Abstract. In this paper we treat the problem of smoothing a user input in form of a stroke consisting of line elements. We pose the problem as a minimization problem for the Dirichlet energy of the resulting spline under the constraint that it must lie inside the $\varepsilon$--offset of the input. We present an approximation solution that works with three steps, namely simplification, regularization and smooth rendering.

The simplification step is described in more detail in our paper [1], we will only recall the most important facts and use the algorithm as it is. In the regularization step we subdivide the simplified spline into equidistant segments and then apply the Savitzky-Golay filter to it. One must take special care to ensure that the regularized spline lies inside the $\varepsilon$--offset of the input. We describe a heuristics that guarantees it.

The smooth rendering step consists in using the nodes of the regularized spline as control points of a $B$--Spline. We discuss the quality and outcome of this approach and give a few examples.

Keywords: Line smoothing · Dirichlet Energy · Savitzky-Golay filter · $B$--splines

1 Introduction

The algorithm that we present here can be used in post-processing of user input in $2d$ and $3d$--graphical user interface or GUI applications. One scenario is that the user draws a free-form stroke on the plane or a curve in $3d$. Another scenario is that when the user works with $3d$--meshes, he makes selections of facets and needs to display their boundary. A piecewise linear boundary does not look nice and needs to be smoothed. We would like to present him with a regularized and smooth version of his stroke or facet selection boundary.

Consider a data set given by a piecewise linear curve or a polyline spline $S$, which is defined by control points connected by straight lines. Our approach consists in three steps. In the first step we reduce the number of input points and straighten up the input using a line simplification algorithm. We then regularize the simplified line, in the sense that we smooth the edges by using the Savitzky-Golay or $SG$--filter. Finally, we use the positions of the points of
the regularized spline as control points of a $B -$spline in order to get a globally
twice differentiable curve.

The plan of the paper is as follows. In this section we discuss the literature
and the previous work. In the second section we provide definitions and exact
mathematical formulations of the problems we are treating. We also include a
definition of the discrete Dirichlet energy of a spline.

In the third section, we present our solution and discuss the necessity and
benefits of each step. We also give a high level overview of each step.

In the fourth section we describe in more detail the implementation of the
$SG -$filter for our use case, including the pre-processing steps and a heuristic to
make sure that the result lies in the $\varepsilon -$offset.

In the fifth section, we show a few examples of smoothing real world user
input with our algorithm. The sixth section is a short conclusion.

We have included a mathematical appendix, with three subsections. In the
first, we derive the discrete Dirichlet energy of a spline. In the second subsection
we give a short derivation of the $SG -$filter and show how it reduces to a weighted
average scheme. In the third subsection we give a short description of $B -$Splines
and how we use them to generate the final outcome.

1.1 Literature discussion

There is a rich literature about the line smoothing problem. From the many
solution methods, we will focus on five classes, namely averaging methods, Chai-
kin algorithm, Hermitean spline approximation, $B -$spline smoothing and active
contour or snake methods.

Averaging methods work by replacing the position of a vertex through the
weighted average of its neighbors. Well known averaging methods include Lapla-
cian smoothing, the Savitzky-Golay filter [2], MacMaster averaging [3] and the
smoothing via iterative averaging or SIA method [4]. Borisov et al. in [5] have
made a comprehensive comparison of smoothing methods based on averaging.
Their conclusion is that by using correct parameters, the SG-filter has the overall
best results. This is no surprise since the weights in the SG-filter are specifically
chosen to make a polynomial least squares fit to the selected points.

The Chaikin corner cutting algorithm [6] is very simple and fast and provides
very good smoothing results. It works by subdividing each segment of the spline
into three parts left, center and right. It removes the left and right parts and
connects the endpoints of the neighboring central parts. This procedure is equiva-
ient to cutting the corners on each vertex. Repeated application of this procedure
leads to a smooth curve with no sharp corners. The disadvantage of this method
is that one cannot control the distance of the smoothed curve to the input curve.
We believe that a modification of this algorithm that cuts corners in such a way
that the resulting curve does not lie outside the given distance tolerance might
be interesting to consider.

The Hermitean spline approximation method is based on the following princi-
ple. It uses the space of piecewise cubic splines with continuous first derivatives
that are called general cubic splines or continuous second derivatives that are
called natural cubic splines. Given a set of points \((x_i, y_i)\), one represents it as a piecewise linear function \(f\) and searches a piecewise hermitean spline \(h\) that minimizes the \(L^2\) norm \(\|h - f\|\) of their difference in the space of all hermitean splines. This is equivalent to a least-squares fit of a piecewise hermitean spline \(h\) to the function values \(f\).

This method is mathematically satisfying in the sense that one has guaranteed error estimates and in the case of natural splines, one has a globally twice differentiable function. In [7, 8] the authors give an excellent exposition of these methods and describe practical recipes in form of linear least square matrix systems and various solution methods for them. Using them, one can obtain spline approximations for input data-sets with random noise. Especially relevant for our setting is the natural spline approximation, since it is globally twice differentiable. The downside of these methods is that the setup and solution of a least squares linear system is costly in terms of memory and runtime.

The selection of the knot points for the spline is a non-trivial problem that cannot be solved linearly. In [9] the authors discuss thoroughly the best methods to select the optimal knot positions. One either has to use a heuristic or some non-linear optimization method to get the best result.

B-spline smoothing [10, 11] is another method that is similar to the Hermitian spline approximation. It fits a set of control points to a set of input points so that the difference between the B-spline and the input points is minimal in the least squares sense. Since the dependency between the control points and the B-spline curve is not linear, one needs to either use heuristics or non-linear optimization to estimate the best positions.

Active contour or snake methods [12, 13] use splines that have an internal bending and curving energy that gives them excellent smoothness and adaptive properties. One can use the input points to define an external energy to make the active contour converge towards a smooth approximation of the data.

The active contours or snakes have very good smooth approximation properties. Their downside is that the energy terms used in their definition are inherently non-linear and one has to use non-linear parameter estimation methods like Gauss-Newton to find the correct final position. This is even more complicated and numerically costly than the previous approaches, even though the results have excellent smoothness properties.

2 Definitions and problem formulation

In this section, we introduce the most important concepts and notation used in the paper. We refer to our paper [1] and the textbooks [14, 15] for more detailed definitions of line segments and splines.

2.1 Splines and \(\varepsilon\)-offsets

Given a set \(\{P_i\}\) of points, we define their line segments by \(s_{ij} = s(P_i, P_j)\).

**Definition 1.** *(Polyline splines)*
A polyline spline $S = (P, \sigma)$ is a set of control points $P = \{P_i | 0 \leq i \leq n\}$ with line segments $\sigma = \{s_{i,i+1} | 0 \leq i < n\}$ connecting any two consecutive points $P_i, P_{i+1}$.

The distance between a point $Q$ and a spline $S$ is given by

$$\text{dist}(Q, S) = \min_{s \in \sigma} \{\text{dist}(Q, s)\}.$$

Definition 2. (Spline $\varepsilon$-offset)

- For a line segment $s$, one defines its $\varepsilon$-offset $O(s, \varepsilon)$ as

$$O(s, \varepsilon) = \{P \in \mathbb{R}^2 | \text{dist}(P, s) < \varepsilon\}$$

- The $\varepsilon$-offset $O(S, \varepsilon)$ of a spline is defined as

$$O(S, \varepsilon) = \{P \in \mathbb{R}^2 | \text{dist}(P, S) < \varepsilon\}$$

or equivalently $O(S, \varepsilon) = \bigcup_{s \in \sigma} O(s, \varepsilon)$.

Definition 3. (Bounding box) Given a spline $S = (P, \sigma)$, with $P_i = (x_i, y_i)$ we define the bounding intervals

$$I_x = \left(\min_{0 \leq i \leq n} \{x_i\}, \max_{0 \leq i \leq n} \{x_i\}\right)$$

$$I_y = \left(\min_{0 \leq i \leq n} \{y_i\}, \max_{0 \leq i \leq n} \{y_i\}\right).$$

The bounding box $B_S = I_x \times I_y$ is defined as the 2d-rectangle obtained as the cartesian product of the corresponding $x, y$-bounding intervals.

If the points $P_i$ lie in $n$-dimensional space, then the bounding box would be defined similarly as $B_S = I_{x_1} \times I_{x_2} \times \ldots \times I_{x_n}$ using $n$ bounding intervals $I_{x_1}, I_{x_2}, \ldots, I_{x_n}$ for each dimension.

2.2 Discrete Dirichlet energy

The curves $S$ obtained by the line simplification algorithm are piecewise linear $C^0$ curves on the interval $[0, 1]$ represented by arc-length parametrization. They are sharp at the corners and consist of lines of different lengths. In order to obtain visually more appealing curves we need to quantify their visual quality. We do this by measuring their discrete Dirichlet energy and their class of differentiability.

Definition 4. Consider a spline $S = (P, \sigma)$. 
Line smoothing algorithms 5

– Average neighboring segments length $A_i$ through a vertex $P_i$ is given by

$$A_i = \frac{\|s_{i-1,i}\| + \|s_{i,i+1}\|}{2}.$$

– Unit segment direction $u_{i,i+1}$ is given by

$$u_{i,i+1} = \frac{s_{i,i+1}}{\|s_{i,i+1}\|}.$$

We now have everything we need to define the discrete Dirichlet energy.

**Definition 5.** The discrete Dirichlet energy of a spline $S = (P, \sigma)$ is defined as

$$E[S] = \sum_{i=1}^{n-1} \|(\nabla S)_i\|^2 = \sum_{i=1}^{n-1} \frac{2 - 2 \langle u_{i-1,i}, u_{i,i+1} \rangle}{A_i^2}.$$

### 2.3 Line smoothing problem

Given a piecewise linear curve $S$ we would like to displace its vertices, so that the displaced spline lies in the $\epsilon$--offset of $S$ and is as smooth as possible. We would also like the curve $C$ corresponding to $S'$ to be globally at least twice differentiable. This is made more precise by the following statement.

**Problem 1.** (Line smoothing) Given a spline $S = (P, \sigma)$, find new positions $P'$ for its vertices, so that for $S' = (P', \sigma')$ we have

– $S' \subset O(S, \epsilon)$,
– $E[S']$ is minimized.

One can obtain an exact solution to minimization of the Dirichlet energy by solving a discrete Poisson equation or using harmonic functions. But there is no guarantee that the resulting curve lies in the $\epsilon$--offset. Also, this approach leads to numerically costly solutions and we look for much simpler approaches using cubic splines.

In Figure 1, one can see a typical use case. We have an input spline, whose segments have been subdivided equidistantly. On the right, we show another spline in red, whose vertices have been displaced so that the resulting spline is smoother and has a lower Dirichlet energy.

The solutions that we propose do not solve the energy minimization problem exactly but due to their construction have a lower Dirichlet energy than the input. And they also regularize the shape of the input in the sense that they fit a polynomial to a set of neighboring points.

The B-splines are globally $C^2$. Hence our solution strategy consists in first obtaining a curve with low Dirichlet Energy from the input using the $SG$–filter. We then ensure that the result lies in $O(S, \epsilon)$ and finally generate a globally twice differentiable curve for rendering.
3 Our solution

3.1 Solution idea

As discussed in subsection 1.1, one can differentiate between three basic approaches to smoothing a polyline spline, namely neighboring points averaging, least squares fit of a cubic or higher order spline or B-spline smoothing.

All of these approaches have their advantages and disadvantages. The first approach is very simple and fast but is not satisfactory as far as smoothness and low frequency noise removal is concerned. The second approach is very good as far as regularity accuracy and adaptability is concerned. It is not straightforward to implement and often requires the setup and solution of a linear system of equations. The third approach is excellent as far as smoothness is concerned, but requires the optimization of a non-linear problem. The last two approaches are computationally not very efficient and hence not preferable for an algorithm that is used interactively in a graphical user interface.

Our approach tries to make a balanced combination of the three. In the line simplification step we replace a section of the spline by a single line segment. This can be interpreted as a local least squares line fit. In this way we linearize the input as much as it is possible for a given tolerance value \( \varepsilon \). At the same time, it removes high as well as low frequency noise from the input.

The SG-filter is a local least squares polynomial fit and for this reason has excellent smoothness and regularity properties. But it is implemented as a weighted average scheme, which is very simple and efficient. And finally the B-splines produce a globally smooth curve without the need to solve any systems of linear equations.

For this reason, we believe that our algorithm offers a reasonable balance between weighted averaging schemes, least squares fit and cubic spline approximations.
3.2 Solution steps

The solution we propose consists of four steps.

- In the first step, we use the line simplification algorithm described in our paper [1]. The purpose of this step is to remove noise from the input and represent the user input using the least possible number of points and lines.
- In the second step, we subdivide the simplified spline into segments of equal length and displace the points using the SG-filter in order to achieve maximal smoothness allowed by the offset tolerance $\varepsilon$.
- In the final step, we use the the points from the previous step as control points of a B-spline in order to generate a twice differentiable curve. This guarantees that there are no abrupt changes of directions in the output curve. This gives a visually appealing final result for the user.

In the following subsections we will discuss each of the sub-steps in more detail.

Algorithm 1 Our spline smoothing algorithm

\begin{algorithm}
\caption{SplineSmoothing($S, \varepsilon$)}
2. Perform equidistant subdivision of segments and smooth them using the SG-filter
3. Render the curve using B-splines.
\end{algorithm}

In the image attached to algorithm 1 we have visualized the most important steps of our solution. On the top we draw the spline as it is seen by the user. On the bottom we show the same splines with the vertices represented by small squares.

On the left, with black color we have presented a typical user input drawn with mouse or on a touch screen. It represents the letter A. One can note the
high frequency noise in the input due to the unsteady hand movement of the user. There is also low frequency noise in form of sections that are meant to be straight, but are drawn imprecisely. In orange we have depicted the output after the first step of our algorithm. One can see that most of the input points and the high and low frequency noise has been removed.

In green, we have represented the output of the steps 2 and 3. One can see that the vertices are equidistant and the corners have been smoothed out. At the same time, the points have not moved too far away from the original input.

In red we have presented the output of the final step of our algorithm, where the spline has been rendered as a twice differentiable smooth curve using B-splines. Finally, we have drawn both the original raw input and the final smooth output to visualize the difference between both splines. The maximal difference between the red and the black spline is smaller than the predefined tolerance value $\varepsilon$.

### 3.3 Line simplification

We have treated this problem in more detail in our paper [1] and refer the user to it for more details.

### 3.4 Savitzky-Golay filter

Given a simplified and rescaled curve $S'$, we subdivide it into pieces of equal length $l$ and apply the following weighted average transformation to each of its points

$$P_i = \sum_{j=-k}^{k} C_j P_{i+j}.$$  

**Definition 6.** One uses the following terminology

- The constant $w = 2k + 1$ is called the filter **window size**
- The coefficients $C_{w,d} = \{C_{-k}, \ldots, C_0, \ldots, C_k\}$ are called the **filter coefficients**.
- $d$ is called the **degree** of the fitting polynomial $p$.

To different values of $w$ and $d$, correspond different filter coefficients $C_{w,d}$.

The $SG$–filter is simple and efficient, but at the same time has excellent smoothing and regularization properties. It combines a polynomial least squares fitting scheme, that gives visually pleasing results with a weighted averaging scheme that is simple to implement and efficient to calculate.

In order to use the $SG$–filter for our purposes and obtain optimal results, we need to post-process the input and make sure that no point has left the $\varepsilon$–offset after the smoothing. For a more detailed description, please see the next section.
3.5 Rendering using B-splines

Consider the set \( \{P_0, P_1, \ldots, P_n\} \) of control points obtained from the previous step. The B-spline curve is given by

\[
P(t) = \sum_{i=0}^{n} B_{i,m}(t) \cdot P_i
\]

The values \( B_{i,m}(t) \) can be calculated using a polynomial evaluation scheme and using only a small subset of \((k+1)\) control points, where \( k \) is the degree of the B-spline. We use B-splines of degree 3, 4 or 5 in our solution. Therefore, the B-splines can be considered as a weighted averaging scheme of the control points and their calculation has linear runtime. For more details, please see subsection 7.2.

3.6 Benefits

The main benefits of our algorithm are that it has a linear runtime in the number of input points, it gives visually satisfactory results and is guaranteed to lie inside the \( \varepsilon \)-offset of the input spline. It is suitable to be used in 2d or 3d graphical user interface applications.

4 Smoothing with the Savitzky-Golay filter

4.1 Pre-processing steps

Equidistant subdivision Before applying the filter, we need to make sure that each segment has length at most \( l \), where \( l \) is a predefined value. If this is not the case, we need to perform an equidistant subdivision of the spline. For this purpose, we check the length of each line segment \( s_{i,i+1} \). If

\[
d_{i,i+1} = \text{dist}(P_i, P_{i+1}) > l,
\]

then we calculate \( m = \left\lfloor \frac{d_{i,i+1}}{l} \right\rfloor \) and add \( m \) equidistant intermediate points \( P_i^{(0)}, P_i^{(1)}, \ldots, P_i^{(m-1)} \) on the segment \( s_{i,i+1} \) between \( P_i \) and \( P_{i+1} \).

Rescaling The shape of the offset and the outcome of the smoothing can show high variation depending on the size of the input. In order to work in the same scale, we first define the size of the expected input \( s \). We then measure the bounding box \( B_S = I_1 \times I_2 \) of the input, where \( I_1 = [a_1, b_1] \) and \( I_2 = [a_2, b_2] \) with lengths \( l_1 = \|b_1 - a_1\| \) and \( l_2 = \|b_2 - a_2\| \). We now define a scaling

\[
T = \begin{pmatrix} \frac{s}{l_1} & 0 \\ 0 & \frac{s}{l_2} \end{pmatrix}
\]

and transform each control point of the spline \( S \) by \( T \) to obtain a new spline \( S' \). The bounding box \( B_{S'} = I'_1 \times I'_2 \) has lengths \( l'_1 = l'_2 = s \), which means that \( S' \) has the expected size. We apply all of the following considerations to the rescaled spline \( S' \).
4.2 Savitzky-Golay filter

Given a simplified curve $S'$, we subdivide it into pieces of equal length $l$ and apply the following transformation to each of its points

$$P_i = \sum_{j=-k}^{k} C_j P_{i+j}$$

**Algorithm 2** Savitzky-Golay smoothing

SavitzkyGolaySmoothing($S$, $k$, $d$, $l$)

1. For $i = 0$ To $n$:
   (a) Calculate $m = \left\lfloor \frac{\text{dist}(P_i, P_{i+1})}{l} \right\rfloor$
   (b) If $m > 0$ Then subdivide $s_{i,i+1}$ into $m + 1$ equal subsegments;
2. Using a lookup table for $k$, $d$ Calculate $C_{k,d} = \{C_{-k}, \ldots, C_0, \ldots, C_k\}$.
3. For $i = k$ To $n - k$:
   (a) Update $P_i = \sum_{j=-k}^{k} C_j P_{i+j}$
4. Return $S$;

In the image attached to algorithm 2, on the left side we have presented a raw input in black, the simplified version in orange and the regularized version in green. In the center, one can see the difference between the raw and regularized line. On the right, one can see the vertex positions of the simplified and regularized spline.

The values of $C_{w,d}$ are tabulated for different window sizes and polynomial degrees [2]. In the simplest case of a quadratic polynomial of degree $d = 2$ with window size $w = 5$, that is $k = 2$, one gets

$$C_{5,2} = \left\{ \begin{array}{c} -3 \\ 12 \\ 17 \\ 12 \\ -3 \end{array} \right\} \left\{ \begin{array}{c} \frac{1}{35} \\ \frac{35}{35} \\ \frac{35}{35} \\ \frac{35}{35} \\ -\frac{1}{35} \end{array} \right\}$$

and hence

$$P_i = \frac{1}{35} (-3 \cdot P_{i-2} + 12 \cdot P_{i-1} + 17 \cdot P_i + 12 \cdot P_{i+1} - 3 \cdot P_{i+1})$$
We denote the result of the application of the filter by \( SG(S) \). It is the same spline as the input \( S \) but with displaced control points \( SG(P_i) \).

### 4.3 Ensuring that all points lie in the \( \varepsilon \)-offset

In order to use the \( SG \)-filter for our purposes and obtain optimal results, we need to post-process the result and make sure that no point has left the \( \varepsilon \)-offset after the smoothing. The simplest way to achieve this is to draw a segment between \( P \) and its displacement \( P' = SG(P) \) and call it \( s \). We have

\[
s = P + t \cdot (P' - P), \quad 0 \leq t \leq 1
\]

If \( l = \text{len}(s) \geq \varepsilon \), then instead of \( P' \) we take the point

\[
P'' = P + \varepsilon / l \cdot (P' - P)
\]

and set \( SG(P) = P'' \). The distance of this point from \( P \) is equal to \( \varepsilon \).

### 4.4 Influence of the input parameters

In the practical implementation, the minimal segment length \( l \), window size \( w \), and polynomial degree \( d \) play an important role in the outcome of the smoothing algorithm. There is a trade-off to be made in the selection of these coefficients so that our solution is as smooth as possible on one hand as has high fidelity on the other. The influence of the input parameters on the outcome is as follows.

For large \( l \) we make few subdivisions to the simplified spline and hence we obtain better smoothing and regularization at the cost of fidelity. If \( l \) is small we get higher fidelity but the input is changed less.

For large \( w \) that the outcome is more regular as there is a larger diffusion between the neighboring points. If \( w \) is small then the regularization and the smoothing are more local in nature.

For large \( d \), the polynomial gives a much more precise interpolation to the input points but we do not obtain much smoothing in this way. For small \( d \) the interpolation is not very precise and we get more displacement of the points but also higher smoothing.

### 5 Examples

We have used the Open Source Inkscape vector graphics application to generate the input. Inkscape has a freehand line drawing tool, where one can set the input smoothing level to 0. So the input that we use represents strokes obtained by freehand movement. Each stroke has between 100 and 300 control points. The width and the height of the strokes is between 2 and 10 cm.

In Figure 2, we have four different shapes that represent typical user inputs, namely drawings of an apple, a circle, an arrow and a doodle. In each of these drawings, we have depicted the outcome of each step of our algorithm with the same colors as in algorithm 1 from subsection 3.2.
6 Conclusion

In this paper, we have presented a new algorithm for line smoothing based on a combination of line simplification, Savitzky-Golay filter and B-splines. In order to measure the quality and smoothness of our output, we use the discrete Dirichlet energy.

It is designed to be used in graphical user interface applications for post-processing of user input. Its main benefits are that it removes noise, regularizes the edges and generates a globally twice differentiable curve as an output. It has a practically linear runtime and is relatively simple to implement.

7 Appendix

In this appendix, we will present some more technical mathematical arguments about the techniques used in the paper.

7.1 Derivation of the Savitzky-Golay filter

This is a standard signal processing and smoothing technique that gives excellent results in our case. We present a short argument for its derivation. For more details, one can consult the papers [2, 17].

We derive the equations of the SG-filter for a set of equidistant points that vary in one single direction. We will use this to generalize the result to an arbitrary curve in any number of dimensions.

Consider a set of points $P_i = (i, x_i)$ on the $2d$ plane. The basic idea is as follows:

- Take a set of input points consisting of a central point $P_j$ with $k$ neighboring points

$$P_{j-k}, \ldots, P_j, \ldots, P_{j+k}$$
before and after. We call the number of selected points \( w = 2k + 1 \) the window.

- Fit a polynomial \( p \) of degree \( d \) to these points in the least squares sense.
- Displace \( P_j \) to the value of \((j, p(0))\) and repeat this for each \( P_j \).

It turns out that this approach can be implemented using a very simple weighted average scheme on the points with pre-calculated weights. Hence this approach leads to a linear filter of the points, where one moves the window one point at a time and displaces each point into its weighted average.

A more precise description is as follows. Given a set of \( 2M + 1 \) points 

\[
\{x(m)\}, \quad -M \leq m \leq M,
\]

fit a polynomial of degree \( N \)

\[
p(m) = \sum_{k=0}^{N} a_k m^k
\]

that minimizes the mean-square approximation error

\[
E[p] = \sum_{m=-M}^{M} (p(m) - x(m))^2
\]

\[
= \sum_{m=-M}^{M} \left( \sum_{k=0}^{N} a_k m^k - x(m) \right)^2
\]

We are only interested to approximate the central value \( x(0) = p(0) = a_0 \). Hence for the SG-filter, we only want to obtain the value of the first coefficient \( a_0 \) of \( p \). It is important to note that we implicitly suppose that the points are equidistantly distributed and expect to have \( p(m) \approx x(m) \).

In order to find the normal equations, we differentiate \( E[p] \) with respect to \( a_i \), set the partial derivatives equal to zero and get

\[
\frac{\partial E[p]}{\partial a_i} = \sum_{m=-M}^{M} 2 \cdot n^i \cdot \left( \sum_{k=0}^{N} a_k m^k - x(m) \right)
\]

These equations can be written as

\[
\sum_{k=0}^{N} \left( \sum_{m=-M}^{M} m^i m^{i+k} \right) \cdot a_k = \sum_{m=-M}^{M} m^i \cdot x(m)
\]

We note that in order for these equations to be well defined, we need at least as many data points as the degree \( N \) of the polynomial.

We define the so called design matrix \( A \) for the polynomial approximation. It is a \((2M + 1) \times (N + 1)\) matrix, defined by

\[
A = \{a_{i,m}\} = \{m^i\}, \quad -M \leq m \leq M, \quad 0 \leq i \leq N,
\]
or more explicitly

\[
A = \begin{pmatrix}
1 & -M & (-M)^2 & \cdots & (-M)^N \\
1 & M + 1 & (-M + 1)^2 & \cdots & (-M + 1)^N \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
1 & M & M^2 & \cdots & M^N \\
1 & M - 1 & (M - 1)^2 & \cdots & (M - 1)^N \\
\end{pmatrix}
\]

Denoting by \(A^T\) the transpose of \(A\), we get the following normal equation

\[
A^T A = \left\{ \sum_{n=-M}^{M} \alpha_{i,m} \cdot \alpha_{m,k} \right\} = \left\{ \sum_{n=-M}^{M} m^{i+k} \right\}.
\]

If we write the data in column vector form \(a, x\) as

\[
a = \begin{pmatrix}
a_0 \\
a_1 \\
\vdots \\
a_{N-1} \\
a_N \\
\end{pmatrix}, \quad x = \begin{pmatrix}
x(-M) \\
x(-M + 1) \\
\vdots \\
x(M - 1) \\
x(M) \\
\end{pmatrix}
\]

then the normal equations can be written as

\[
A^T A \cdot a = A^T \cdot x,
\]

whose solution is

\[
a = H \cdot x
\]

with \(H = (A^T A)^{-1} \cdot A^T\). For the term \(a_0\), we only need the first row \(h_0\) of \(H\), hence we get the following weighted average of the input points

\[
a_0 = h_0 \cdot x = h_{0,0} \cdot x(-M) + \ldots + h_{0,M} \cdot x(0) + \ldots + h_{0,2M} \cdot x(M).
\]

The row \(h_0\) can be pre-calculated and tabulated for polynomials of different degrees since its value is independent of the user input. It only depends on the design matrix \(A\).

### 7.2 B-spline approximation

This is another standard approximation technique introduced by Schoenberg [15, 18]. We will use the recursive de Boor-Cox algorithm for B-spline evaluation [19, 20]. The name B-spline refers to basis splines. They are a special class of polynomials \(B_{i,m}(t)\) of degree \(m - 1\) with compact support on a subset of the interval \([0, 1]\). The \(B_{i,m}(t)\) are called B-spline basis functions and they build a partition of unity, that is

\[
\sum_{i=0}^{n} B_{i,m}(t) = 1
\]
Given a set \( \{P_0, P_1, \ldots, P_n\} \) of control points to be interpolated, the B-spline curve is given by

\[
P(t) = \sum_{i=0}^{n} B_{i,m}(t) \cdot P_i
\]

The degree \( m \) can be chosen between 2 and \( n + 1 \). The basis functions can be calculated in the following way.

First define a vector \( T = (t_0, t_1, \ldots, t_{n+m}) \), called the knot vector. We take \( t_0 = 0 \) and \( t_{n+m} = 1 \). The intervals \([t_i, t_{i+1}]\) build the compact support for the partition of unity and the knot vector \( T \) gives a subdivision of the interval \([0, 1]\) into compact subsets.

One defines recursively

\[
B_{i,0}(t) = \begin{cases} 
1 & t_i \leq t \leq t_{i+1} \\
0 & \text{otherwise}
\end{cases}
\]

and

\[
B_{i,m}(t) = \frac{t - t_i}{t_{i+m} - t_i} \cdot B_{i,m-1}(t) + \frac{t_{i+m+1} - t}{t_{i+m+1} - t_{i+1}} \cdot B_{i+1,m-1}(t)
\]

It is practical to use repeated knot values \( t_i = t_{i+1} = \cdots = t_{i+k} \). In this case we will get division by zero in the denominators \( \frac{t - t_i}{t_{i+k} - t_i} = \frac{0}{0} \). We will treat these degenerate denominators as 0 in the above formula, that is we will set \( \frac{t - t_i}{t_{i+k} - t_i} = 0 \) whenever \( t_{i+k} = t_k \).

**Knot vector** We use a uniform non-periodic knot vector for our purposes. Given the degree \( m \) and the number of control points \( n + 1 \), we define

\[
t_i = \begin{cases} 
0 & \text{if } 0 \leq i < m \\
i - m + 1 & \text{if } m \leq i \leq n \\
1 & \text{if } n \leq i \leq n + m
\end{cases}
\]

which gives the following structure to the knot vector

\[
0, 0, \ldots, 0, \frac{1}{n - m + 2}, \frac{2}{n - m + 2}, \ldots, \frac{n}{n - m + 2}, 1, 1, \ldots, 1
\]

For example, for \( n = 5 \) and \( m = 3 \), one get the knot vector

\[
T = \left[ 0, 0, 0, \frac{1}{4}, \frac{2}{4}, \frac{3}{4}, 1, 1, 1 \right].
\]

The use of this knot structure ensures that the B-splines that we construct pass through the initial control point \( P_0 \) and endpoint \( P_n \). The number of entries of \( T \) is \( n + m + 1 \).
Bibliography


