

Term paper

Sample title
for a report

Author: First name Last name
Matriculation No.: xxxxxxxx
Course of Studies: Mechanical Engineering

First examiner: Prof. Dr. Elmar Wings
Submission date: November 8, 2023

University of Applied Sciences Emden/Leer · Faculty of Technology ·
Mechanical Engineering Department
Constantiaplatz 4 · 26723 Emden · <http://www.hs-emden-leer.de>

Contents

Contents	i
List of Figures	iii
List of Tables	v
List of Listings	vii
Acronyms	ix
1. Hints	1
2. Bézier-Curves	3
2.1. Introduction	3
2.2. Allgemeine mathematische Beschreibung Bézier-Kurve	5
3. Verrundung mit einem Kreisbogen	9
3.1. Gleichungen	9
3.2. Bewertung	12
3.3. Examples	14
4. Maple files	17
4.1. Geometries with Maple	17
4.2. Geometry elements used	17
4.3. Building the data structure	18
4.4. Structure of a module	19
4.4.1. General structure of a module	21
4.4.2. Saving a module	23
4.4.3. Verwendung eines Moduls	23
4.4.4. Creating a Maple module	24
4.5. Functions of the modules	24
4.5.1. Module MPoint	24
4.5.2. Module MLine	25
4.5.3. Module MArc	26
4.5.4. module MBezier	26
4.5.5. Module MPolygon	27
4.5.6. module MGeoList	28
4.5.7. Module MHermiteProblem	28

4.5.8. Module MHermiteProblemSym	29
4.5.9. Module MConstant	30
4.5.10. Module MGeneralMath	30
4.5.11. Module Biarc	31
4.5.12. Module MBiarc	31
4.6. Programme Flowchart	32
4.6.1. Overall Flow	32
4.6.2. Verrundung der Kurve	32
5. Representation of Python Programs	35
6. First Chapter	39
7. CAGD	41
A. drawings with tikz	43
B. Criteria for a good \LaTeX project	49
Bibliography	51
Index	53

List of Figures

2.1. Bernstein polynomial of degree 3	6
2.2. Bézier curve for example 2.2	7
3.1. Smoothing a corner with the help of an arc - triangle . .	9
3.2. Angular change with blending arc	12
3.3. Curvature progression for a blending arc	12
3.4. Maximum range for a blending arc of a circle - $L(\varepsilon = \text{const}, \alpha)$	12
3.5. Deviation when specifying the distance L when rounding with a circular arc	13
4.1. Programme flow chart „Corner rounding“	33
4.2. Programme flowchart „Selection of the rounding strategies“	34

List of Tables

List of Listings

5.1. „Hello World“ in Python – Variant 1	36
5.2. „Hello World“ in Python – Variant 2	37

Acronyms

CNC Computerized Numerical Control

SPS Speicherprogrammierbare Steuerung

1. Hints

Please, use **biber.exe**.

If you want to create a symbol directory, call makeindex:
makeindex %.nlo -s nomencl.ist

2. Bézier-Curves

2.1. Introduction

Bézier curves are parameter curves that can be used to represent free-form curves. They are named after the French engineer Pierre Bézier, who developed them in the 1960s at Renault to design car body shapes. During the same period, French physicist and mathematician Paul de Casteljau developed these curves independently of Pierre Bézier at Citroën. Paul de Casteljau's results were available earlier, but they were not published. This is the reason why Pierre Bézier is the namesake of these parameter curves.[Far02]

The Bézier curves are the basis for computer-aided design of models. This is referred to either as Computer Aided Design (CAD) or, when the emphasis is more on a mathematical view, Computer Aided Geometric Design (CAGD). [BN11] Computer requires a mathematical description of shapes to represent them. The most suitable description method for this is the use of parametric curves and surfaces. Here Bézier curves play the central role, because they are the most numerically stable polynomial bases used in CAD/CAGD software.[Far02]

Typography on the computer also uses Bézier curves. There are two ways of representing type with the computer. The simplest way is to save each individual letter with a fixed resolution and size as a bitmap and copy it to the memory area of the screen as needed. These so-called bitmap fonts can be displayed quickly but require a lot of memory if the characters are to be available in different sizes. In this case, an extra bitmap must be created for each size. The quality also decreases when these bitmap fonts are scaled in size and resolution. Vector fonts are an alternative. Their name is derived from the fact that they use curves defined in a two-dimensional vector space to represent the characters. The advantage here is that the characters displayed in this way can be scaled without loss of quality. Two standards have developed for vector fonts. One is the TrueType font and the other is the PostScript font. The TrueType fonts use square Bézier curves while the PostScript fonts use cubic Bézier curves. The cubic Bézier curves have more control points which leads to a better quality.

Another field of application is the control of machine tools. When moving to a corner, the axis must be braked to a standstill at the corner point and then accelerated again after direction correction. This procedure ensures that it is not possible to move at a constant speed, which has negative consequences for the cycle time and quality. A possible solution to this problem is to place a curve between the two sections instead of a sharp corner, which can be run at a constant speed. Bézier curves are suitable for this purpose because only two points and two tangents are needed to form them. In the case of a corner, the points as well as the tangents would lie on the sections that form the corner. The resulting error would be analytically controllable and would allow an adjustment of the curve tolerance.[SS14]

2.2. Allgemeine mathematische Beschreibung Bézier-Kurve

Bézier curves are formed with the help of Bernstein polynomials. If at least two points and two tangents are known, control points can be determined with which a control polygon is formed. The course of the curve is oriented to this control polygon. The number of control points depends on the degree of the Bézier curve. A Bézier curve of degree n has $n+1$ control points. The Bézier curve is calculated using the De Casteljau algorithm.

Definition. In the interval $[0; 1]$ the **Bernstein polynomial of n^{th} degree** is defined by:

$$b_{i,n}(\lambda) = \binom{n}{i} (1-\lambda)^{n-i} \lambda^i, \quad \lambda \in [0, 1], \quad i = 0, \dots, n \quad (2.1)$$

Definition. The binomial coefficient is defined by:

$$\binom{n}{i} = \frac{n!}{i!(n-i)!}, \quad i = 0, \dots, n \quad (2.2)$$

Here the Bernstein polynomials $b_{i,n}$ form a basis of the vector space $\mathbb{P}^n(I)$ of polynomials of degree at most n over I . Thus every polynomial of degree at most n can be written uniquely as a linear combination. [Far02]

Beispiel. Determination of Bernstein polynomials for $n = 3$ holds:

$$b_{3,0}(\lambda) = \binom{3}{0} (1-\lambda)^{3-0} \lambda^0 = (1-\lambda)^3$$

$$b_{3,1}(\lambda) = \binom{3}{1} (1-\lambda)^{3-1} \lambda^1 = 3\lambda(1-\lambda)^2$$

$$b_{3,2}(\lambda) = \binom{3}{2} (1-\lambda)^{3-2} \lambda^2 = 3\lambda^2(1-\lambda)$$

$$b_{3,3}(\lambda) = \binom{3}{3} (1-\lambda)^{3-3} \lambda^3 = \lambda^3$$

$b_{3,i}(\lambda)$ with $i = 0, 1, 2, 3$ are the cubic Bernstein polynomials of degree 3.

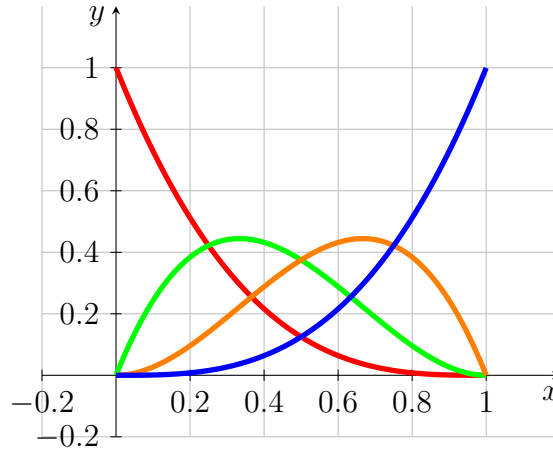


Figure 2.1.: Bernstein polynomial of degree 3

Definition. Given are the points

$$Q_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix} \quad \text{mit} \quad Q_i \in \mathbb{R}^2, \quad i = 0, 1, \dots, n$$

A **Bézier curve** is then defined by

$$C(\lambda) = \sum_{i=0}^n Q_i \cdot b_{i,n}(\lambda) \quad (2.3)$$

The points $Q_i, i = 0, \dots, n$ are called **control points**.

The control points of a Bézier curve form the so-called control polygon.

Bemerkung. Let the starting point P_0 and the end point P_1 , as well as the tangents \vec{t}_0 and \vec{t}_1 be given. The tangents are not necessarily normalised.

The control points Q_0, Q_1, Q_2 and Q_3 of the associated Bézier curve are given by the following equations:

$$Q_0 = P_0, \quad Q_1 = P_0 + \lambda_0 \vec{t}_0, \quad Q_2 = P_1 - \lambda_1 \vec{t}_1, \quad Q_3 = P_1 \quad (2.4)$$

[Jak+10]

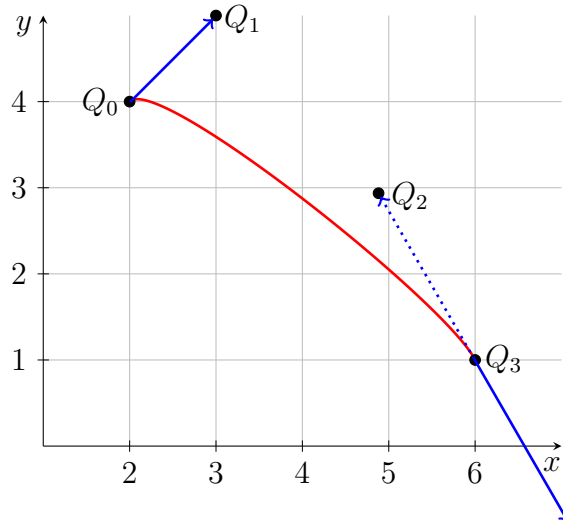


Figure 2.2.: Bézier curve for example 2.2

Beispiel. Given are

$$P_0 = \begin{pmatrix} 2 \\ 4 \end{pmatrix}, \quad \vec{t}_0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \text{und} \quad P_1 = \begin{pmatrix} 6 \\ 1 \end{pmatrix}, \quad \vec{t}_1 = \begin{pmatrix} 1 \\ -2 \end{pmatrix}$$

Then, according to theorem ?? for control points of the associated Bézier curve results:

$$Q_0 = P_0 = \begin{pmatrix} 2 \\ 4 \end{pmatrix}, \quad Q_1 = P_0 + \vec{t}_0 = \begin{pmatrix} 2 \\ 4 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 5 \end{pmatrix},$$

$$Q_2 = P_1 - \vec{t}_1 = \begin{pmatrix} 6 \\ 1 \end{pmatrix} - \begin{pmatrix} 1 \\ -2 \end{pmatrix} = \begin{pmatrix} 5 \\ 3 \end{pmatrix}, \quad Q_3 = P_1 = \begin{pmatrix} 6 \\ 1 \end{pmatrix}$$

The Bézier curve and its control points are shown in the figure ??.

3. Verrundung mit einem Kreisbogen

3.1. Gleichungen

Blending with an arc is a simple variant of smoothing corners. This variant enables the processing and the generation of files according to DIN 66025. For smoothing, three points P_0 and S and P_1 are always considered, thus a symmetric Hermite problem results. According to theorem ?? it is assumed to be in the standard form $HP(L, \alpha)$.

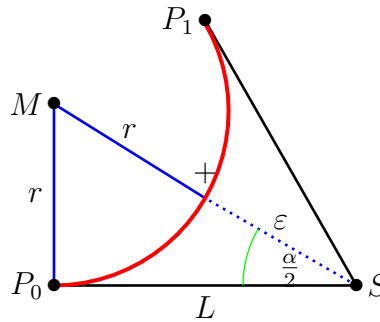


Figure 3.1.: Smoothing a corner with the help of an arc - triangle

The figure ?? represents the situation. The points P_0 , S and P_1 are given. The included angle $\alpha = \angle(P_0; S; P_1)$ as well as the distance $L = \|S - P_0\| = \|P_1 - S\|$ are shown in the graph. The red arc is the desired result. The distance of its centre M to S is then $r + \varepsilon$, where r is the radius of the arc and ε is the given tolerance. Thus we obtain a right triangle P_0SM whose edge lengths are L , $r + \varepsilon$ and r .

According to the definition of the sine and the tangent, it follows:

$$\sin\left(\frac{\alpha}{2}\right) = \frac{L}{r + \varepsilon} \quad \text{und} \quad \tan\left(\frac{\alpha}{2}\right) = \frac{L}{r}$$

$$\Leftrightarrow \varepsilon = \frac{L}{\sin\left(\frac{\alpha}{2}\right)} - r \quad \text{und} \quad r = \cot\left(\frac{\alpha}{2}\right) L$$

The two equations can be combined to give the following conditions:

$$\varepsilon = L \cdot \frac{1 - \cos\left(\frac{\alpha}{2}\right)}{\sin\left(\frac{\alpha}{2}\right)} \quad \text{bzw.} \quad L = \varepsilon \cdot \frac{\sin\left(\frac{\alpha}{2}\right)}{1 - \cos\left(\frac{\alpha}{2}\right)}$$

This gives the equation for the radius:

$$r = L \cdot \cot\left(\frac{\alpha}{2}\right) = L \cdot \sqrt{\frac{1 - \cos(\alpha)}{1 + \cos(\alpha)}} = L \cdot \frac{\sin(\alpha)}{1 + \cos(\alpha)} = L \cdot \frac{P_{1,x}}{P_{1,y}}$$

The factor for converting L and ε can be simplified using trigonometric transformations.

$$\frac{1 - \cos\left(\frac{\alpha}{2}\right)}{\sin\left(\frac{\alpha}{2}\right)} = \frac{1 - \sqrt{\frac{1 - \cos(\alpha)}{2}}}{\sqrt{\frac{1 + \cos(\alpha)}{2}}} = \frac{\sqrt{2} - \sqrt{1 - \cos(\alpha)}}{\sqrt{1 + \cos(\alpha)}} = \frac{\sqrt{1 + \cos(\alpha)} - \sin(\alpha)}{1 + \cos(\alpha)}$$

By comparing this with the symmetric Hermite problem in standard form, the following notation is then obtained:

$$\frac{1 - \cos\left(\frac{\alpha}{2}\right)}{\sin\left(\frac{\alpha}{2}\right)} = \frac{\sqrt{P_{1,x} - P_{1,y}}}{P_{1,x}}$$

The previous considerations are now summarised in the following sentence.

Satz. Let a symmetric Hermite problem $(P_0, \vec{t}_0, P_1, \vec{t}_1, S, L)$ be given. Without restriction of generality, it is in the standard form

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} L + L \cdot \cos(\alpha) \\ L \cdot \sin(\alpha) \end{pmatrix}, \begin{pmatrix} \cos(\alpha) \\ \sin(\alpha) \end{pmatrix}, \begin{pmatrix} L \\ 0 \end{pmatrix}, L \right\}$$

with $L \in \mathbb{R}^{>0}$ and $\alpha \in (-\pi; \pi]$.

Then an arc can be found that connects the points P_0 and P_1 , has the same tangent directions at the two points.

For the circular arcs applies:

$$r = L \cdot \left| \frac{P_{1,x}}{P_{1,y}} \right|; \quad \phi_0 = -\text{sign}(\alpha) \cdot \frac{\pi}{2}; \quad \phi_1 - \phi_0 = \text{sign}(\alpha) \cdot \pi - \alpha = \beta;$$

$$M = P_0 + \text{sign}(\alpha) \cdot r \cdot \vec{t}_0^\perp = \text{sign}(\alpha) \cdot r \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

From the specification of the distance L , the maximum error can be calculated, as shown in theorem ???. According to the derivation, it is also possible to specify the maximum error ε and determine the maximum distance L from it.

Satz. Let a symmetric Hermite problem $(P_0, \vec{t}_0, P_1, \vec{t}_1, S, L)$ be given. Without restriction of generality, it is in the standard form

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} L + L \cdot \cos(\alpha) \\ L \cdot \sin(\alpha) \end{pmatrix}, \begin{pmatrix} \cos(\alpha) \\ \sin(\alpha) \end{pmatrix}, \begin{pmatrix} L \\ 0 \end{pmatrix}, L \right\}$$

with $L \in \mathbb{R}^{>0}$ and $\alpha \in (-\pi; \pi]$.

- a) Let the maximum error ε be given. The maximum distance L for which an arc exists according to the theorem ??? that takes the error into account is given by:

$$L(\varepsilon, \alpha) = \varepsilon \cdot \frac{P_{1,x}}{\sqrt{P_{1,x}} - P_{1,y}} = \varepsilon \cdot \frac{L + L \cdot \cos(\alpha)}{\sqrt{L + L \cdot \cos(\alpha)} - L + L \cdot \cos(\alpha)}$$

- b) When the distance L is specified, the following maximum error results:

$$\varepsilon(L, \alpha) = L \cdot \frac{\sqrt{P_{1,x}} - P_{1,y}}{P_{1,x}} = L \cdot \frac{\sqrt{L + L \cdot \cos(\alpha)} - L + L \cdot \cos(\alpha)}{L + L \cdot \cos(\alpha)}$$

These considerations result in limitations for the use of this strategy, which are summarised in the following comment.

Bemerkung. The following conditions for applying the strategy of a circle must be fulfilled:

- a)

$$P_0 \neq P_1$$

- b)

$$\vec{t}_0 = -\vec{t}_1 \quad \Leftrightarrow \quad \alpha = 0$$

- c)

$$\vec{t}_0 = \vec{t}_1 \quad \Leftrightarrow \quad \alpha = \pi$$

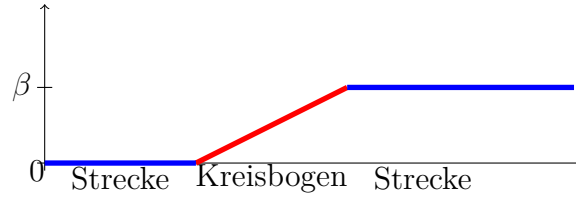


Figure 3.2.: Angular change with blending arc

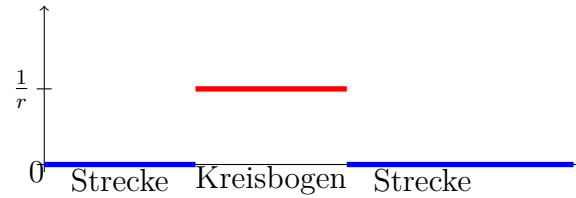


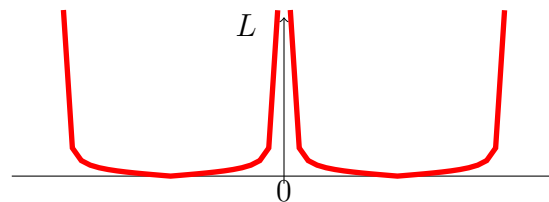
Figure 3.3.: Curvature progression for a blending arc

3.2. Bewertung

Rounding by means of a circular arc leads to a continuous course of the angle change. The figure ?? illustrates this.

Due to the rounding with a circular arc, the curvature of the curve is not continuous. The figure ?? shows that the curvature is constant in the range of distances 0 and on the circular arc with the value $\frac{1}{r}$, where r is the radius of the circular arc used. Due to the formula $a = \frac{v^2}{r}$ for a movement on a circular arc, the acceleration course exhibits continuity jumps at the transitions for a constant path velocity.

Depending on the angle change β at the corner S and the tolerance ε , the maximum length of the shortening of the lines can be calculated. The figure ?? shows the corresponding diagram, where the tolerance ε is set to the value 1.

Figure 3.4.: Maximum range for a blending arc of a circle - $L(\varepsilon = \text{const}, \alpha)$

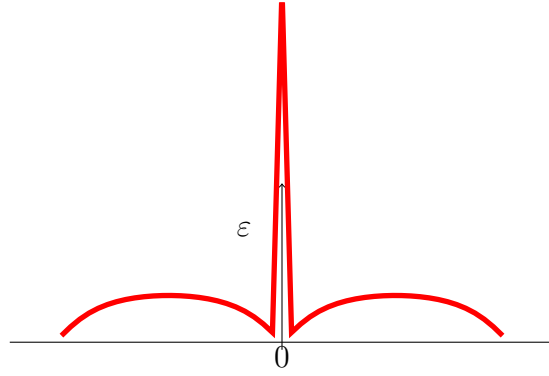


Figure 3.5.: Deviation when specifying the distance L when rounding with a circular arc

The area provided can also be specified. Depending on the distance L from the corner point S , the deviation ε can be calculated. The figure ?? represents the function as a function of L and the angle α .

The figures ?? and ?? show the curves of the length as a function of the angle change for a fixed tolerance and the error as a function of the angle change for a given length. If the angle change is minimal, then the error or length L is extreme. In this case, rounding by means of an arc is not possible. In order to develop a sensible strategy, a minimum angle change must be specified from which a blending arc is permitted. Likewise, a maximum angle change must also be defined. For an angular change of $\pm\pi$ is a bend. In this case, a blending is also not possible.

3.3. Examples

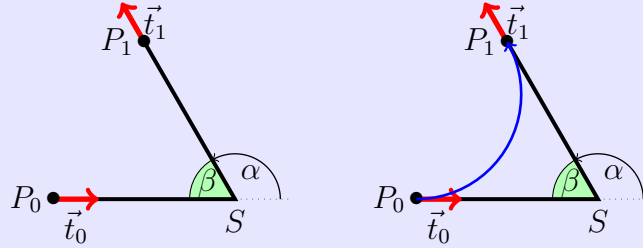
Beispiel. Let it be the symmetric Hermite problem

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 6.0 + 6.0 \cdot \cos\left(\frac{2}{3}\pi\right) \\ 6.0 \cdot \sin\left(\frac{2}{3}\pi\right) \end{pmatrix}, \begin{pmatrix} \cos\left(\frac{2}{3}\pi\right) \\ \sin\left(\frac{2}{3}\pi\right) \end{pmatrix}, \begin{pmatrix} 6.0 \\ 0 \end{pmatrix}, 6.0 \right\}$$

with $L = 6$ and $\alpha = \frac{2}{3}\pi$.

For the blending arc follows:

$$M = \begin{pmatrix} 0 \\ 3,4641 \end{pmatrix}; \quad r = 3,46412; \quad \phi_0 = -\frac{\pi}{2}; \quad \alpha = \frac{2}{3}\pi$$



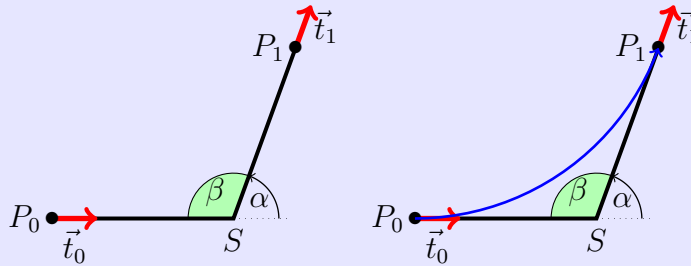
Beispiel. Let it be the symmetric Hermite problem

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 6.0 + 6.0 \cdot \cos\left(\frac{7}{18}\pi\right) \\ 6.0 \cdot \sin\left(\frac{7}{18}\pi\right) \end{pmatrix}, \begin{pmatrix} \cos\left(\frac{7}{18}\pi\right) \\ \sin\left(\frac{7}{18}\pi\right) \end{pmatrix}, \begin{pmatrix} 6.0 \\ 0 \end{pmatrix}, 6.0 \right\}$$

with $L = 6$ and $\alpha = \frac{7}{18}\pi$.

For the blending arc follows:

$$M = \begin{pmatrix} 0 \\ 8,5689 \end{pmatrix}; \quad r = 8,5689; \quad \phi_0 = -\frac{\pi}{2}; \quad \alpha = \frac{7}{18}\pi$$



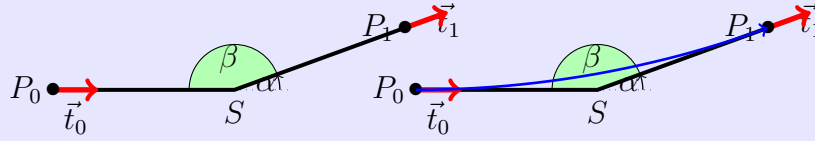
Beispiel. Let it be the symmetric Hermite problem

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 6.0 + 6.0 \cdot \cos\left(\frac{1}{9}\pi\right) \\ 6.0 \cdot \sin\left(\frac{1}{9}\pi\right) \end{pmatrix}, \begin{pmatrix} \cos\left(\frac{1}{9}\pi\right) \\ \sin\left(\frac{1}{9}\pi\right) \end{pmatrix}, \begin{pmatrix} 6.0 \\ 0 \end{pmatrix}, 6.0 \right\}$$

with $L = 6$ and $\alpha = \frac{1}{9}\pi$.

For the blending arc follows:

$$M = \begin{pmatrix} 0 \\ 34,0277 \end{pmatrix}; \quad r = 34,0277; \quad \phi_0 = -\frac{\pi}{2}; \quad \alpha = \frac{1}{9}\pi$$

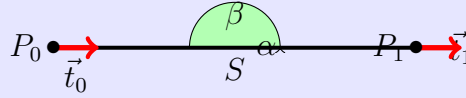


Beispiel. Let it be the symmetric Hermite problem

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 12.0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 6.0 \\ 0 \end{pmatrix}, 6.0 \right\}$$

with $L = 6$ and $\alpha = 0$.

There is no blending arc here.



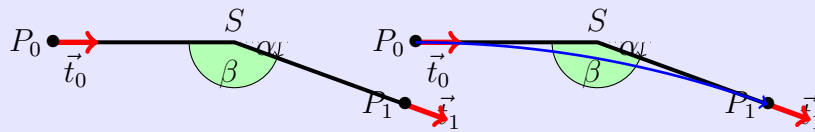
Beispiel. Let it be the symmetric Hermite problem

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 6.0 + 6.0 \cdot \cos\left(-\frac{1}{9}\pi\right) \\ 6.0 \cdot \sin\left(-\frac{1}{9}\pi\right) \end{pmatrix}, \begin{pmatrix} \cos\left(-\frac{1}{9}\pi\right) \\ \sin\left(-\frac{1}{9}\pi\right) \end{pmatrix}, \begin{pmatrix} 6.0 \\ 0 \end{pmatrix}, 6.0 \right\}$$

with $L = 6$ and $\alpha = -\frac{1}{9}\pi$.

For the blending arc follows:

$$M = \begin{pmatrix} 0 \\ -34,0277 \end{pmatrix}; \quad r = 34,0277; \quad \phi_0 = \frac{\pi}{2}; \quad \alpha = \frac{1}{9}\pi$$



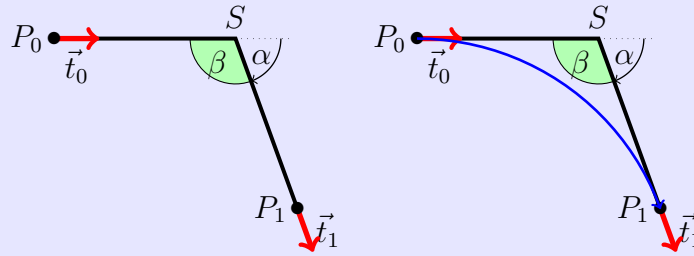
Beispiel. Let it be the symmetric Hermite problem

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 6.0 + 6.0 \cdot \cos\left(-\frac{7}{18}\pi\right) \\ 6.0 \cdot \sin\left(-\frac{7}{18}\pi\right) \end{pmatrix}, \begin{pmatrix} \cos\left(-\frac{7}{18}\pi\right) \\ \sin\left(-\frac{7}{18}\pi\right) \end{pmatrix}, \begin{pmatrix} 6.0 \\ 0 \end{pmatrix}, 6.0 \right\}$$

with $L = 6$ and $\alpha = -\frac{7}{18}\pi$.

For the blending arc follows:

$$M = \begin{pmatrix} 0 \\ -8,5689 \end{pmatrix}; \quad r = 8,5689; \quad \phi_0 = \frac{\pi}{2}; \quad \alpha = -\frac{7}{18}\pi$$



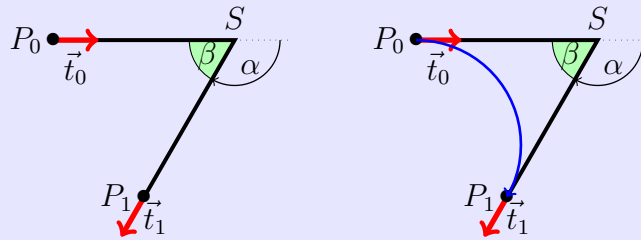
Beispiel. Let it be the symmetric Hermite problem

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 6.0 + 6.0 \cdot \cos\left(-\frac{2}{3}\pi\right) \\ 6.0 \cdot \sin\left(-\frac{2}{3}\pi\right) \end{pmatrix}, \begin{pmatrix} \cos\left(-\frac{2}{3}\pi\right) \\ \sin\left(-\frac{2}{3}\pi\right) \end{pmatrix}, \begin{pmatrix} 6.0 \\ 0 \end{pmatrix}, 6.0 \right\}$$

with $L = 6$ and $\alpha = -\frac{2}{3}\pi$.

For the blending arc follows:

$$M = \begin{pmatrix} 0 \\ -3,4641 \end{pmatrix}; \quad r = 3,46412; \quad \phi_0 = \frac{\pi}{2}; \quad \alpha = -\frac{2}{3}\pi$$



4. Maple files

[Wat17c; Wat17d; Wat17b; Wat17e; Wat17a; Wat17f]

4.1. Geometries with Maple

The algorithms presented can be well represented using geometries. Two aspects can be investigated. On the one hand, the effort for an implementation, the computational accuracy and the stability of an algorithm are interesting; on the other hand, the methods are to be evaluated and compared with regard to their usefulness. For this purpose, modules for the formula manipulation system Maple [Wat17c] were developed.

Maple offers the environment to implement algorithms easily and quickly. Furthermore, graphical representation is possible with simple means. However, a simple workbook of Maple is not designed for very large software projects. Here, one must resort to the possibility of creating and using libraries. On the one hand, Maple offers the possibility of creating one's own libraries, so-called modules. In this way, one remains within the environment and syntax of Maple. This path will be pursued further here. Another possibility is the use of libraries created by means of a higher programming language, e.g. C++, the so-called DLLs.

In the following, the creation and use of a test environment with the help of modules is described. First, the geometry elements that are stored in various modules and their use are described. The structure and use of a module in Maple is then explained so that own extensions and additions are possible.

4.2. Geometry elements used

This is a test environment for the algorithms presented, only geometries that have been described are also used.

- points (**MPoint**)
- lines (**MLine**)
- arcs (**MArc**)
- Bézier curves (**Bezier**)

- polygon courses (**MPolygon**)
- Geometry List (**MGeoList**)
- Hermite problems (**MHermiteProblem**)
- Symmetric Hermite Problems (**MHermiteProblemSym**)

For each geometry element a corresponding module has been created, the name of which is given in the list above. The list of which functions are available is presented in another section.

When implementing such a project, it quickly becomes clear that when using several geometry elements, a clear data structure and well-defined access to the data is essential. For example, when representing straight lines, the decision must be made whether the representation should be point-directional or by means of two points. The choice is generally made depending on the application. Here, the path is taken that, due to a well-defined access to the data, the use of both representations is possible.

4.3. Building the data structure

The idea is to use a data structure that is as uniform and simple as possible. Maple does offer the possibility to define objects. However, it is difficult to use within a procedural environment. Therefore, all data is basically represented as lists. The first list element always contains an identifier of the element `??`. This is followed by the data, which in turn can be geometry elements. It should be noted that direct access to the data cannot be prevented; here the user is responsible.

Access to the data should be exclusively via procedures of a module. The following is an example of the data structure for points:

```
[MVPOINT, [x, y]]
```

The creation of a point then takes place via a procedure **New**:

```
NeuerPunkt := MPoint:-New(10,15);
```

When called up in the test file, the following is then output:

```
TestP0 := ["Point", [10,15]]
```

These data structures are used for the calculation of geometries. They are used in functions or procedures. Unlike variables, the data structures and procedures are defined globally and not locally. Under the command **export** the procedures are declared at the beginning of the module and can thus also be used outside the module. If, for example, a data structure from **MPoint** is to be used in another module or outside the modules, it is called as follows:

```
P0 := MPoint:-New(x,y);
```

In addition to the modules for the geometries, there is a module (MConstant) for saving constants and names. There, for **MVPOINT**, for example. „Point “ or e.g. a constant for the comparison to zero for real numbers.

Finally there is the test file, which is not a module, in which the modules are loaded, called and tested in their function. More about this file later in „1.4 Maple test file “.

4.4. Structure of a module

The following section deals with the purpose of modules and their structure.

The project is about capturing the above geometries in a data structure and representing them in Maple. However, the calculations and formulas that have to be used are a hindrance to the final representation. Modules are very well suited for summarising and hiding the calculations that take place in functions. This way, the important functions can be accessed in Maple without seeing what is in them.

Example:

The function **Angle** from the module **MPoint**: Function to calculate an angle between location vector and x-axis:

```
Angle := proc(P)
  local alpha, x, y;
  x := GetX(P);
  y := GetY(P);
  alpha := 0;
  if abs(x) < 0.00001
  then
    alpha := 3*Pi/2;
  else
    alpha := Pi/2;
  end if;
else
  alpha := arctan(y/x);
  if x < 0
  then
    alpha := alpha + Pi;
  else
    if y < 0
    then
```

```
        alpha := alpha + 2*Pi;
      end if;
    end if;
  end if;
  return factor(alpha);
end proc;
```

This function is long, but it only represents a small part of the programme for the representation in question. That is why it is in the module and can thus be called externally with a single command:

```
Winkel := MPoint:-Angle(P)
```

The following section describes the structure of a module. Since Maple offers a wide range of possibilities, a restriction is made. Only the structure of the modules used is described, here on the basis of the module `MPoint`. For more detailed possibilities, please refer to the Maple manual [Wat17c].

structure:

The module must first be started. This is done with the name (here always a capital M and the geometry) of the module and the following command:

```
MPoint := module()
```

Then, similar to procedures, the variables and functions must be defined. You can declare them either locally or globally. If the variables or procedures are only used and changed in the module, the declaration is made with the command `local` as follows:

```
local Variable names, with, comma, separated;
```

If the variables or procedures are also to be usable outside the module, this is defined with `export`:

```
export Function names, with, comma, separated;
```

This is followed by the specification of the options:

```
option package;
```

the description of the module:

```
description "Self-selected module description, e.g. module for points";
```

and the initialisation of the module:

```
ModuleLoad := proc
  MVPPOINT:=MConstant:-GetPoint();
  print("Module MPoint is loaded");
end proc;
```

From here on, programming is done as usual in Maple. The previously defined procedure and variable names are used and programming is done with commands that are also used outside of a module in Maple. It is important to know that with modules, each function can be called constantly, so the order of the functions is irrelevant.

To finally end the module, use the following command:

```
end module;
```

4.4.1. General structure of a module

In summary, the modules have the following structure:

```
Modulname := module()

  local Names, with, comma, separated;

  export Names, with, comma, separated;

  global Names, with, comma, separated;1

  option package;

  description „ Self-selected module description“;

  ModuleLoad := proc()2
    MVPPOINT:=MConstant:-GetPoint();
    print("Modul MPoint is loaded");
  end proc;

  Procedure1 := proc (Passing parameters)
    inhalt;
  end proc;

  Procedure2 := proc (Passing parameters)
```

¹Possible, but not used in the modules

²Example **MPoint**


```
        content;  
    end proc;
```

```
    ...
```

```
end module;
```

4.4.2. Saving a module

The management of modules is done automatically by Maple. However, since the modules are passed on and have to be edited individually, some settings have to be made. Therefore, the following prefix is used for each Maple file of the project:

```
1 restart;
2 with(LibraryTools);
3 lib := "C:/FH/Tools/Maple/MyLibs/Blending.mla";
4 march('open', lib);
5 ThisModule := 'MArc';
```

The first line initialises the system. The next 3 lines enable working with archives. In the second line, the tools are loaded so that the variable `lib` can be occupied. All modules are stored in an archive; in this example it is the file `Blending.mla` in the directory `C:/FH/Tools/Maple/MyLibs/`. The command `ThisModule := 'MArc';` contains the name of the current module.

A module is then saved using the command `savelib('ModuleName')`. A file `ModuleName.mla` is then created. Depending on the configuration of Maple, the set path where the file is automatically created may not be writable. Then you can configure the system so that the mla file is stored in the current directory or in a directory of your choice.

If the above prefix is used for a Maple file, all that is required to save the module is

```
savelib(ThisModule, lib);
```

4.4.3. Verwendung eines Moduls

A module that has been saved can now be used in other Maple worksheets. To do this, the command

```
with(ModuleName)
```

is used. If you have used a special path, Maple cannot find the file `ModuleName.mla`. Then the path must be made known, e.g.:

```
savelibname := "c:/Maple/MyLibs";, savelibname;
```

Now the procedures of the module can be accessed. An overview of all exported procedures is provided by the command

```
Describe(ModuleName)
```

is displayed. A list of the names including the names of the transfer parameters is displayed. If a procedure is equipped with the field `description`, this text is also displayed.

4.4.4. Creating a Maple module

Creating your own module is done quickly if you use the framework above. However, one should make some considerations beforehand and maintain a standard.

Before creating an own module, the data model and the corresponding procedures should be worked out. A programme flow chart is useful here. The central task of the module is usually quickly determined. In addition, however, the following rules should be observed.

Rule 1 Basically, both the module and each procedure receive a description. This is not limited to the optional argument **description**, but is placed in front of each procedure. The description basically contains the description of the task. The prerequisite is also mentioned or an error handling is described. Then follows the description of all input parameters, their function and their data structure. The return value is then described.

rule 2 Each module receives a procedure **Version()**, which returns the current version number.

rule 3 Data is not global. To access data, procedures **Set*** and **Get*** are provided. These procedures are also used within the procedures of a module.

rule 4 At least one test function is written for each procedure. The test function can be used to illustrate the use and any special features.

4.5. Functions of the modules

In the following, the individual modules are listed. The data structure of each module and all functions with associated tasks are listed.

4.5.1. Module **MPoint**

MPoint is a module for points and works with a data structure and with procedures/functions. The data structure **New** for a point is a list, which is structured as follows:

[MVPOINT, [x,y]]

Its first element is a name to identify the module. Your second element is again a list containing the elements of the geometry. In this case the name is "Point" and the elements are the x and y coordinates for a point. No distinction is made here between point and vector. A point can also be seen as a vector from the coordinate origin to the point.

Via the Get functions **GetX** and **GetY** one can capture the coordinates of the point and use them in other functions. The other functions can then be used to calculate with the points/vectors.

MPoint

E	New	Data structure for a point
E	GetX	Reading the x-coordinate
E	GetY	reading the y-coordinate
E	Angle	calculating the angle between the x-axis and the point
E	Add	Calculates a linear combination of two points/vectors
E	Sub	Calculates the difference of two points/vectors
E	Cos	Calculates the cosine between two vectors
E	Sin	Calculates the sine between two vectors
E	Scale	Scales a vector with a factor
E	Perp	Calculates a vector that is orthogonal to the given vector
L	IllustrateXY	Plot function to illustrate a blue point
E	Illustrate	Plot of a blue point
E	Plot	Plot of a green point
E	Plot2D	Plot of a point with own options
E	Length	Calculates the distance of the point to the coordinate origin
E	Uniform	Normalises the vector to length 1
E	LinetoVector	Calculates vector from a distance
E	Distance	Calculates the distance between two points

4.5.2. Module **MLine**

MLine is a module for lines and works with a data structure and with procedures/functions. The data structure **New** for a line is a list which is structured as follows:

```
[MVLINE, [P0,P1]]
```

Its first element is a name to identify the module. Its second element is again a list containing the elements of the geometry. In this case the name is „Line“ and the elements are the start and end points for a line.

The data structure **NewPointVerctor** is also a data structure for a line, but here the line is defined by a start point and a direction vector.

The Get functions **StartPoint** and **EndPoint** can be used to capture the start and end points of the route and use them in other functions. The other functions can then be used to calculate with the routes.

MLine

E	New	Data structure for a line (two-point form)
E	NewPointVector	creation of a route (point-direction form)
E	StartPoint	reading the starting point
E	EndPoint	reading the end point
E	Position	Calculate a point that lies on the line
E	Plot2D	Plot a part of the route starting from the starting point
E	Plot2DTangent	Plot of a point with tangent
E	Plot2DTangentArrow	Plot of a point with tangent (as arrow)
E	LineLine	Calculation of the intersection of two straight lines.
E	AngleLine	Calculation of the angle between two straight lines

4.5.3. Module **MArc**

MArc is a module for circular arcs and works with a data structure and with procedures/functions. The data structure **New** for an arc is a list that is structured as follows:

[MVARC, [mx,my,r,phi,alpha]]

Its first element is a name to identify the module. Your second element is again a list containing the elements of the geometry. In this case the name is "Arc" and the elements are the x- and y-coordinate for the centre, the radius, the start angle and the angle change for an arc.

The Get functions **GetM**, **GetMX**, **GetMY**, **GetR**, **GetPhi**, and **GetAlpha** can be used to collect the data for an arc and use it in other functions. The other functions can then be used to calculate with the arcs.

MArc

E	New	Data structure for an arc
E	GetMX	Reading the x-coordinate of the centre point.
E	GetMY	read the y-coordinate of the centre point
E	GetR	read the radius
E	GetPhi	reading the start angle
E	GetAlpha	Reading the change of angle
E	GetM	reading the centre point
E	Position	Calculating a point on the arc
E	Plot2D	Plot a part of the arc starting from the start angle
E	Blend	calculating an arc from a symmetric Hermite problem

4.5.4. module **MBezier**

The **New** data structure for a polygon is a list constructed as follows:

[MVBEZIER, [PointList]]

Within the module **MBezier** exist the procedures listed in the following table.

MBezier

E	New	Manual input of control points for a Bézier curve
E	Version	Output of the versions
E	BlendCurvature	Determination of control points from symmetric Hermite problem
E	BlendCurvatureEpsilon	determination of control points from symmetric Hermite problem with given error
E	Position	position on the Bézier curve
E	GetTheta	Reading the angle
E	GetEpsilon	reading the tolerance
E	GetControlPoint	Read the control points
E	Plot2D	Read the Bézier curve
E	PlotControlPoints	representation of all control points

4.5.5. Module **MPolygon**

MPolygon is a module for polygons and works with a data structure and with procedures/functions. The data structure **New** for a polygon is a list that is structured as follows:

[MVPOLYGON, [PointList]]

Its first element is a name to identify the module. Your second element is again a list containing the elements of the geometry. In this case the name is "Polygon" and the elements are any number of points.

Using the Get functions **GetPoint** and **GetN** you can get the number of points and the points themselves and use them in other functions. The other functions can then be used to calculate with the points or the polygon course.

MPolygon

E	New	Data structure for a list of points
E	GetPoint	Reading the ith point from the point list
E	GetN	Determine the number of points in the point list
E	Length	Determination of the Euclidean length of the polygon
E	Position	Calculation of a point on the polygon course
E	Tangents	calculation of the tangent
L	Plot2DAll	Plot list of all points
E	Plot2D	Plot of all points as polygonal plots.
E	Plot2DTangent	representation of the polygon course with tangent
E	PlotPoints	representation of all points

4.5.6. module **MGeoList**

MeoList is a module for a geometry list and works with a data structure and with procedures/functions. The data structure **New** for a geometry list is a list that is structured as follows:

```
[MVGEOLIST, []]
```

Its first element is a name to identify the module. Its second element is again a list containing the elements of the geometry. In this case the name is „GeoList“ and the elements are any number of individual geometries.

Using the Get functions **GeoGeo** and **GetN**, the i-th element of the list and the number of elements in the list can be captured and used in other functions. The other functions can then be used to calculate with the geometries or the list. In the functions **Length**, **Plot2DAll**, **Plot2D** and **Position** the functions are called in themselves. This is possible because the functions work independently of each other.

MGeoList

E	New	Data structure for a geometry list
E	Append	Append a geometry element to the data structure
E	Prepend	Inserting a geometry element as the first element of the list
E	Replace	Replace a geometry element with another one.
E	GetN	Determine the number of geometry elements in the list
E	GeoGeo	Reading the i-th geometry element
E	Length	Calculate the Euclidean length of the geometry elements
E	Position	Calculates a point on the geometry
L	Plot2DAll	Plot function for plotting the geometry elements
E	Plot2D	Plot a part of the geometry list starting from the first element

4.5.7. Module **MHermiteProblem**

MHermiteProblem is a module for Hermite problems and works with a data structure and with procedures/functions. The data structure **New** for a route is a list, which is structured as follows:

```
[MVHERMITEPROBLEM, [P0,T0n,P1,T1n]]
```

Your first element is a name to identify the module. Its second element is again a list containing the elements of the geometry. In this case the name is „HermiteProb“ and the elements are two points with associated tangents (lines).

Using the Get functions **StartPoint**, **EndPoint**, **StartTangent** and **EndTangent**, the points and their tangents can be captured and used in other functions. The other functions can then be used to calculate with the data for the Hermite problem.

MHermiteProblem

E	New	Data structure for a Hermite problem
E	StartPoint	reading the start point
E	EndPoint	Reading the end point command
E	StartTangent	reading the start tangent
E	EndTangent	Reading the end tangent
E	Plot2D	Plotting the Hermite problem

4.5.8. Module MHermiteProblemSym

MHermiteProblemSym is a module for symmetric Hermite problems and works with a data structure and with procedures/functions. The data structure **New** for a symmetric Hermite problem is a list built as follows:

[MVHERMITEPROBLEMSYM, [P0,T0n,P1,T1n,S,L]]

Your first element is a name to identify the module. Its second element is again a list containing the elements of the geometry. In this case the name is „SymHermiteProb“ and the elements are two points with associated tangents, their intersection, and the distance of the points to the intersection.

Via the Get functions **StartPoint**, **EndPoint**, **StartTangent**, **EndTangent**, **CrossPoint** and **ParameterL** one can acquire the data and use it in other functions. The other functions can then be used to calculate with the data for the symmetric Hermite problem.

MHermiteProblemSym

E	New	Data structure for a symmetric Hermite problem
E	StartPoint	reading the start point
E	EndPoint	Reading the end point command
E	StartTangent	reading the start tangent
E	EndTangent	Reading the end tangent
E	ParameterL	reading of the distance
E	CrossPoint	reading of the intersection point
E	Plot2D	Plotting of the symmetrical Hermite problem
E	Create	creation of a Sym. Hermite problem by 3 points
E	BlendArc	rounding of the corner point by an arc

4.5.9. Module **MConstant**

MConstant is a module for storing fixed constants and names to identify data structures. In the locally declared functions, starting with CV, the constants for declaring the different geometries are defined. These are names for recognising the geometry elements in the test file. In the globally declared functions, starting with Get, the names for identifying the geometry elements are returned.

MConstant

L	NULLEPS	constant to compare to zero
L	CVPOINT	constant for geometry elements: Point
L	CVLINE	constant for geometry elements: Line
L	CVARC	constant for geometry elements: Arc
L	CVPOLYGON	constant for geometry elements: polygon
L	CVGEOLIST	constant for geometry elements: GeoList
L	CVHERMITEPROBLEM	constant for geometry elements: HermiteProb
L	CVHERMITEPROBLEMSYMMETRIC	Const. for geometry elements: SymHermiteProb
L	CVBIARC	Const. for geometry elements: Biarc
E	GetNullePs	return of the zero comparison command.
E	GetPoint	identifier for points
E	GetLine	Identifier for lines
E	GetArc	identifier for circular arcs
E	GetPolygon	identifier for polygons
E	GetGeoList	Identifier for geometry lists
E	GetHermiteProblemSymmetric	Identifier for symmetric Hermite problems
E	GetHermiteProblem	ID for Hermite problems
E	GetBiarc	identifier for Biarcs

4.5.10. Module **MGeneralMath**

MGeneralMath is a module for general mathematical functions and works with the data structures and procedures.

MenerealMath

E	MPoint	Data structure for a point
E	MPointX	Reading of the x-coordinate for a point
E	MPointY	reading the y-coordinate for one point
L	MPointIllustrateXY	plot structure for a blue point
E	MPointIllustrate	plot structure for a blue point
E	MPointPlot	Illustration of a green point
E	MLine	Data structure for a line
E	MLineStartPoint	Reading of the start point for a draw frame
E	MLineEndPoint	Reading the end point for a line
E	MPointOnLine	Calculation of a point on the line
E	MLinePlot2D	Plot the part of a line starting at the starting point
. E	MLineLine	calculating the intersection of two linesline

4.5.11. Module Biarc

Data Structure

Requirements and specifications:

The Biarc is to be used to solve a Hermite problem. A Hermite problem is defined as follows:

Given two points P_0 and P_1 with associated normalised tangents \vec{t}_0 and \vec{t}_1 . The biarc must connect the points such that the tangents of the start and end points of the biarc coincide with the tangents of the Hermites problem.

Data structure:

The data structure **New** for a biarc must contain two arcs.

So the data structure for one arc is needed: **MArc: -New**.

This in turn contains the coordinates for the centre, the radius, the start angle and the angle change.

4.5.12. Module MBiarc

MBiarc is a module for Biarcs and works with a data structure and with procedures/functions. The data structure **New** for a Biarc is a list which is structured as follows:

[MVBIARC, [Arc0, Arc1]]

Its first element is a name to identify the module. Your second element is again a list containing the elements of the geometry. In this case the name is „Biarc“ and the elements are two arcs.

Using the Get functions **GetArc0** and **GetArc1** one can capture the two arcs for the biarc. The functions listed below can be used to calculate the data for the biarc.

MBiarc

E	New	Data structure for a biarc
E	GetArc0	Reading of the first arc
E	GetArc1	Reading the second arc
E	Circle	calculating the circle K_J from the Hermite problem data.
E	Plot2DCircle	plot of the circle K_J
E	angle	Calculate the angle from the centre of the circle to points on the circle
E	Plot2D	Plot of the biarc
E	ConnectionPoint	Calculation of the connection point J (Equal Chord)
E	TangentTj	Calculation of the tangent to J
E	Tangent Biarc	rotation of Tj for the biarc.
E	BiarcCenter	Calculate the centres of the arcs of the biarc
E	Biarc	calculate the centre of the biarc.
E	Position	Determine a point on the biarc
E	Blend	Calculate the biarc only from the Hermite problem

4.6. Programme Flowchart

4.6.1. Overall Flow

4.6.2. Verrundung der Kurve

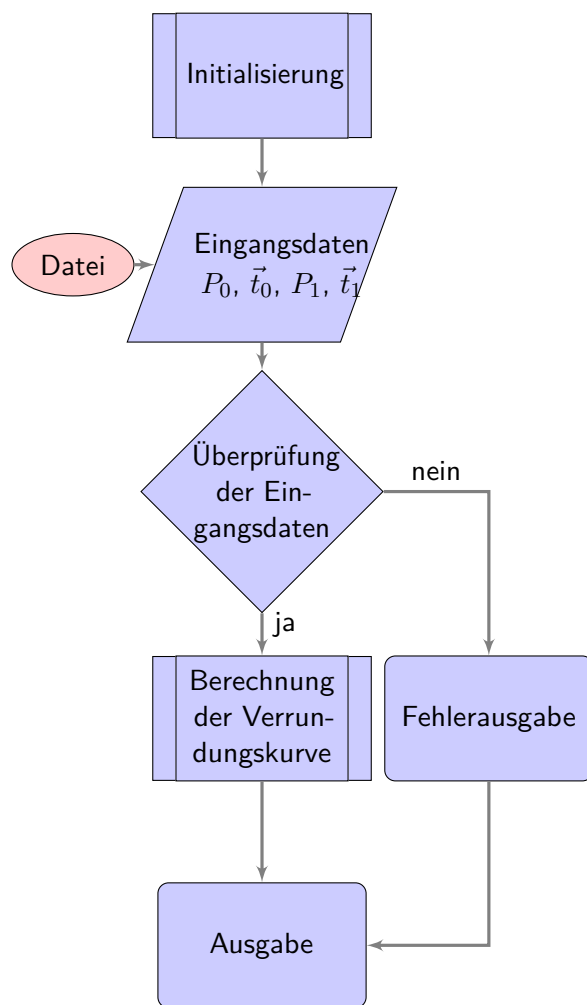


Figure 4.1.: Programme flow chart „Corner rounding“

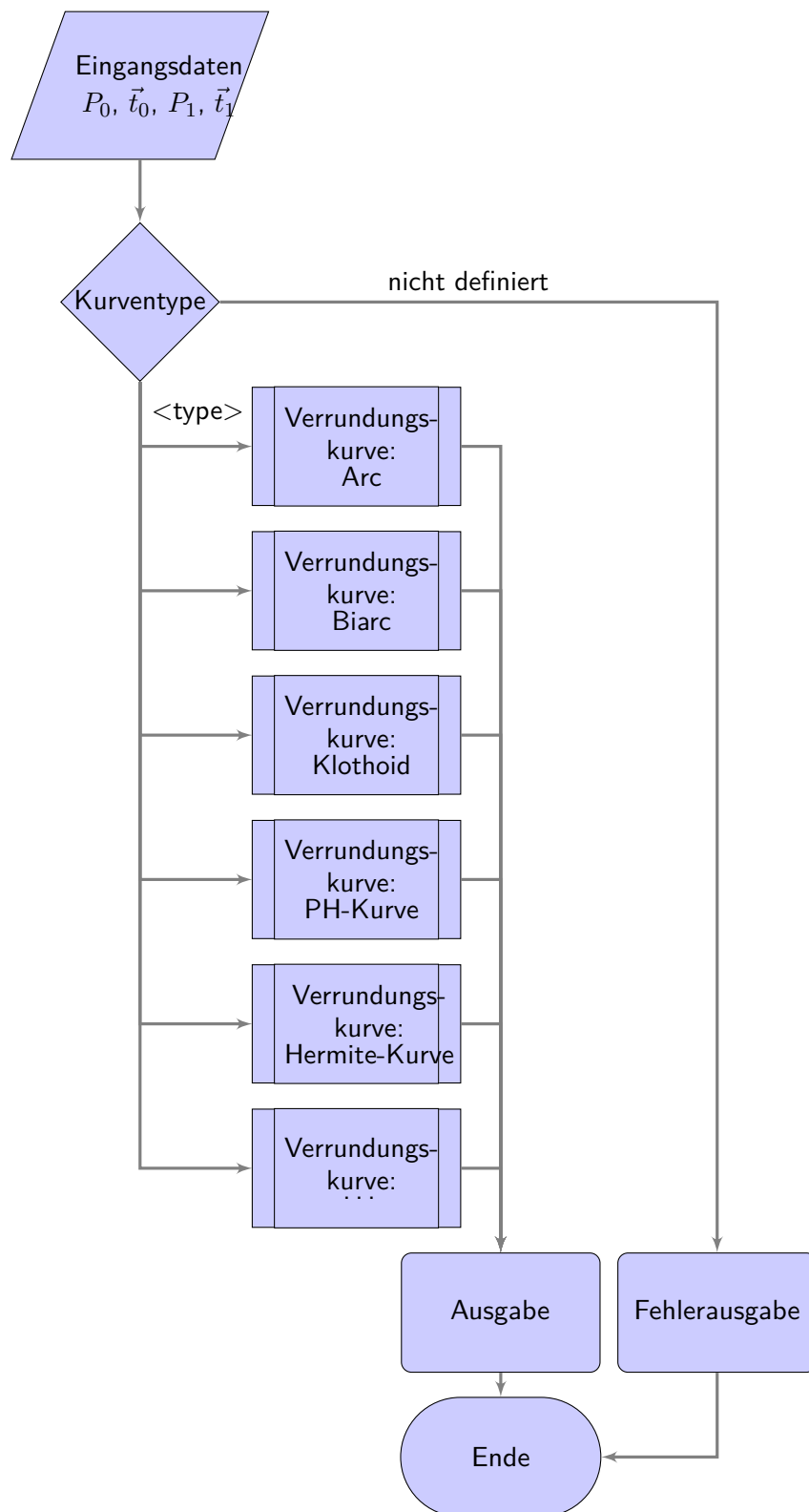


Figure 4.2.: Programme flowchart „Selection of the rounding strategies“

5. Representation of Python Programs

It is possible to integrate a part into the document, see 5.1. This is the most elegant method and is also preferable. Individual lines and line ranges can also be selected in the list of options. If a file is integrated, it is always as up-to-date as the document.

It may also be useful to integrate program lines, see 5.2.

The integration of programs with the help of images is pointless.

```
# Hello World for microcontroller boards
import pyb

redLED = pyb.LED(1) # built-in red LED
greenLED = pyb.LED(2) # built-in green LED
blueLED = pyb.LED(3) # built-in blue LED
while True:
    # Turns on the red LED
    redLED.on()
    # Makes the script wait for 1 second (1000 milliseconds)
    pyb.delay(1000)
    # Turns off the red LED
    redLED.off()
    pyb.delay(1000)
    greenLED.on()
    pyb.delay(1000)
    greenLED.off()
    pyb.delay(1000)
    blueLED.on()
    pyb.delay(1000)
    blueLED.off()
    pyb.delay(1000)
```

Listing 5.1.: The program “Hello World” in Python for microcontroller boards is inserted from the file [Blink.py](#).

```
# Hello World for microcontroller boards
import pyb

redLED = pyb.LED(1) # built-in red LED
greenLED = pyb.LED(2) # built-in green LED
blueLED = pyb.LED(3) # built-in blue LED
while True:
    # Turns on the red LED
    redLED.on()
    # Makes the script wait for 1 second (1000 miliseconds)
    pyb.delay(1000)
    # Turns off the red LED
    redLED.off()
    pyb.delay(1000)
    greenLED.on()
    pyb.delay(1000)
    greenLED.off()
    pyb.delay(1000)
    blueLED.on()
    pyb.delay(1000)
    blueLED.off()
    pyb.delay(1000)
```

Listing 5.2.: The program “Hello World” in Python for microcontroller boards has been inserted directly into the L^AT_EX file..

6. First Chapter

...

A Speicherprogrammierbare Steuerung (SPS) is ...

A Computerized Numerical Control (CNC) needs a SPS (PLC) to ...

7. CAGD

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

The book CAGD by Gerald Farin is a classic on splines. [Far02]

The standard 66025 for programming CNC machines is also a classic; however, it does not deal with splines. [DIN66025-2]

Mr. F. Farouki has dealt with both CNC machine programming and splines. His article¹ on a real-time interpolator also shows this. [FS17]

A new dimension of machine tools have emerged with the invention of 3D printers. [Rus+07]

Another aspect of automation technology is communication. Another milestone has been reached with 5G technology. another milestone has been reached.²

¹co-author is J. Srinathu

²Zaf+20.

A. drawings with tikz

The package `tikz` is a powerful tool for creating graphics. Many introductions exist. Here only the first steps are shown, so that you can easily create flowcharts.

Drawing a line and arrows

```
\begin{tikzpicture}
  \draw (0,0) — (1,0);

  \draw[>-] (2,0) — (3,0);

  \draw[<->] (5,0) — (6,0);
\end{tikzpicture}
```

_____ _____> <_____>

Drawing a thick blue line

```
\begin{tikzpicture}
  \draw[line width=2pt, blue] (0,0) — (1,0);

  \draw[line width=2pt, red, dotted] (2,0) — (3,0);

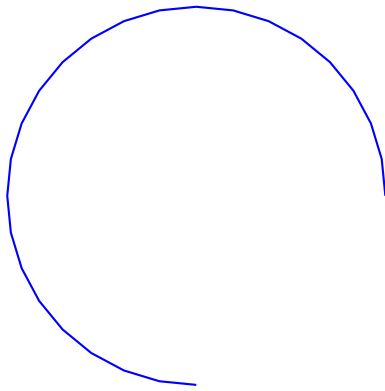
  \draw[line width=2pt, dashed, green] (4,0) — (5,0);

  \draw [thick, dash dot] (0,1) — (5,1);
  \draw [thick, dash pattern={on 7pt off 2pt on 1pt off 3pt}] (0,2)
\end{tikzpicture}
```

.....

Drawing an arc

```
\begin{tikzpicture}
  \draw [blue,thick,domain=0:270] plot ({5+2.5*cos(\x)}, {1+2.5*sin(\x)});
\end{tikzpicture}
```



Draw a function

```
\begin{tikzpicture}[
  declare function={%
    F(\x)                =3-2*pow(2.7979,-0.8*\x);
  }
]

% Zeichnen der Funktion
\draw[red,line width=2pt,domain=0:4] plot({\x},{F(\x)});

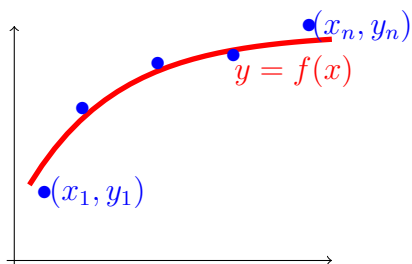
% Bezeichnung
\node[red] (O) at (3.5,2.5) {$y=f(x)$};

% Punkte
\node[blue] (P1n) at (0.9,0.9) {$(x_1,y_1)$};
\node[blue] (P1) at (0.2,0.9) {$\bullet$};

\node[blue] (P2) at (2.7,2.7) {$\bullet$};
\node[blue] (P3) at (1.7,2.6) {$\bullet$};
\node[blue] (P4) at (0.7,2) {$\bullet$};

\node[blue] (P5n) at (4.4,3.1) {$(x_n,y_n)$};
\node[blue] (P5) at (3.7,3.1) {$\bullet$};

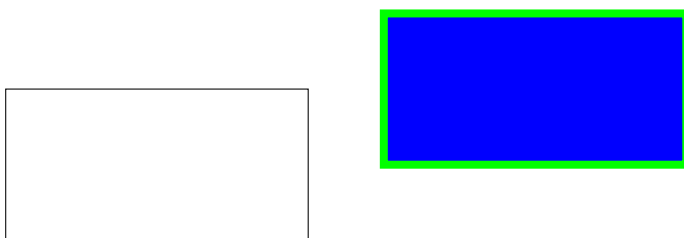
% Koordinatensystem
\draw[color=black,->] (-0.2,-0.1) — (-0.2,3.1);
\draw[color=black,->] (-0.3,0) — (4,0);
\end{tikzpicture}
```



Drawing rectangles and moving objects

```
\begin{tikzpicture}
  \draw (0,0) — (4,0) — (4,2) — (0,2) — cycle;

  \begin{scope}[shift={ (5,1) }]
    \draw[green,fill=blue,line width=3pt] (0,0) — (4,0)
— (4,2) — (0,2) — cycle;
  \end{scope}
\end{tikzpicture}
```



Use of variables

```
\begin{tikzpicture}
\pgfmathsetmacro{\PHI}{-15}
% Now use \PHI anywhere you want -15 to appear,
% can also be used in calculations like 2*\PHI
\def\x{10};

\draw[red] (0,4) — (1-\PHI*0.5,4);
\draw[green] (0,2) — (1+1/\x,2);
\draw[blue] (0,0) — ({1+3*(\x/5+1)},0);
\end{tikzpicture}

\bigskip

%\usetikzlibrary{math} %needed tikz library

\begin{tikzpicture}
\pgfmathsetmacro{\drehpunkt}{11.63815573}
```

```

%Variables must be declared in a tikzmath environment but
% can be used outside
\tikzmath{\x1 = 1; \y1 =1;
%computations are also possible
\x2 = \x1 + 1; \y2 =\y1 +3; }
\draw[>-] (\x1, \y1)--(\x2, \y2);
\end{tikzpicture}

```



Use of points

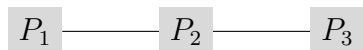
```
%\usetikzlibrary{backgrounds} % is needed
```

```

\begin{tikzpicture}
  \node [fill=gray!30] (P1) at (0,0) { $P_1$ };
  \node [fill=gray!30] (P2) at (2,0) { $P_2$ };
  \node [fill=gray!30] (P3) at (4,0) { $P_3$ };

  \begin{scope}[on background layer]
    \draw (P1) — (P3);
  \end{scope}
\end{tikzpicture}

```



Use of nodes

```

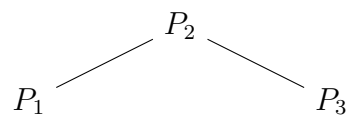
\begin{tikzpicture}
  \node (P1) at (0,0){ $P_1$ };
  \node (P2) at (2,1){ $P_2$ };
  \node (P3) at (4,0){ $P_3$ };

```

```

\begin{scope}
  \draw (P1) — (P2) — (P3);
\end{scope}
\end{tikzpicture}

```

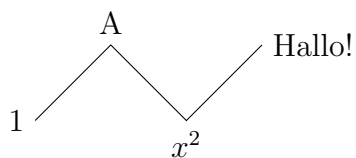


Use of nodes

```

\begin{tikzpicture}
  \draw (0, 0) node[left]{1}
    — ++(1, 1) node[above]{A}
    — ++(1,-1) node[below]{$x^2$}
    — ++(1, 1) node[right]{Hallo!};
\end{tikzpicture}

```



B. Criteria for a good L^AT_EX project

A good report is not only characterised by good content, but also fulfils formal aspects. The following list should help to comply with basic rules. Before submitting, all points should be checked and ticked off.

- ☐ Is a suitable directory structure used?
- ☐ Is an appropriate division into files used?
- ☐ Are meaningful directory and file names used?
 - ☐ Are directory and file names such as report or term paper avoided?
 - ☐ Are meaningful names used for images and not „image1“?
 - ☐ Are directory and file names not too long?
 - ☐ Are special characters and spaces avoided?
- ☐ Are commands like `\newline` and `\\` avoided?
- ☐ Are the L^AT_EX files clearly laid out?
 - ☐ Are indentations used in the L^AT_EX files?
 - ☐ Are free lines inserted?
 - ☐ Is the project mentioned in the header of the files?
 - ☐ Does the header of the files mention the main sources?
 - ☐ Is the author mentioned in the header of the files?
- ☐ Are all necessary files given?
- ☐ Are the temporary files deleted?
- ☐ Are graphics created by yourself?
- ☐ Are the sources of the images given?
- ☐ Are citations made with the command `\cite`?
- ☐ Is a bib file used?

- ☐ Are the entries of the bib-file clearly arranged?
- ☐ Are the entries of the bib-file edited to show them correctly?
- ☐ Are the correct types used for the bib entries?
- ☐ Are meaningful keys used in the bib files?

Unfortunately, the list cannot be complete, but it provides some clues.

Bibliography

- [BN11] H. Babovsky and W. Neundorf. *Numerische Approximation von Funktionen*. Tech. rep. TU Ilmenau, 2011. URL: <https://www.tu-ilmenau.de/fileadmin/media/num/neundorf/Dokumente/Preprints/NumApp1.pdf> (visited on 01/13/2017).
- [DIN66025-2] DIN Deutsches Institut für Normung e.V. *DIN 66025-2:1988-09: Programmaufbau für numerisch gesteuerte Arbeitsmaschinen: Wegbedingungen und Zusatzfunktionen*. Norm. Berlin, Sept. 1988.
- [FS17] R. Farouki and J. Srinathu. “A real-time CNC interpolator algorithm for trimming and filling planar offset curves”. In: *Computer-Aided Design* 86 (Jan. 2017). DOI: [10.1016/j.cad.2017.01.001](https://doi.org/10.1016/j.cad.2017.01.001).
- [Far02] G. Farin. *Curves and Surfaces for CAGD*. 5. [\[pdf\]](#). San Diego, CA: Academic Press, 2002.
- [Jak+10] G. Jaklic et al. “On Interpolation by Planar Cubic G^2 Pythagorean-Hodograph Spline Curves”. In: *Mathematics of Computation* (2010). [\[pdf\]](#).
- [Rus+07] D. Russell et al. *Apparatus and Methods for 3D Printing*. United States Patent, Patent N0.: US 7,291,002 B2. 2007.
- [SS14] B. Sencer and E. Shamoto. “Curvature-continuous sharp corner smoothing scheme for Cartesian motion systems”. In: *2014 IEEE 13th International Workshop on Advanced Motion Control (AMC)*. [\[pdf\]](#) und [\[Version 2 - pdf\]](#). Piscataway, NJ: IEEE, 2014, pp. 374–379. ISBN: 978-1-4799-2323-6. DOI: [10.1109/AMC.2014.6823311](https://doi.org/10.1109/AMC.2014.6823311).
- [Wat17a] Waterloo Maple Inc. 2017. *Description*. [letzter Zugriff 14.09.2017](#). 2017. URL: <https://de.maplesoft.com/support/help/Maple/view.aspx?path=module/description>.
- [Wat17b] Waterloo Maple Inc. 2017. *Export*. [letzter Zugriff 14.09.2017](#). 2017. URL: <https://de.maplesoft.com/support/help/Maple/view.aspx?path=module/export>.

- [Wat17c] Waterloo Maple Inc. 2017. *Module*. [letzter Zugriff 14.09.2017](https://de.maplesoft.com/support/help/maple/view.aspx?path=module). 2017. URL: <https://de.maplesoft.com/support/help/maple/view.aspx?path=module>.
- [Wat17d] Waterloo Maple Inc. 2017. *ModuleLoad*. [letzter Zugriff 14.09.2017](https://de.maplesoft.com/support/help/Maple/view.aspx?path=ModuleLoad). 2017. URL: <https://de.maplesoft.com/support/help/Maple/view.aspx?path=ModuleLoad>.
- [Wat17e] Waterloo Maple Inc. 2017. *Option*. [letzter Zugriff 14.09.2017](https://de.maplesoft.com/support/help/Maple/view.aspx?path=module/option). 2017. URL: <https://de.maplesoft.com/support/help/Maple/view.aspx?path=module/option>.
- [Wat17f] Waterloo Maple Inc. 2017. *Procedures*. [letzter Zugriff 14.09.2017](https://de.maplesoft.com/support/help/Maple/view.aspx?path=procedure). 2017. URL: <https://de.maplesoft.com/support/help/Maple/view.aspx?path=procedure>.
- [Zaf+20] A. Zafeiropoulos et al. “Benchmarking and Profiling 5G Verticals’ Applications: An Industrial IoT Use Case”. In: *IEEE Conference on Network Softwarization, NetSoft 2020*. 2020. DOI: [10.1109/NetSoft48620.2020.9165393](https://doi.org/10.1109/NetSoft48620.2020.9165393).

Index

3D printer, 41

Programmable Logic Controller, 39

CAGD, 41

 Splines, 41

CNN, ix, 39

Computerized Numerical Control

see CNN, ix, 39

PLC, *see* Programmable Logic Controller

Speicherprogrammierbare Steuerung

see SPS, ix, 39

SPS, ix, 39