematics and Applications*, Wiley series in computing, J. Wiley, London, New York, English language edition, 1972.

[3] K. Bittner, H. G. Brachtendorf, *Fast Algorithms for Adaptive Free-Knot Spline Approximation Using Non-Uniform Biorthogonal Spline Wavelets*, SIAM J. Scient. Computing, 37(2): 283-304, 2015.

[4] C. de Boor, *A Practical Guide to Splines*, Springer-Verlag, New York, 1978.

[5] J. Bratlie, R. Dalmo, P. Zanaty, *Fitting of Discrete Data with GERBS*, in volume 8353 of Lecture Notes in Computer Science, pages 569-576. Springer, 2014.

[6] M. Brezak, I. Petrovic, *Real-time Approximation of Clothoids With Bounded Error for Path Planning Applications*, IEEE Transactions on Robotics, volume: 30, pages: 507 - 515, 2014.

[7] U. Clarenz, M. Rumpf, A. Telea, *Robust Feature Detection and Local Classification for Surfaces Based on Moment Analysis*, IEEE Transactions on Visualization and Computer Graphics, volume: 10, pages: 516-524, 2004.

[8] R. Dalmo, *Expo-Rational B-Splines in Geometric Modeling*, Methods for Computer Aided Geometric Design. PhD thesis, University of Oslo, 2016.

[9] L. T. Dechevsky, A. Lakså, B. Bang, *Expo-Rational B-Splines*, International Journal of Pure and Applied Mathematics, volume: 27 (3), pages: 319-367, 2006.

[10] G. Farin, *Curves and Surfaces for Computer Aided Geometric Design*, Academic Press, Inc., Fifth Edition, 2002.

[11] J.H. Friedman, *Multivariate adaptive regression splines*, Stanford linear accelerator center, Stanford, California, volume: 19, pages: 1-67, 1990.

[12] I. Guttman, *Introductory Engineering Statistics*, John Wiley & Sons, Inc., 1965.

[13] P. J. Hartley and C. J. Judd, *Parametrization of Bézier-type B-spline curves and surfaces*, Computer Aided Design, volume: 10, pages: 130-134, 1978.

[14] W. Härdle, *Applied Nonparametric Regression*, volume: 34(2), pages: 341–342, 1989.

[15] G. M. James, *Curve Alignment by Moments*, The Annals of Applied Statistics, volume: 1, pages: 480–501, 2007.

[16] G. Jekabsons, *ARESLab: Adaptive Regression Splines toolbox for Matlab/Octave*, Institute of Applied Computer Systems, Riga Technical University, Riga, Latvia, 2009.

[17] A. Lakså, *Blending technics for curve and surface constructions*, Narvik University College, Narvik, Norway, 2012.

[18] L. L. Shumaker, *Spline Functions*, Cambrige University Press, Cambridge Cb2 8RU, UK, 3rd edition, 2007.

[19] M. D. Van der Walt, *Real-time, local spline interpolation schemes on bounded intervals*, Applied Mathematical Sciences, volume: 10, pages: 205-234, 2015.

[20] K.V. Vorontsov, *Lectures about algorithms for dependencies reconstruction*, 2007.

[21] Weather service yr.no, *www.yr.no/place/Norway/Nordland/Narvik/Narvik/almanakk.html, www.yr.no/place/Norway/Nordland/Narvik/Narvik/radar.html*, Norwegian Meteorological Institute and Norwegian Broadcasting Corporation, 2007 - 2017, dates 01.12.2016-03.01.2017.

[22] P. Zanaty, L. T. Dechevsky, *On the numerical performance of FEM based on piecewise rational smooth resolutions of unity on triangulations*, AIP Conference Proceedings, volume: 1570, page:191, 2013