Smoothing of a curve in a plane

Vyhlazování křivek v rovině

Bachelor thesis
Zadání bakalářské práce

Lukáš Rapant

Student:
Studijní program: B2647 Informační a komunikační technologie
Studijní obor: 1103R031 Výpočetní matematika
Téma: Vyhlasování křivek v rovině
Smoothening of the curves in the plain

Zásady pro vypracování:
Metodou Bézierovou a metodou splajnou se v rovině vyhlažují po částech lineárně křivky, které vznikly lineárním spojením sousedních bodů (v rovině) naměřených hodnot. Cílem práce je porovnat výsledky procesů vyhlasování křivek.
Práce by měla mít tyto části:
- Definice pojmů
- Vyhlasování křivek - Béziere
- Vyhlasování křivek - Spline
- Kvalitativní srovnání výsledků

Seznam doporučené odborné literatury:
http://en.wikipedia.org/wiki/B%C3%A9zier_curve
http://en.wikipedia.org/wiki/Interpolation
dle pokynů vedoucího diplomové práce

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: RNDr. Marek Lampart, Ph.D.

Datum zadání: 30.11.2008
Datum odevzdání: 07.05.2009

prof. RNDr. Zdeněk Dostál, DSc.
vedoucí katedry

prof. Ing. Ivo Vondrák, CSc.
děkan fakulty
I declare, that I have worked up this bachelor thesis myself. I have presented all References that I used.

Ostrava 7th May 2009
I would like to express gratitude to RNDr. Marek Lampart, Ph.D. for his help. This thesis would have never existed without his help.
Abstrakt
Tato práce se zabývá vyhlazováním dat získaných z měření nitroděložního tlaku. Jejím cílem je vyhledat data, která jsou narušená neočekávanými událostmi. Použijeme k tomu čtyři metody: Interpolaci pomocí Spline funkce, Bézierovu křivku, Klouzavý průměr a Vyhlazování těžišti.

Klíčová slova: spline, Bézierova křivka, klouzavý průměr, vyhlazování těžišti, filtrace dat

Abstract
This work deals with smoothing data from measuring of uterine pressure. Its goal is to smoothen the data which are disrupted by some unexpected events. We are using four methods for smoothing: Spline interpolation, Bézier curve, Moving average and Smoothing by gravity centres.

Keywords: Spline, Bézier curve, Moving average, Smoothing by gravity centres, data filtration
List of Used Abbreviations and Symbols

- \( \mathbb{R} \) – the set of all real numbers
- \( f(x) \) – a real function of a real variable \( x \)
- \( \int f(x) \, dx \) – an integral of the function \( f \) with respect to \( x \)
- \( f^{(i)} \) – \( i^{\text{th}} \) derivative of the function \( f \)
- \( \binom{n}{i} \) – binomial coefficient
- \( C^r \) – the set of all functions with continuous derivative up to degree \( i \)
## Contents

1 Preface 4

2 Spline 5  
  2.1 Introduction 5  
  2.2 Definition and notation 5  
  2.3 Implementation 12

3 Bézier curve 15  
  3.1 Introduction 15  
  3.2 Definition and notation 15  
  3.3 Implementation 18

4 Moving average 20  
  4.1 Introduction 20  
  4.2 Definition and notation 20  
  4.3 Implementation 22

5 Smoothing by gravity centres 23  
  5.1 Introduction 23  
  5.2 Definition and notation 23  
  5.3 Implementation 26

6 Comparison of results 29

7 Conclusion 34

8 References 35

Appendix 36

A List of Appendices 36
# List of Figures

1. Example of cubic spline ........................................... 6
2. Cubic spline from example 2.1 .................................... 12
3. Example of Bézier curve .......................................... 16
4. Bézier curve from example 3.1 .................................... 18
5. Result from example 4.1 ........................................... 21
6. Input data ............................................................ 24
7. Input data and data filtered by algorithm $G_1$ .............. 24
8. Data filtered by algorithm $G_1$ and data filtered by algorithm $G_2$ .............. 25
9. Data filtered by algorithms $G_2$ and $G_3$ .................... 25
10. Input data and data filtered by all three algorithms .......... 26
11. Testing data ........................................................ 29
12. Testing data smoothed by the moving average .............. 30
13. Testing data smoothed by the gravity centres ............... 30
14. Comparison of moving average and smoothing by centres of gravity .............. 31
15. Testing data smoothed by the Bézier curve .................. 32
16. Testing data smoothed by the spline .......................... 32
17. Detail of spline interpolation ................................. 33
List of Source Codes

1. Spline solver *m*-file .............................................................. 12
2. Spline display *m*-file .............................................................. 14
3. Bézier curve solver *m*-file ...................................................... 18
4. Simple moving average curve filter *m*-file ................................. 22
5. Smoothing by a centres of gravity *m*-file ................................... 26
1 Preface

This work is motivated by research of prenatal care which ran in cooperation with Thomas Bata University of Zlín and regional hospital in Zlín. The main aim is to design a “model” for data smoothing. These data were got by measuring uterine pressure by a special body belt in periodic intervals. Unfortunately, during a measurement some unpredictable effects appear as a result of baby movement or some other mother’s body movements. We use four methods to clear these effects out of a data: Spline interpolation, Bézier curve, Moving average and Smoothing by gravity centres. In conclusion we will do comparison and evaluation of these methods.
2 Spline

2.1 Introduction

The first mathematical reference to splines is the 1946 paper by Schoenberg (see e.g. [12]). It was the first place that the word “spline” was given in connection with smooth, piecewise polynomial approximation. The ideas of splines come from the aircraft and shipbuilding industries and the technique called “lofting”. During World War II British aircraft industry used lofting to construct templates for airplanes by passing thin wooden strips (called “splines”) through points laid out on the floor of a large design loft. This technique was borrowed from ship-hull design. For years the practice of ship design had employed models to design in the small and then re-plotting key points of a loft to larger scale using a graph paper. The thin wooden strips provided an interpolation of the key points into smooth curves. The strips would be held in place at discrete points and between these points would assume shapes of minimum strain energy. Next step to spline was a rise of “conic lofting”, which used conic sections to model the position of the curve between the ducks. Conic lofting was replaced by what we would call splines in the early 1960s based on work by J. C. Ferguson at Boeing and later by M.A. Sabin at British Aircraft Corporation. Today the splines are also commonly used in computer graphics. The word “spline” itself was originally an East Anglican dialect word.

2.2 Definition and notation

First of all, we will focus on the polynomial case. In this case, a spline is a piecewise polynomial function. Let us denote this function by $f$ and this function takes values from the interval $[x_0, x_n]$ and maps them to $\mathbb{R}$. As we want $f$ to be defined piecewise, we decompose $[x_0, x_n]$ into $n$ disjoint subintervals $[x_i, x_{i+1}]$ where

$$x_0 \leq x_1 \leq x_2 \ldots \leq x_{n-2} \leq x_{n-1} \leq x_n.$$ 

This will give us $n$ points called knots. Then $x = (x_0, \ldots, x_n)$ is a knot vector for the spline. If these knots are distributed equidistantly in the interval, the spline is uniform, otherwise it is non-uniform. On each of these subintervals we define polynomial $\varphi_i$:

$$\varphi_i : [x_i, x_{i+1}] \rightarrow \mathbb{R}.$$ 

If the polynomial pieces on subintervals $x_0 \leq x_1 \leq x_2 \ldots \leq x_{n-2} \leq x_{n-1} \leq x_n$ have maximum degree of $k$, then the spline is called to be of degree of $k$. Most simple spline is of degree of 0. It is called a step function. The next most simple spline has degree 1. It is also called a linear spline. A closed linear spline in the plane (i.e. linear spline which begins and ends in the same point) is called a polygon. Most common type of spline is a cubic spline and we will define it later.

To construct spline function we also need boundary conditions. Two types of boundary conditions are clamped boundary conditions and free boundary conditions (both are defined in Definition 2.2). In the case of free boundary conditions, graph takes shape that a long
flexible line would assume if forced to go through each of points. Clamped conditions
give more accurate approximation but require derivative values at the endpoints or their
accurate approximations.

Let us now talk about cubic spline. It is noteworthy that any cubic spline belongs into
$C^2$ (see e.g. [4]). We denote, for simplicity, a cubic spline by:

$$\varphi_i(x) = a_i + b_i \cdot x + c_i \cdot x^2 + d_i \cdot x^3.$$ 

Spline interpolation is not absolutely accurate and has some numerical error. The
error is defined in Theorem 2.3 (for more information about polynomial interpolation
errors see [3]).

**Definition 2.1** Let us denote by $f_i(x_i)$ a value of the function $f_i$ in the knot $x_i$, $z_i = f''_i(x_i)$
second derivative in the knot $x_i$ and $h_i = x_{i+1} - x_i$. The cubic spline $f_i(x)$ is then defined by
formula:

$$f_i(x) = \frac{z_i(x_{i+1} - x)^3}{6h_i} + \frac{z_{i+1}(x-x_i)^3}{6h_i} + \left[ \frac{f(x_i)-z_i \cdot h_i}{h_i} \cdot (x_{i+1} - x) \right] + \left[ \frac{f(x_{i-1})-z_{i+1} \cdot h_i}{h_i} \cdot (x-x_i) \right].$$

![Figure 1: Example of cubic spline](image)
This formula is constructed by linear splines defined by formula:

\[ f''_i(x) = \frac{z_i(x_{i+1} - x)}{h_i} + \frac{z_{i+1}(x - x_i)}{h_i}. \]

From this formula we get expression for the natural cubic spline by integrating twice. It introduces two constants of integration, and the result can be manipulated so that it has the form:

\[ f_i(x) = \frac{z_i(x_{i+1} - x)^3}{6h_i} + \frac{z_{i+1}(x - x_i)^3}{6h_i} + p_i(x_{i+1} - x) + q_i(x - x_i). \]

Substituting \( x_i \) and \( x_{i+1} \) into previous equation and using the values \( y_i = f_i(x_i) \) and \( y_{i+1} = f_i(x_{i+1}) \) yields the following equations that involve \( p_i \) and \( q_i \), respectively:

\[ \begin{align*}
y_i &= \frac{z_i h_i^2}{6} + p_i h_i, \\
y_{i+1} &= \frac{z_{i+1} h_{i+1}^2}{6} + q_i h_i.
\end{align*} \]

By solving these equations we get the formula for cubic spline. However, for practical computing we need to know values of \( z_i \). We get them by using a derivative of \( f_i(x) \):

\[ f'_i(x) = -\frac{z_i(x_{i+1} - x)^2}{2h_i} + \frac{z_{i+1}(x - x_i)^2}{2h_i} - \left( \frac{f(x_i)}{h_i} - \frac{z_i h_i}{6} \right) + \left( \frac{f(x_{i-1})}{h_i} - \frac{z_{i+1} h_{i+1}}{6} \right). \]

By evaluating it in \( x_i \) we get

\[ f'_i(x_i) = -\frac{z_i h_i}{3} - \frac{z_{i+1} h_i}{6} + d_i, \]

where

\[ d_i = \frac{f(x_{i+1}) - f(x_i)}{h_i}. \]

Now, we replace \( i \) by \( i - 1 \) in the formula for derivative \( f'_i(x) \) and again evaluate it in \( x_i \):

\[ f'_{i-1}(x_i) = -\frac{z_i h_{i-1}}{3} - \frac{z_{i-1} h_{i-1}}{6} + d_{i-1}. \]

Now, we use continuity of a cubic spline in \( C^2 \) to define important relationship between \( z_{i-1}, z_i \) and \( z_{i+1} \):

\[ h_{i-1} z_{i-1} + 2(h_{i-1} + h_i) z_i + h_i z_{i+1} = 6(d_i - d_{i-1}) \text{ for } i = 1, 2, ..., n - 1. \]

This equation enables us to create a linear system for solving a spline. However, this system is underdetermined as it has \( n - 1 \) linear equations involving \( n + 1 \) variables. This
is solved by boundary conditions (see Definition 2.2). Let us consider that values of \( z_0 \) and \( z_n \) are given and so \( h_0 z_0 \) and \( h_{n-1} z_n \) can be computed. Then first and last equation of the system are

\[
2 (h_0 + h_1) z_1 + h_1 z_2 = 6 (d_2 - d_1) - h_0 z_0, \quad (2)
\]

\[
h_{n-2} z_{n-2} + 2 (h_{n-2} + h_{n-1}) z_{n-1} = 6 (d_i - d_{i-1}) - h_{n-1} z_n. \quad (3)
\]

By now, we have all equations needed for solving linear system defining the spline.

**Definition 2.2** Let us denote by \( f_i(x) \) a cubic spline defined on the interval \([x_0, x_n]\). The cubic spline \( f_i(x) \) then have boundary conditions \( z_0 = f''(x_0) \) and \( z_n = f''(x_n) \) which represent values of the function at the end points. These boundary conditions are either clamped boundary conditions defined by:

\[
z_0 = \frac{3}{h_0} \cdot (d_0 - f'(x_0)) - \frac{z_1}{2},
\]

\[
z_n = \frac{3}{h_{n-1}} \cdot (f'(x_n) - d_{n-1}) - \frac{z_{n-1}}{2},
\]

or free boundary conditions defined by:

\[
z_0 = 0,
\]

\[
z_n = 0.
\]

Let us note that the proofs of the following theorems are well known. We include them for completeness.

**Theorem 2.1** If the function \( f \) is defined on \([x_0, x_n]\), then there exists a unique cubic spline with the clamped boundary conditions.

**Proof.** We are going to solve the linear system derived from equations 1, 2 and 3:

\[
\begin{pmatrix}
\frac{3}{2} h_0 + 2 h_1 \\
h_1 \\
h_1 \\
h_1 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0
\end{pmatrix}
\begin{pmatrix}
z_1 \\
z_2 \\
z_i \\
z_{i+1} \\
z_{n-2} \\
z_{n-1} \\
z_n \\
z_n \\
z_n \\
z_n \\
z_n \\
z_n
\end{pmatrix}
= \begin{pmatrix}
(3/2) h_0 + 2 h_1 \\
h_1 \\
h_1 \\
h_1 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0
\end{pmatrix}
\begin{pmatrix}
u_1 \\
u_2 \\
u_2 \\
u_2 \\
u_1 \\
u_1 \\
u_1 \\
u_1 \\
u_1 \\
u_1 \\
u_1 \\
u_1
\end{pmatrix}
\]

where \( z_i = f''(x_i) \), \( h_i = x_i - x_{i-1} \) and \( u_i = 6 (d_i - d_{i-1}) \).

Then the matrix \( A \) of this system \( Az = B \) is of \((n + 1)\)-by-\((n + 1)\) type:

\[
A = \begin{pmatrix}
(3/2) h_0 + 2 h_1 & h_1 & 0 & 0 & \cdots & 0 & 0 & 0 \\
h_1 & 2 (h_1 + h_2) & h_2 & 0 & \cdots & 0 & 0 & 0 \\
0 & h_2 & 2 (h_2 + h_3) & h_3 & \cdots & 0 & 0 & 0 \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
0 & 0 & 0 & 0 & \cdots & h_{n-3} & 2 (h_{n-3} + h_{n-2}) & h_{n-2} \\
0 & 0 & 0 & 0 & \cdots & 0 & h_{n-2} & (2 h_{n-2} + 3/2 h_{n-1})
\end{pmatrix}
\]
and

$$B = \begin{bmatrix}
  u_1 - 3(d_0 - f'(x_0)) \\
  u_2 \\
  u_3 \\
  \vdots \\
  u_{n-2} \\
  u_{n-1} - 3(f'(x_n) - d_{n-1})
\end{bmatrix}.$$  

From matrix $A$ it is obvious that it is strictly diagonally dominant. From the Gershgorin Theorem (see e.g. [9]) we know that every strictly diagonally dominant matrix is also regular. As a matrix is regular, the linear system must have unique solution for $f''(x_i)$ for $i = 1, 2, 3, \ldots, n - 1$.

**Theorem 2.2** If the function $f$ is defined on $[a, b]$, then there exists a unique cubic spline with the free boundary conditions called the natural spline.

**Proof.** We are going to solve the linear system created from equations 1, 2 and 3:

$$2 (h_0 + h_1) z_1 + h_1 z_2 = u_1,$$

$$h_{i-1} z_{i-1} + 2 (h_{i-1} + h_i) z_i + h_i z_{i+1} = u_i \text{ for } i = 2, 3, \ldots, n - 2,$$

$$h_{n-2} z_{n-2} + 2 (h_{n-2} + h_{n-1}) z_{n-1} = u_{n-1},$$

where $z_i = f''(x_i)$, $h_i = x_i - x_{i-1}$ and $u_i = 6(d_i - d_{i-1})$.

Then the matrix $A$ of this system $Az = B$ is of $(n + 1)$-by-$(n + 1)$ type:

$$A = \begin{bmatrix}
  2 (h_0 + h_1) & h_1 & 0 & 0 & \ldots & 0 & 0 & 0 \\
  h_1 & 2 (h_1 + h_2) & h_2 & 0 & \ldots & 0 & 0 & 0 \\
  0 & h_2 & 2 (h_2 + h_3) & h_3 & \ldots & 0 & 0 & 0 \\
  \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
  0 & 0 & 0 & 0 & \ldots & h_{n-3} & 2 (h_{n-3} + h_{n-2}) & h_{n-2} \\
  0 & 0 & 0 & 0 & \ldots & 0 & h_{n-2} & 2 (h_{n-2} + h_{n-1})
\end{bmatrix}.$$  

and

$$B = \begin{bmatrix}
  u_1 \\
  u_2 \\
  u_3 \\
  \vdots \\
  u_{n-2} \\
  u_{n-1}
\end{bmatrix}.$$  

To the end it is enough to follow ideas from the previous proof.
Theorem 2.3 Let \( I \) be an interval containing the \( n + 1 \) interpolating points \( x_0, x_1, x_2, ..., x_n \). Let \( f(x) \) be a continuous function and have continuous derivatives of order \( n + 1 \) for all \( x \) in \( I \). Let \( p(x) \) be a polynomial which interpolates at the points \( x_0, x_1, x_2, ..., x_n \). Then at any point \( x \) on \( I \) the following equation holds:

\[
f(x) - p(x) = \frac{\psi(x)f^{(n+1)}(\xi)}{(n + 1)!}
\]

where

\[
\psi(x) = (x - x_0)(x - x_1)(x - x_2)...(x - x_n)
\]

and \( \xi \) is some point on the interval \( I \).

Proof. There are two cases: \( x \) is the knot \( x_i \) or \( x \) is not the knot. In the first case \( f(x) = p(x) \) and \( \psi(x) = 0 \) so both sides of equation vanish. In the case that \( x \) is not the knot we define auxiliary function:

\[
\phi(s) = f(s) - p(s) - g(x)\psi(s)
\]

where

\[
g(x) = \frac{f(x) - p(x)}{\psi(x)}.
\]

If \( s = x_i \) for \( i = 0, 1, 2, ..., n \), then the equation will vanish just like with the knot. At a point \( s = x \) distinct from the \( x_i \) it is

\[
\phi(x) = f(x) - p(x) - g(x)\psi(x) = 0,
\]

because of the definition of \( g(x) \). Then the function \( \phi(s) \) has \( n + 2 \) zeros in the interval \( I \). By Rolle’s theorem (see e.g. [1] [2]), \( \phi'(s) \) have \( n + 1 \) zeros in \( I \) and \( \phi''(s) \) must have \( n \) zeros in \( I \) and finally \( \phi^{(n+1)}(s) \) must have at least one zero in \( I \). Let \( \xi \) be one such zero. Now, \( \phi(s) \) is differentiated \( n + 1 \) times and then we set \( s = \xi \), since the \( (n + 1)st \) derivate of \( p(x) \)disappears. We get

\[
\phi^{(n+1)}(\xi) = 0 = f^{(n+1)}(\xi) - g(n + 1)!. 
\]

To the end, if we replace \( g(x) \) for \( (f(x) - p(x))/\psi(x) \), we get

\[
f(x) - p(x) = \frac{\psi(x)f^{(n+1)}(\xi)}{(n + 1)!}.
\]

Example 2.1
Find the natural cubic spline that passes through \((0, 0.0), (1, 0.5), (2, 2.0), \) and \((3, 1.5)\) with the free boundary conditions \( f''(x_0) = f''(x_3) = 0 \).
First of all we compute $\{h_i\}$, $\{d_i\}$ and $\{u_i\}$ using these formulas:

\[
h_i = x_i - x_{i-1},
\]

\[
d_i = (f(x_{i+1}) - f(x_i)) / h_i,
\]

\[
u_i = 6 \left( d_i - d_{i-1} \right).
\]

From the formulas we get

\[
0 = h_1 = h_2 = 1,
\]

\[
0 = 0.5 - 0/1 = 0.5,
\]

\[
1 = 2 - 0.5/1 = 1.5,
\]

\[
2 = 1.5 - 2/1 = -0.5,
\]

\[
1 = 6 (1.5 - 0.5) = 6,
\]

\[
2 = 6 (-0.5 - 1.5) = -12.
\]

Now we use these values to solve linear system 1:

\[
2 \left( 1 + 1 \right) z_1 + z_2 = 6,
\]

\[
z_1 + 2 \left( 1 + 1 \right) z_2 = -12.
\]

The matrix of this linear system is

\[
\begin{bmatrix}
4 & 1 \\
1 & 4
\end{bmatrix}
\begin{bmatrix}
z_1 \\
z_2
\end{bmatrix}
\begin{bmatrix}
6 \\
-12
\end{bmatrix}.
\]

From this matrix we get the solution:

\[
z_1 = 2.4,
\]

\[
z_2 = -3.6,
\]

\[
z_0 = f''(x_0) = 0,
\]

\[
z_3 = f''(x_3) = 0.
\]

Now, we use the following formulas to get the spline function:

\[
f_i(x) = ((d_i \cdot (x - x_i) + c_i) \cdot (x - x_i) + b_i) \cdot (x - x_i) + a_i,
\]

where

\[
a_i = f(x_i),
\]

\[
b_i = d_i - h_i \cdot (2z_i + z_{i+1}) / 6,
\]

\[
c_i = z_i / 2,
\]

\[
d_i = z_{i+1} - z_i / (6h_i).
\]

Consequently, we have

\[
f_0(x) = 0.4x^3 + 0.1x \quad \text{for} \ 0 \leq x \leq 1,
\]

\[
f_1(x) = - (x - 1)^3 + 1.2 (x - 1)^2 + 1.3 (x - 1) + 0.5 \quad \text{for} \ 1 \leq x \leq 2,
\]

\[
f_2(x) = 0.6 (x - 2)^3 - 1.8 (x - 2)^2 + 0.7 (x - 2) + 2.0 \quad \text{for} \ 2 \leq x \leq 3.
\]

The resulting spline is shown in the Figure 2.
2.3 Implementation

Spline curve smoothing is implemented in Matlab, because it already has implemented some functions needed for computing splines. It also can be used in other programming languages, which can import libraries from Matlab (i.e. Java). Program solves splines basically like it is shown in the example 2.1. Its input parameters are vectors of $x$ and $y$ coordinates of a points on the curve. It checks input parameters. Then it generates spline in piecewise polynomial form using divided difference polynomials and checks type of spline. After that it makes linear system and solves it. Then it constructs spline by using values given by linear system. Output is piecewise polynomial representing the spline curve. For visualization we need to evaluate this polynomial. This is done in second $m$-file by the function `ppval` which evaluates piecewise polynomial at the points of vector $x$. Output of this function is a matrix with the points representing resulting spline. Here are source codes of both $m$-files:

```matlab
function output = spline(x,y)
%Cubic spline data interpolation.
% output = SPLINE(X,Y) provides the piecewise polynomial form of the
% cubic spline for all values of x and y. It is used in conjunction
% with the evaluator PPVAL.
% x must be a vector.
% If y is a vector, then y(i) is taken as the value to be matched at y(i),
% so y must be of the same length as X.
% If y is a matrix, then y(:,...,i) is taken as the value to
```

Figure 2: Cubic spline from example 2.1
% be matched at x(i), hence the last dimension of y must equal length(y)
%

cs=[];
% generate the cubic spline in ppform
dd = ones(yd,1); dx = diff(x); divdif = diff(y,2)/dx(dd,:);
if n==2
% the spline is a straight line
  if isempty(endslopes)
    pp=mkpp(x,[divdif y(:,1)],sizey);
  % the spline is the cubic polynomial
  else
    pp = pwch(x,y,endslopes,dx,divdif); pp.dim = sizey;
  end
% the spline is a parabola
elseif n==3&&isempty(endslopes)
  y(:,3)=diff(divdif)’/(x(3)-x(1));
  y(:,2)=y(:,2)-y(:,3)*dx(1);
  pp = mkpp(x([1,3]),y(:,[3 2 1]),sizey);
else
  b=zeros(yd,n);
  b(:,1:3)=3*(dx(dd,2:n-1),*divdif(:,1:n-2)+dx(dd,1:n-2),*divdif(:,2:n-1));
  if isempty(endslopes)
    x31=x(3)-x(1);xn=x(n)-x(n-2);
    b(:,1)=((dx(1)+2*x31)*dx(2)+divdif(:,1)+dx(1)’*divdif(:,2))’/x31;
    b(:,n)=...    
    (dx(n-1)’*divdif(:,n-2)+(2*xn+dx(n-1))+dx(n-2)*divdif(:,n-1))/xn;
  else
    x31 = 0; xn = 0; b(:,[1 n]) = dx(dd,[2 n-2],*:endslopes;
  end
dxt = dx(:,1);
c = spdiags([ [x31,dxt(1:n-2),0] ...  
  [dxt(2),dxt(2:n-1)+dxt(1:n-2)];dxt(n-2)] ...  
  [0,dxt(2:n-1);xn],[-1 0 1],n,n);

% sparse linear equation solution for the slopes
mmdflag = spparms(’autommd’);
spparms(’autommd’,0);
s=b/c;
spparms(’autommd’,mmdflag);

% construct piecewise cubic polynomial
% to values and computed slopes
pp = pwch(x,y,s,dx,divdif ); pp.dim = sizey;
end
output = pp;
end

Source code 1: Spline solver m-file
function output = spld(A)
% Cubic spline evaluation.
% output = spl(A) provides the matrix with x and y values of points defining Cubic spline.
% A is matrix containing x and y values for points which are interpolated by spline

tic
% check matrix dimensions
	t = size(A);
	if 
		t(2) ~= 2
		error('Wrong dimension of matrix with input points')
end
% prepare parameters for Spline solver
	t = length(A);
	x = A(:, 1)';
	y = A(:, 2);
% Spline solver

cs = spline(x,[0 y' 0]);
% vector of points in which the spline is evaluated
	xx = 0:0.01:A(t);
% evaluate spline

output(:, 1) = xx;
output(:, 2) = ppval(cs, xx);
% plot spline in matlab

plot(output(:, 1), output(:, 2));
toc

Source code 2: Spline display m-file
3 Bézier curve

3.1 Introduction

Bézier curves are named after Dr. Pierre Bézier. Bézier was an engineer with the Renault car company and in the early 1960’s developed a curve formulation which would lend itself to shape design. They were used to design shapes of the cars. However, they were not invented by him but by Paul de Casteljau in 1959 (see e.g. [11]). They are named after Bézier because he made them widely known. Nowadays they are mostly used for computer graphics. In vector graphics, Bézier curves are an important tool used to model smooth curves. They are used in programs such as Inkscape, Adobe Illustrator, Adobe Photoshop, and GIMP. Bézier curves are also used in animation as a tool to control motion in applications such as Adobe Flash, Adobe After Effects, and Autodesk 3ds max.

3.2 Definition and notation

Let us show how Bézier curve works on center of mass of a four point masses. Normally the center of mass of these four point masses is given by the equation:

\[ P = \frac{m_0 P_0 + m_1 P_1 + m_2 P_2 + m_3 P_3}{m_0 + m_1 + m_2 + m_3} \]

In the next step let us say that masses are not constant but are defined by some functions of parameter \( t \). For example \( m_0 = (1 - t)^3 \), \( m_1 = 3t (1 - t)^2 \), \( m_2 = 3t^2 (1 - t) \) and \( m_3 = t^3 \). Now, let \( t \) be a parameter from a closed interval \([0, 1]\). The centre of mass shall change accordingly to changing \( t \) and draw Bézier curve. Also note that for any value of \( t \) it holds \( m_0 + m_1 + m_2 + m_3 = 0 \). We can define equation for Bézier curve as \( P = m_0 P_0 + m_1 P_1 + m_2 P_2 + m_3 P_3 \). Another important property is when \( t = 0 \), \( m_0 = 1 \), \( m_1 = m_2 = m_3 = 0 \) curve passes through \( P_0 \) and when \( t = 1 \), \( m_0 = m_1 = m_2 = 0 \), \( m_3 = 1 \) curve passes through \( P_3 \). This means that curve has two fixed points (i.e. points through which curve always passes) for \( t = 0 \) and \( t = 1 \) and is therefore good for describing various shapes.

In case of Bézier curves variable masses \( m_i \) are normally called blending functions and their locations \( P_i \) are known as control points or Bézier points. If straight lines between adjacent control points is drawn, the resulting figure is known as a control polygon.

Each Bézier curve has some degree \( n \) (degree of polynomials which define blending function). Bézier curve has \( n + 1 \) control points. It is straight line between two points. Quadratic curve is defined by three points. Its shape is parabolic segment. In the introductory example cubic curve is used. It is defined by four points and its polynomials are cubic (i.e. \( t^3 \)). Its shape is also some kind of paraboloid. It is used by modern imaging systems like PostScript, Asymptote and Metafont for drawing curved shapes.

Blending functions, in the case of Bézier curves, are known as Bernstein polynomials (for more details on Bernstein polynomials see e.g. [5]). They are denoted as \( B_i^n (t) \), where

\[ B_i^n (t) = \binom{n}{i} (1 - t)^{n-i} t^i, i = 0, 1, ..., n. \]
In our example, \( n = 3 \) and \( m_0 = B^3_0 = (1-t)^3 \), \( m_1 = B^3_1 = 3t(1-t)^2 \), \( m_2 = B^3_2 = 3t^2(1-t) \) and \( m_3 = B^3_3 = t^3 \).

**Definition 3.1** Let \( P_i \) be a set of all control points and \( B_i^n(t) \) be a Bernstein polynomial of degree \( n \). Then the Bézier curve \( P(t) \) is defined by:

\[
P(t) = \sum_{i=0}^{n} B_i^n(t) P_i.
\]

Figure 3: Example of Bézier curve

In the following Theorem we prove that Bézier curve is differentiable. Consequently we show that Bézier curve is continuous.

**Theorem 3.1** Bézier curve has derivatives of all orders on the interval \([0, 1]\).

**Proof.** The derivative of \( P(t) \), with respect to \( t \), is

\[
P'(t) = \frac{d}{dt} \sum_{i=0}^{n} B_i^n(t) P_i = \sum_{i=0}^{n} \frac{d}{dt} B_i^n(t) = \sum_{i=0}^{n} n \left( B_{i-1}^{n-1}(t) - B_i^{n-1}(t) \right)
\]

where the last equality is given by formula for derivation of Bernstein polynomial (see e.g. [5] for full deduction). Setting \( t = 0 \) and substituting \( B_i^n(0) = 1 \) for \( i = 0 \) and \( B_i^n(0) = 0 \) for \( i \geq 1 \) into the right side of the expression for \( P'(t) \) we get

\[
P'(0) = \sum_{i=0}^{n} P_i n \left( B_{i-1}^{n-1}(0) - B_i^{n-1}(0) \right) = n \left( P_1 - P_0 \right).
\]
Similarly, we can make

\[ P'(1) = \sum_{i=0}^{n} P_i \cdot n \left( B_{i-1}^{n-1}(1) - B_i^{n-1}(1) \right) = n (P_N - P_{N-1}). \]

By proving that Bézier curve defined on closed interval [0, 1] has derivatives in the end points and using well known fact that each curve which has derivatives in the end points of its interval has also derivatives of entire curve, we assert that Bézier curve has derivatives of all orders on interval [0, 1].

**Remark 3.1** Another way to prove the Theorem 3.1 is to use a fact that \( P(t) \) is polynomial function. Then the assertion is obvious.

**Corollary 3.1** *Any Bézier curve is continuous.*

**Proof.** The assertion follows from Theorem 3.1 and well known fact that each differentiable function is also continuous (see e.g. [1] [2]).

**Example 3.1**
Find the Bézier curve which has the control points (2, 2), (1, 1.5), (3.5, 0), and (4, 1).

First of all, we substitute the \( x \) and \( y \) coordinates of the control points and \( n = 3 \) into following formulas:

\[ x(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i x_i, \]

\[ y(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i y_i. \]

This gives us functions for the \( x \) and \( y \) coordinates:

\[ x(t) = 2 (1-t)^3 + 3t (1-t)^2 + 10.5 t^2 (1-t) + 4t^3, \]

\[ y(t) = 2 (1-t)^3 + 4.5t (1-t)^2 + t^3. \]

This can be simplified into formula:

\[ P(t) = \left( 2 - 3t + 10.5 t^2 - 5.5t^3, 2 - 1.5t - 3t^2 + 3.5t^3 \right) \text{ where } 0 \leq t \leq 1. \]
3.3 Implementation

Bézier curve smoothing is implemented in Matlab as it already has some functions needed for computing Bézier curve from set of points (Bernstein polynomials for example). It can be also used in other programming languages, which can import libraries from Matlab (i.e. Java). Input of the function is matrix with control points. Program adjusts input parameter and substitute it into formula for the Bernstein polynomials. Then it substitutes given Bernstein polynomial into the formula for computing Bézier curve. Output is vector with points of Bézier curve. Here we include source code:

```matlab
function output = bezier(A)
    % Bézier curve data smoothing.
    % output = bezier(A,n) provides Bézier curve which is drawn from the points in the matrix A.
    % A must be a matrix of (x,2) type.
    % check matrix dimensions
    tic
    t = size(A);
    if t(2) ~= 2
        error('Wrong dimension of matrix with input points')
    end
    % adjusting of input parameters for formula for Bernstein pol.
    C = A';
    M = 1000;
    n = length(C(1,:));
```
a = 0:(1/(M−1)):1;
I = (0:(n−1))'*ones(1,M);
P = ones(n,1)*a;
%substitution into formula for Bernstein pol.
A = binopdf(I,n−1,P);
%substitution into formula for Bézier curve pol.
output = C*A;
%plotting of the Bézier curve in matlab
plot(output(:,1), output(:,2));
toc

Source code 3: Bézier curve solver m-file
4 Moving average

4.1 Introduction

Moving average was invented by an aerospace engineer J.M. Hurst in 1970 (see e.g. [10]). Hurst provided the mathematical details to support what he called his “price motion model”. At first it was used for stock price predictions and calculations. Today the moving averages are one of the most popular tools available for the technical analysis. They smooth data and allow to spot trends much easier.

4.2 Definition and notation

There are many kinds of the moving averages (for more information see e.g. [8]), but in our work we use only a simple moving average (SMA). SMA is the unweighted mean of the previous $n$ data points. For example, a 10-day simple moving average of closing price is the mean of the previous 10 day’s closing prices. Formula for this example is

$$SMA = \frac{P_n + P_{n-1} + P_{n-2} + \ldots + P_{n-9}}{10}$$

**Definition 4.1** Let $P$ be a set of data points and $n$ a natural number. Then we compute a simple moving average value of point $P_{i\text{SMA}}$ by formula:

$$P_{i\text{SMA}} = \frac{P_i + P_{i-1} + P_{i-2} + \ldots + P_{i-(n-1)}}{n}$$

The calculation is repeated for each price bar on the chart. As new value comes from the formula, the oldest one will be dropped. The averages are then joined to form a smooth curving line - the moving average line. There are various popular values for $n$, like 10 days, 40 days or 200 days. The period selected depends on the kind of movement one is concentrating on, such as short, intermediate or long term.

In all cases a moving average lags behind the latest data point, simply from the nature of its smoothing. An SMA can lag to an undesirable extent, and can be disproportionately influenced by old data points dropping out of the average. This means that SMA is good in following trends but not for getting precise information.

**Remark 4.1** Let us note that SMA is a piecewise smooth curve. We get it as linear interpolation of two neighboring points.

**Example 4.1**

We smooth curve defined by points $(1, 2), (2, 4), (3, 7), (4, 3), (5, 1), (6, 5), (7, 6)$ by $n = 3$ using simple moving average.

We take first three points and compute new value for point $(3, 7)$:

$$\frac{2 + 4 + 7}{3} = 4.33$$
So our new point is \((3, 4.33)\). Then we shall compute a value of fourth point using new point given in previous step:

\[
(4 + 4 + 3) / 3 = 4.66
\]

The value of our new point is \((4, 4.66)\). This will be done with all points. After that we get

<table>
<thead>
<tr>
<th>Original</th>
<th>SMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 2)</td>
<td></td>
</tr>
<tr>
<td>(2, 4)</td>
<td></td>
</tr>
<tr>
<td>(3, 7)</td>
<td>(3, 4.33)</td>
</tr>
<tr>
<td>(4, 3)</td>
<td>(4, 4.66)</td>
</tr>
<tr>
<td>(5, 1)</td>
<td>(5, 3.66)</td>
</tr>
<tr>
<td>(6, 5)</td>
<td>(6, 3)</td>
</tr>
<tr>
<td>(7, 6)</td>
<td>(7, 4)</td>
</tr>
</tbody>
</table>

In the right column there are new values of the points after doing SMA. Both curves (original is blue and new is green) are drawn in the Figure 5.

Figure 5: Result from example 4.1
4.3 Implementation

Simple moving average curve smoothing is implemented in Matlab because it has some crucial functions already implemented. The program is quite simple. It takes two arguments. The first is a matrix with control points and the second is number of points from which the mean is calculated. Then it creates empty vector and fills it using filter matlab function to compute SMA. Output is a matrix with computed values of SMA.

```matlab
function output = moving(A, n)
% Moving average data smoothing.
% x = moving(A,b) provides matrix defining curve smoothed by moving average
% which is drawn from the points in the matrix A.
% A must be a matrix of (x,2) type.
% n is a natural number representing number of points on which moving average is done and must not be grater than number of rows of matrix A

tic
% check matrix dimensions
if t(2)<2
    error('Wrong dimension of matrix with input points')
end
if t(1)<n
    error('Not enough points for moving average smoothing or number of points on which moving is done is too high')
end
% adjust input parameters
x=A(:,2)';
q=length(x);
b=ceil(n);
% create a vector of nans
output = nan(q,2);
% Simple moving average realized by filter function
m = filter(ones(1,b)/b, 1, x(1,:));

% Fill in the vector
for i = b:1:q
    output(i,1) = A(i,1);
    output(i,2) = m(i);
end
% plotting moving average in matlab
plot(output(:,1),output(:,2));
toc
```

Source code 4: Simple moving average curve filter m-file
5 Smoothing by gravity centres

5.1 Introduction

This method was recently developed in [6] and is currently used to filter data received from the measurements. The invented algorithm was based on idea of triangular center of gravity.

5.2 Definition and notation

This method is based on centres of gravity. Basically this method can be divided into three steps. In the first step method takes three points from a set of points on which is working and computes a centre of gravity. It is denoted as:

$$G_1 \left( \left[ x_{i+j}, y_{i+j} \right]_{j=0}^2 \right) = \left[ x_{i+1}, \frac{y_i + y_{i+1} + y_{i+2}}{3} \right].$$

Now we have new set of data points, which contains only one third of points from previous set. Now we apply next step, which is similar to the previous one with few differences. It takes six points and computes the centre of gravity from the first three. Then it compares the value of gravity with the following two values and creates new point by the next formula:

$$G_2 \left( \left[ x_{i+j}, y_{i+j} \right]_{j=0}^5 \right) = \begin{cases} [x_{i+1}, c_i] [x_{i+4}, c_i] & \text{if } y_{i+3}, y_{i+4} < c_i \text{ or } y_{i+3}, y_{i+4} > c_i \\ [x_{i+1}, c_i] [x_{i+4}, c_{i+3}] & \text{otherwise} \end{cases}$$

where $c_i = (y_i + y_{i+1} + y_{i+2})/3$. Again we have one third of points from previous set. The third step is very similar to the second one with the only difference. Where second step compares the value of gravity with the following two values, the third one compares to three next values. It is defined by formula:

$$G_3 \left( \left[ x_{i+j}, y_{i+j} \right]_{j=0}^5 \right) = \begin{cases} [x_{i+1}, c_i] [x_{i+5}, c_i] & \text{if } y_{i+3}, y_{i+4}, y_{i+5} < c_i \text{ or } y_{i+3}, y_{i+4}, y_{i+5} > c_i \\ [x_{i+1}, c_i] [x_{i+4}, c_{i+3}] & \text{otherwise} \end{cases}$$

where $c_i = (y_i + y_{i+1} + y_{i+2})/3$. By now we have exactly 1/27 of original data points creating smooth curve.

**Definition 5.1** Let $D$ be a set of data points and $G_1$, $G_2$ and $G_3$ three steps of the algorithm. Then by applying $G_1$, $G_2$ and $G_3$ on $D$ we shall get $D_G$ which is smoothed and filtered set of data points from $D$.

**Remark 5.1** Let us note that smoothing by gravity centres gives us a piecewise smooth curve. We get it as linear interpolation of two neighboring points.
Example 5.1
Smooth curve defined by 432 data points from real measurement. Input curve is drawn in the Figure 6.

First, we use algorithm $G_1$. Result is shown in the Figure 7. After applying algorithm $G_1$ we have one third of original points.

![Figure 6: Input data](image)

![Figure 7: Input data and data filtered by algorithm $G_1$](image)
On data we get from algorithm $G_1$ we apply algorithm $G_2$. The result is again shown in the Figure 8. By now we have only $1/9$ of original points.

![Figure 8: Data filtered by algorithm $G_1$ and data filtered by algorithm $G_2$](image)

Now, we use algorithm $G_3$ on data filtered by algorithms $G_1$ and $G_2$. After applying algorithm $G_3$ we have final curve smoothed by centres of gravity. It is shown in the Figure 9 and it contains only $1/27$ of original points.

![Figure 9: Data filtered by algorithms $G_2$ and $G_3$](image)
Input data and smoothened curve are compared in the Figure 10.

Figure 10: Input data and data filtered by all three algorithms

5.3 Implementation

Smoothing by centres of gravity is implemented in Matlab. It does not require any special functions but it is easier to compare it with other methods. It takes one argument which is vector with data points. Then the program works just like it is described in the previous part. Output is a matrix with smoothed and filtered data points.

```matlab
function output = gravcen( A )
%Centres of gravity data smoothing
% output = gravcen(A) provides matrix defining curve smoothed by centres of
% gravity which is computed from the points in the matrix A.
% A must be a matrix of (x,2) type and must have at least 27 rows.

tic
%check matrix dimensions
[t] = size(A);
if t(2)~=2
    error('Wrong_dimension_of_matrix_with_input_points')
end
if t(1)<27
    error('Not_enough_points_for_gravity_centres_smoothing')
end
```
%adjust input parameters
t = length(A);
te = ceil(t/3);
B = zeros(te,2);
b = 1;
%application of G1 algorithm
for i = 1 : 3 : t;
    B(b,1) = (A(i,1) + A(i+1,1) + A(i+2,1))/3;
    B(b,2) = (A(i,2) + A(i+1,2) + A(i+2,2))/3;
    b = b + 1;
    %check if next iteration is possible
    if (i+5 > t)
        break;
    end;
end;
%adjust output of G1
q = length(B);
if (B(q,1) == 0)
    B(q,:) = [];
end;
r = length(B);
A = B;
B = zeros(te,2);
b = 1;
%application of G2 algorithm
for i = 1 : 6 : t
    if (((A(i,1) + A(i+1,1) + A(i+2,1))/3) > A(i+3) && ((A(i,1) + A(i+1,1) + A(i+2,1))/3) > A(i+4) || ((A(i,1) + A(i+1,1) + A(i+2,1))/3) < A(i+3) && ((A(i,1) + A(i+1,1) + A(i+2,1))/3) < A(i+4))
        B(b,1) = (A(i,1) + A(i+1,1) + A(i+2,1))/3;
        B(b,2) = (A(i,2) + A(i+1,2) + A(i+2,2))/3;
        B(b+1,1) = (A(i+3,1) + A(i+4,1) + A(i+5,1))/3;
        B(b+1,2) = (A(i+3,2) + A(i+4,2) + A(i+5,2))/3;
    else
        B(b,1) = (A(i,1) + A(i+1,1) + A(i+2,1))/3;
        B(b,2) = (A(i,2) + A(i+1,2) + A(i+2,2))/3;
        B(b+1,1) = (A(i+3,1) + A(i+4,1) + A(i+5,1))/3;
        B(b+1,2) = (A(i+3,2) + A(i+4,2) + A(i+5,2))/3;
    end;
    b = b + 2;
    %check if next iteration is possible
    if (i+11 > r)
        break;
    end;
end;
%adjust output of G2
q = length(B);
if (B(q,1) == 0)
    B(q,:) = [];
end;
r = length(B);
A = B;
te = ceil(r/6);

B=zeros(te,2);
b=1;

%application of G3 algorithm
for i = 1 : 6 : t ;
    if (((A(i ,1)+A(i+1,1)+A(i+2,1))/3)>A(i+3)&&(A(i,1)+A(i+1,1)+A(i+2,1))/3)>A(i+4)&&(A(i,1)+A(i+1,1)+A(i+2,1))/3)>A(i+5)&&(A(i,1)+A(i+1,1)+A(i+2,1))/3)<A(i+3)&&(A(i,1)+A(i+1,1)+A(i+2,1))/3)<A(i+4)&&(A(i,1)+A(i+1,1)+A(i+2,1))/3)<A(i+5))
        B(b,1)=(A(i ,1)+A(i+1,1)+A(i+2,1))/3;
        B(b,2)=(A(i ,2)+A(i+1,2)+A(i+2,2))/3;
        B(b+1,1)=A(i+5,1);
        B(b+1,2)=(A(i,2)+A(i+1,2)+A(i+2,2))/3;
    else
        B(b,1)=(A(i ,1)+A(i+1,1)+A(i+2,1))/3;
        B(b,2)=(A(i ,2)+A(i+1,2)+A(i+2,2))/3;
        B(b+1,1)=(A(i+3,1)+A(i+4,1)+A(i+5,1))/3;
        B(b+1,2)=(A(i+3,2)+A(i+4,2)+A(i+5,2))/3;
    end;
    b=b+2;
end;

%check if next iteration is possible
if (i+11>r)
    break;
end;

%adjust output of G3
q=length(B);
if (B(q,1)==0)
    B(q,:) = [];
end;

%plot smoothed curve
plot(B(:,1) ,B(:,2) ) ;
output=B;
toc

Source code 5: Smoothing by a centres of gravity m-file
6 Comparison of results

Now, we compare results that we got by application of all four methods on a testing data. These methods were tested on computer with processor AMD Athlon 64 3500+ and 2 GB of physical memory. Real results of measurements represent the testing data. A pressure is measured every 0.1 second and our testing data contain 5137 such measurements. Many of these measurements are disrupted by some events. We need to clear these disruptions because we need to identify slower and steadier rises of pressure. These rises represent contractions. In the Figure 12 are the testing data plotted into a curve.

![Testing data](image)

Figure 11: Testing data

Our methods are divided into two categories: ones which work in real time and ones which need entire curve.

Moving average and Smoothing by gravity centres fall into the first category. Both methods can work in real time by taking measurements and immediately smoothing them. As it is mentioned in Remarks 4.1 and 5.1 both of them are also not smooth curves but linear interpolations of individual data points. It makes them very fast for computing. In the Figures 12 and 13 testing data are smoothed by both of methods. We have computed moving average from 100 points and we used algorithms described in chapters 2 and 3, respectively.
Figure 12: Testing data smoothed by the moving average

Figure 13: Testing data smoothed by the gravity centres
Results look very similar but there are some differences and each method has its advantages over the other. In time demandingness they are very similar: moving average needs 0.031308 second to smooth the data and gravity centres needs 0.035506 second. In a term of accuracy, moving average is better as the curve preserves all of its data points. Gravity centres have only 1/27 of its original data points. Because of this, curve smoothed by gravity centres look more “jagged” and loses some of its information. But in this case it is not a loss because still gravity centres are able to show peaks accurately enough. With this feature is connected big advantage of smoothing by gravity centres: this method is much less demanding on memory space for storing the data. Last notable difference is that moving average is slightly behind the centres of gravity. Its cause passes from Definition 4.1. Moving average is longer influenced by older data points then centres of gravity. Therefore it looses some of its accuracy but delay is so small that it does not influence it usefulness. This effect is shown in the Figure 14.

![Figure 14: Comparison of moving average and smoothing by centres of gravity](image)

In the second category there are methods which require entire curve for smoothing and cannot work in a real time. These methods are: Bézier curve and spline. Their most obvious common thing is that they need closed set of points to function as passes from Definitions 3.1 and 2.1. Although this fact disallows them to function as prime methods for smoothing the data from measuring but they are still useful. They can be used to smooth data later when the measurement is complete. These methods are also drawing complete curve so no other interpolation is needed. From features of the splines mentioned in part 2.2 and Corollary 3.1 we know that they are both continuous. This is their advantage against the other ones because they are loosing no information to linear interpolation. In the Figures 12 and 13 there are tested data smoothed by both of these methods and we used algorithms described in chapters 4 and 5, respectively.
Results from these two methods are very different. At the first sight it is evident that spline does not have eliminated any disrupting points. It is evident from Definition 2.1 that it passes through all points and therefore it cannot eliminate them. But it is still very
useful because it interpolates lines between points into smooth and continuous curve (see Figure 17). From the features of Bézier curve mentioned in part 3.2 we know that it only passes through start and end point of the curve which makes perfect tool for smoothing. In a term of time demandingness there is also a huge difference: spline needs only 0.622996 second for smoothing whereas Bézier curve needs 13.509466 seconds. This is caused by polynomial nature of Bézier curve as it needs to compute polynomials of very high degree and therefore is very demanding on computational power. This means that Bézier curve has optimal output (smooth and continuous) but is very demanding. Spline is more useful for secondary smoothing (for example data received from real time methods) as primary method filters disruptions and spline creates smooth and continuous curve.

Figure 17: Detail of spline interpolation
7 Conclusion

In this thesis we have used and tested four methods for a data smoothing. Bézier curve and spline proved to be unusable for real time smoothing but very useful for smoothing data after the measurement. Bézier curve presents the greatest smoothing but it is very demanding on a computational power. Spline on its own is almost useless but it is suitable to work in cooperation with some real time method by adding smoothness and continuity to it. Moving averages and smoothing by centres of gravity proved to be the best choice. They can work in a real time so they can process data from measuring at once. Smoothing by centres of gravity is slightly better because it is less demanding on memory and loss of accuracy is insignificant. So the best method for a data smoothing is combination of smoothing by centres of gravity with the spline. It gives us smoothed and continuous curve and is not demanding on memory or computational power.
8 References


A List of Appendices

CD with $m$-files