universität wien

DIPLOMARBEIT

Titel der Diplomarbeit

Numerical enclosures of solution manifolds at near singular points

Verfasser

Simon Konzett

angestrebter akademischer Grad

Magister der Naturwissenschaften (Mag.rer.nat.)

Wien, 2011

Studienkennzahl lt. Studienblatt:	A 405
Studienrichtung lt. Studienblatt:	Diplomstudium Mathematik
Betreuerin / Betreuer:	o. Univ.-Prof. Dr. Arnold Neumaier

Danksagung

Als erstes möchte ich Herrn Univ.-Prof. Dr. Arnold Neumaier für die hervorragende Betreuung beim Erstellen dieser Arbeit danken. Er hat sich immer Zeit für Besprechungstreffen genommen, dabei aufkommende Fragen bestmöglich beantwortet und gute Impulse gegeben um die Arbeit in eine erfolgsversprechende Richtung zu leiten.

Als nächstes möchte ich meiner Familie danken. Im Speziellen natürlich meinen Eltern, Rudi und Wilma, die mich mit der nötigen Geduld immer bestens unterstützt haben, sowohl finanziell als auch moralisch und ohne die dieses Studium so nicht möglich gewesen wäre. Weiters möchte ich auch meinen Geschwistern, Matthias, Martina und Lukas, danken, für tolle gemeinsame Zeiten, die auch das Leben als Student verschönert haben, als auch für gute Ratschläge aller Art.

Dann möchte ich auch allen Freunden für tolle Zeiten danken. Dafür dass ich eine tolle Studienzeit erlebt habe und dafür dass sie mich in schwierigeren Zeiten immer mit gutem Rat unterstützt haben.

Ihnen allen sei diese Arbeit gewidmet.

Abstract

In this work we consider reel parameter-dependent functions which are continuously differentiable and in general nonlinear. We aim to find enclosures for paths of the corresponding roots dependent on this parameter. Especially we are interested in enclosures of such solution paths near singularities. This means near points where two different solution paths are bifurcating or almost bifurcating. There is a augmented problem considered which shall correct the singularities in the original problem and from which one can deduce a low-dimensional problem. The solution of the low-dimensional problem reflects the behaviour of the solution of the originally considered problem. In particular the singular behaviour also is reflected.
First the form of problem we want to consider is introduced according to Neumaier [19]. Then the most important concepts we need to apply the method are provided. These concepts are mainly interval analysis and interpolation with triangular cubic Bezier patches. We need interval analysis because our method shall provide rigorously verified enclosures of the solution paths and in particular we often need good enclosures of the range of reel functions. The Bezier patches are used to get a good interpolation of the solution manifold. In particular a method first introduced by Clough and Tocher and later modified by Farin is used. Further we give some existence and uniqueness results on the solution of the considered problem as well as a short discussion on weighted maximum norms.
Next the reduction process for the considered problem is summarized. Here the main theorems are presented. Further the discussion on the choice of the extension of the problem in detail is done. Moreover a predictor-corrector method is provided for the considered form of problem. The the discussion of the previous sections is applied and we give sketches of algorithms to find rigorous enclosures of the considered solution path by using the provided method. Further the existence and uniqueness of solution branches inside enclosures of the solution manifold is discussed.
The fifth section is about the implementation of the results. The implementation of the results is done in MATLAB R2009a and in particular the toolbox INTLAB is used to get rigorous results. For this toolbox we reference to Rump [23]. Then my own implementation is presented and at least some results of the implementation are presented.
Then a short outlook follows and at least the headers of my implementation in MATLAB are completing the work.

Contents

1 Overview

We consider $F : \Lambda \times D \subseteq \mathbb{R}^p \times \mathbb{R}^n \to \mathbb{R}^m$, a $k_0 \geq 1$ times continuously differentiable nonlinear mapping. We want to study the solution manifold of

$$F(\lambda, u) = 0, \tag{1}$$

especially nearby (near) singularities.
We work with interval analysis to find rigorously verified inclusions where a (near) singularity appears. The method by Neumaier (see Neumaier [19]) aims to describe the solution manifold with a low-dimensional problem even when the solution manifold gets singular or near singular. The method is a generalisation of the Lyapunov-Schmidt reduction technique. However the method does not need an exact point $(\lambda_0, u_0) \in \Lambda \times D$ on the solution manifold and the method still works in the case of imperfect bifurcations.
Beside of interval analysis we discuss in this work Bezier patches to interpolate solution paths. Moreover some existence and uniqueness results like the implicit function theorem are applied.
Now the considered problem is introduced for which the results of Neumaier [19] are applicable.

1.1 A fixed point formulation

As a first step we expand the problem,

$$\tilde{F}(\lambda, \sigma, u, \xi) := \begin{pmatrix} F(\lambda, u) + A_1 \xi \\ \mu(\lambda, u) - \sigma \end{pmatrix}. \tag{2}$$

Here the linear operator $A_1 : \mathbb{R}^k \to \mathbb{R}^m$ and the functional $\mu : \Lambda \times D \to \mathbb{R}^l$ shall be chosen such that a singularity which appears in the original problem at some point shall not appear in the expanded problem any more. In practice, the so called unfolding functional $\mu : \Lambda \times D \to \mathbb{R}^l$ is given and the practitioner is interested on the behaviour of these unfolding functional on the solution manifold. If $p = l = 1$ the plot of the unfolding functional μ against λ is called the bifurcation diagram which may give a useful description of the solution manifold.
We assume that we know an approximate root $(\lambda_0, u_0) \in \Lambda \times D$ of $F : \Lambda \times D \subseteq \mathbb{R}^p \times \mathbb{R}^n \to \mathbb{R}^m$ and further we assume that we have linear mappings $A_0 : \mathbb{R}^n \to \mathbb{R}^m$, A_1 and $A_2 : \mathbb{R}^n \to \mathbb{R}^l$ such that A_0 approximates the derivative of F with respect to u at the point $(\lambda_0, u_0) \in \Lambda \times D$. We write $D_u F(\lambda_0, u_0)$ for this derivative. The linear operator A_2 is chosen such that

$A_2 \approx D_u\mu(\lambda_0, u_0)$. Further the linear operator $A : \mathbb{R}^n \times \mathbb{R}^k \to \mathbb{R}^m \times \mathbb{R}^l$ is defined by

$$A := \begin{pmatrix} A_0 & A_1 \\ A_2 & 0 \end{pmatrix} \tag{3}$$

shall be a bijection between $\mathbb{R}^n \times \mathbb{R}^k$ and $\mathbb{R}^m \times \mathbb{R}^l$ and shall have a bounded inverse

$$\|A^{-1}\| \leq \alpha. \tag{4}$$

Obviously the linear operator A approximates the derivative of (2) with respect to (u, ξ) in a neighbourhood of its approximate root $(\lambda_0, \sigma_0, u_0, \xi_0)$ where $\xi_0 = 0$ and $\sigma_0 = \mu(\lambda_0, u_0)$.
We know that for the case of our interest, when there is a true or an imperfect bifurcation near $(\lambda_0, u_0) \in \Lambda \times D$, there is a point $(\bar{\lambda}, \bar{u}) \in \Lambda \times D$ near our approximate root such that the linear operator $\bar{A}_0 := D_u F(\bar{\lambda}, \bar{u})$ is singular. This means the dimension of the kernel of $\bar{A}_0$ is greater zero.
Now we assume that the linear operator A is chosen appropriately, that means A has the form (3) and (4) holds. For fixed values $\lambda \in \mathbb{R}^p$ and $\sigma \in \mathbb{R}^l$, we consider the nonlinear mapping $\Phi_{\lambda,\sigma} : D \times \mathbb{R}^k \to \mathbb{R}^m \times \mathbb{R}^l$ defined by

$$\Phi_{\lambda,\sigma} \begin{pmatrix} u \\ \xi \end{pmatrix} := \begin{pmatrix} u \\ \xi \end{pmatrix} - A^{-1} \begin{pmatrix} F(\lambda, u) + A_1\xi \\ \mu(\lambda, u) - \sigma \end{pmatrix}. \tag{5}$$

Obviously, any fixed point of the equation above satisfies

$$F(\lambda, u) + A_1\xi = 0 \text{ and } \mu(\lambda, u) = \sigma. \tag{6}$$

Obviously then a solution $(\lambda, \sigma, u, \xi = 0) \in \Lambda \times \mathbb{R}^l \times D \times \mathbb{R}^k$ of the expanded problem implies that $(\lambda, u) \in \Lambda \times D$ is a solution of the original problem.

2 Background

First we introduce the main concepts of interval arithmetic and interval analysis because we want to apply our results in a rigorous manner. Further in one subsection triangular Bezier patches are considered. These Bezier patches we use to interpolate the solution path. Moreover this section includes a short discussion on the appropriate choice of norms as well as some existence and uniqueness results for solution branches of the considered form of problems.

2.1 Interval analysis

To apply our method we need rigorous enclosures of the range of reel functions we are considering. For this aim we introduce next the concept of interval analysis. The following sections are mainly based on the book Neumaier [18].

Introduction and terminology of interval analysis

First we define an interval as a set

$$\mathbf{x} = [\underline{x}, \overline{x}] := \left\{ x \in \mathbb{R} \;\middle|\; \underline{x} \leq x \leq \overline{x} \right\}. \tag{7}$$

For the set of all intervals we write by $\mathbb{IR}$. In this work we denote intervals with bold letters. The expressions $\underline{x}$ and $\overline{x}$ always mean the lower respectively the upper bound of the interval $\mathbf{x}$. Further we define

$$\underline{x} := \inf \mathbf{x} \qquad \overline{x} := \sup \mathbf{x}.$$

A real number $x \in \mathbb{R}$ is represented by the interval $\mathbf{x} = [x, x]$. Such a interval where $\underline{x} = \overline{x}$ we call **thin.** If an interval is not thin we call it **thick.** Further terms we introduce is the **midpoint** and the **radius** of an interval. They are defined in the following way,

$$\operatorname{mid} \mathbf{x} := \frac{\overline{x} + \underline{x}}{2},$$
$$\operatorname{rad} \mathbf{x} := \frac{\overline{x} - \underline{x}}{2}.$$

Next we define an extension of the absolute value of real numbers on intervals. The **magnitude** and in similarly way the **mignitude** of a real interval are defined by

$$|\mathbf{x}| = \operatorname{mag} \mathbf{x} := \max \left\{ |x| \;\middle|\; x \in \mathbf{x} \right\},$$
$$\langle \mathbf{x} \rangle = \operatorname{mig} \mathbf{x} := \min \left\{ |x| \;\middle|\; x \in \mathbf{x} \right\}.$$

Next we define the **convex hull** of a nonempty bounded subset of $\mathbb{R}$,

$$\square S := [\inf S, \sup S].$$

Further some order relations can be defined in interval analysis. We define

$$\mathbf{x} < \mathbf{y} \Leftrightarrow \overline{x} < \underline{y}, \quad \mathbf{x} > \mathbf{y} \Leftrightarrow \underline{x} > \overline{y},$$
$$\mathbf{x} \leq \mathbf{y} \Leftrightarrow \overline{x} \leq \underline{y}, \quad \mathbf{x} \geq \mathbf{y} \Leftrightarrow \underline{x} \geq \overline{y}.$$

These order relations are **antisymmetric**, **transitive** and additionally the order relation $(\leq, \geq)$ is **reflexive**. However two intervals are not **comparable.**
Next we define the elementary operations $(+, -, *, /, **)$ where $\mathbf{a} ** \mathbf{b}$ means $\mathbf{a}^{\mathbf{b}}$ on the set of real intervals by

$$\mathbf{x} \circ \mathbf{y} := \square \left\{ x \circ y \;\middle|\; x \in \mathbf{x}, y \in \mathbf{y} \right\}. \tag{8}$$

where $\circ$ stands for one of the elementary operations. For the elementary operation $\mathbf{x}/\mathbf{y}$ we need the restriction that $0 \notin \mathbf{y}$. For the elementary operation $**$ we also need some restrictions.
Next we define some elementary functions for intervals by

$$\varphi(\mathbf{x}) := \square \left\{ \varphi(x) \;\middle|\; x \in \mathbf{x} \right\}, \tag{9}$$

where φ stands for one function of the set Φ and the set Φ consist of the most elementary functions, here listed: *abs, sqr, sqrt, exp, ln, sin, cos, arctan.*
For all of these elementary operations and functions explicit formulas dependent on the endpoints of the intervals can be written down.

About the rounding

We want to use interval analysis to rigorously solve a system of equations. So we must be aware that we do not loose rigour. We know that a computer only uses a finite set of numbers, often called the **machine-representable numbers.** This set is a subset of $\mathbb{R}$, but the exact results of the computations we want to do must not be in this subset. So we must care about rounding. So if we have an interval $\mathbf{x} = [\underline{x}, \overline{x}]$ where at least one of the endpoints is no machine-representable number and we want to use this interval in a computer we have to take care of that the interval $\tilde{\mathbf{x}}$ which the computer uses is an enclosure of the exact interval, this means $\mathbf{x} \subseteq \tilde{\mathbf{x}}$ must hold. Optimally $\tilde{\mathbf{x}} = [\underline{\tilde{x}}, \overline{\tilde{x}}]$ where $\underline{\tilde{x}}$ is the largest machine-representable number which is less or equal than $\underline{x}$ and $\overline{\tilde{x}}$ is the smallest machine-representable

number which is greater or equal than $\overline{x}$. This procedure is called **optimal outward rounding.**
So we have use all the elementary operations and functions with optimal outward rounding because otherwise we loose the rigour of our computations. We define

$$\begin{aligned} \mathbf{x} \diamondsuit \mathbf{y} &:= \diamondsuit(\mathbf{x} \circ \mathbf{y}) \quad \text{where } \circ \text{ means a elementary operation,} \\ \varphi^{\diamondsuit}(\mathbf{x}) &:= \diamondsuit\varphi(\mathbf{x}) \quad \text{where } \varphi \text{ is an elementary function,} \end{aligned}$$

where $\diamondsuit$ means the optimal outward rounding.

Extension to interval vectors and matrices

Here we define the set of interval vectors with length n, $\mathbb{IR}^n$, by the set of all interval vectors $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ with $\mathbf{x}_i \in \mathbb{IR}$ for all $i = 1, \ldots, n$. Geometrically this can be seen as a rectangular box in $\mathbb{R}^n$. The main terminology from above can be adopted from above in a component-wise sense. We get for the order relations,

$$\mathbf{x} \,\omega\, \mathbf{y} :\Leftrightarrow \mathbf{x}_i \,\omega\, \mathbf{y}_i \quad \forall i = 1, \ldots n,$$

where ω means one of the previously defined order relations $(<, \leq, >, \geq)$. For the terms of infimum, supremum, midpoint, radius, magnitude, hull and outward rounding the extension works in the same way component-wise. The definition for elementary operations and elementary functions can be applied in every component. The addition and subtraction of interval vectors can be defined by a straightforward way component-wise. Also a 'scalar' multiplication is fairly straightforward,

$$\mathbf{ax} := \square \left\{ ax \;\middle|\; a \in \mathbf{a}, x \in \mathbf{x} \right\},$$

where $\mathbf{a} \in \mathbb{IR}$ and $\mathbf{x} \in \mathbb{IR}^n$.
A $m \times n$ interval matrix $\mathbf{A} \in \mathbb{IR}^{m \times n}$ is defined such that every entry $\mathbf{A_{ik}} \in \mathbb{IR}$. Again the definitions for infimum, supremum, midpoint, radius, magnitude, the hull and outward rounding we can adapt easily in a component-wise sense. Hardly surprising the addition and subtraction is defined again in a component-wise sense. Now we define the multiplication. We have $\mathbf{A} \in \mathbb{IR}^{m \times n}$ and $\mathbf{B} \in \mathbb{IR}^{n \times p}$ then we define matrix product $\mathbf{AB} \in \mathbb{IR}^{m \times p}$ by

$$\mathbf{AB} := \square \left\{ AB \;\middle|\; A \in \mathbf{A}, B \in \mathbf{B} \right\}.$$

We also define a 'scalar' multiplication. We have $\mathbf{a} \in \mathbb{IR}$ and $A \in \mathbb{IR}^{n \times n}$ and define

$$\mathbf{aA} = \square \left\{ aA \;\middle|\; a \in \mathbf{a}, A \in \mathbf{A} \right\}.$$

Again the order relations are understood component-wise.

Interval expressions

Next we define some important terms in interval analysis.

Definition 2.1. *An interval function* $f : \mathbb{IR}^n \to \mathbb{IR}$ *is called* INCLUSION ISOTONE *if, for* $\mathbf{x}, \mathbf{y} \in \mathbb{IR}^n$ *the property*

$$\mathbf{x} \subseteq \mathbf{y} \Rightarrow f(\mathbf{x}) \subseteq f(\mathbf{y}) \tag{10}$$

holds.

Definition 2.2. *An interval function* $f : \mathbb{IR}^n \to \mathbb{IR}$ *is called an* INTERVAL ENCLOSURE *of a real function* $f_0 : D \subseteq \mathbb{R}^n \to \mathbb{R}$ *if the properties*

$$f(x) = f_0(x) \qquad x \in D, \tag{11}$$
$$f_0(x) \in f(\mathbf{x}) \quad \forall x \in \mathbf{x} \in \mathbb{I}D, \tag{12}$$

hold. An interval enclosure provides

$$\{f_0(x) \mid x \in \mathbf{x}\} \subseteq f(\mathbf{x}). \tag{13}$$

Obviously the property from above is important for rigour enclosure of a real function f. So one is interested in a class of functions which satisfies this property. Luckily the class of functions which are composed arithmetically through the most common elementary functions we mentioned above satisfies this property. In a very similar way one can also find a class of functions for which this property holds with outward rounding. So we see that the most common functions are satisfying the property above and we have not to care much about this property. For more details on this topic we refer to Neumaier [18, section 1.4].

Comments and prospects

For more on interval evaluation, algebraic and topological properties in interval arithmetic we refer mainly to Neumaier [18] as well as to Moore et al. [16].
Further an important topic to optimize interval evaluation are centred forms. The most common centred forms are mean value forms and slope forms. Here we also refer to Neumaier [18], Moore et al. [16] and additionally to Kearfott [12] and Baumann [2].

2.2 Appropriate norms

Here we shortly discuss weighted maximum norms in vector spaces and the induced matrix norm. We also discuss how to choose this weights. These norms are useful for our method.
First we recall the extended problem which looks in the following way,

$$\tilde{F}_{\lambda,\sigma}(u,\xi) = \begin{pmatrix} F(\lambda,u) + A_1\xi \\ \mu(\lambda,u) - \sigma \end{pmatrix}, \tag{14}$$

where $\tilde{F}_{\lambda,\sigma} : \mathbb{R}^n \times \mathbb{R}^k \to \mathbb{R}^m \times \mathbb{R}^l$. Usually there is $n = m$ and $l = k = 1$. We define the weighted maximum vector norm.

Definition 2.3. *For $x \in \mathbb{R}^n$ we define its* WEIGHTED MAXIMUM NORM *$\|x\|_{\infty,w}$ with weight $w \in \mathbb{R}^n$ where $w_i > 0$ for all $i = 1, \ldots, n$ by*

$$\|x\|_{\infty,w} := \max_i \frac{|x_i|}{w_i}. \tag{15}$$

If we have chosen an appropriate weight $w \in \mathbb{R}^n$ then $\|x\|_{\infty,w} \approx 1$ shall hold when $x \in \mathbb{R}^n$ is in the expected range. So we must choose appropriate weights $w^1, w^2 \in \mathbb{R}^n$ corresponding to the domain and co-domain of the function $\tilde{F}_{\lambda,\sigma} : \mathbb{R}^n \times \mathbb{R}^k \to \mathbb{R}^m \times \mathbb{R}^l$. Furthermore we keep this notation w^1, w^2 for the weights corresponding to the domain and co-domain of the function $\tilde{F}_{\lambda,\sigma}$.
Further we also need the induced compatible matrix norm.

Definition 2.4. *Let $A : \mathbb{R}^n \to \mathbb{R}^m$ be a linear operator and $(\mathbb{R}^n, \|\cdot\|_{\infty,w^1})$, $(\mathbb{R}^m, \|\cdot\|_{\infty,w^2})$ normed vector spaces then the corresponding induced norm is defined by*

$$\|A\|_{\infty,w^1,w^2} := \max_{1\le i\le m} \frac{1}{w_i^2} \sum_{j=1}^{n} w_j^1 \left|a_{ij}\right|. \tag{16}$$

We call this norm the WEIGHTED ABSOLUTE ROW SUM *of the matrix.*

Remarks. *(i) It is easily to see that the choice of a weight $w \in \mathbb{R}^n$ such that $w \in \mathbb{R}^n$ is one in each component leads to the common maximum norm. Similarly the choice of the weights $w^1, w^2 \in \mathbb{R}^n$ such that each component is one leads to the commonly by the maximum norm induced matrix norm, the common absolute row sum of the matrix.*

(ii) If $w^1 \neq w^2$ then we have to take care between which domains the linear operator A maps. In particular $\|A\|_{\infty,w^2,w^1} \neq \|A\|_{\infty,w^1,w^2}$.

Now we define an important operator we use often later.

Definition 2.5. *The operator* $\text{midrad} : \mathbb{R}^n \times \mathbb{R}_+ \to \mathbb{IR}^n$ *is defined by*

$$(\text{midrad}(x,r))_i = [x_i - r, x_i + r] \quad i = 1, \ldots, n.$$

The operator is also valid if $r \in \mathbb{R}^n_+$. *Then* $\text{midrad} : \mathbb{R}^n \times \mathbb{R}^n_+ \to \mathbb{IR}^n$ *is defined by*

$$(\text{midrad}(x,r))_i = [x_i - r_i, x_i + r_i] \quad i = 1, \ldots, n.$$

Remark.
One can see that the midrad *operator can be used to set an enclosure of* $B_\delta(u_0) := \left\{ u \in \mathbb{R}^n \;\middle|\; \|u - u_0\| \leq \delta \right\}$. *If we are using the common maximum norm then* $\text{midrad}(u_0, \delta) = B_\delta(u_0)$ *holds. If we use a weighted maximum norm then we must replace* δ *by* δw^1, *where the vector* w^1 *is a weight like introduced above. Obviously then* δw^1 *is a vector.*

2.3 Bezier interpolation

Here initially the assumptions we make shall be the same like in the previous section. We can use the discussion in the previous section to determine a grid $\mathfrak{G}$ to interpolate the solution in an area $\tilde{\Lambda} \times \tilde{\Sigma} \subseteq (\boldsymbol{\lambda}, \boldsymbol{\sigma})$. Moreover we assume for simplicity that $p = l = 1$.
Next we use the grid $\mathfrak{G}$ to define a triangulation $\mathcal{T}$ in $\tilde{\Lambda} \times \tilde{\Sigma}$. We use a Delaunay triangulation.

Definition 2.6. *We define a* Delaunay triangulation *for a set of points* $\mathfrak{G}$ *in* $\mathbb{R}^2$ *such that the triangulation has to satisfy the property that no point in* $\mathfrak{G}$ *is inside the circumcircle of any triangle of the triangulation.*

Remarks. *(i) The definition of the Delaunay triangulation can be extended to higher dimensions. Then one has to consider circumscribed spheres instead of circumcircles corresponding to the n-simplices.*

(ii) The Delaunay triangulation is not necessarily unique and possibly does not even exist. We do not have uniqueness for four or more points on the same circle. This means the vertices determine a rectangle. The Delaunay triangulation does not exist if all points are on the same line.

(iii) The Delaunay triangulation is dual to the Voronoi diagram.

For more on Delaunay triangulation we refer to Klein [14].
On this triangulation we want to interpolate the solution of the extended problem using cubic **Bezier-splines** on the triangular patches defined with $\mathcal{T}$.
For each of the following introducing subsections we want to reference to Prautzsch et al. [20].

Bernstein polynomials

The basics for using Bezier splines to fit data are the Bernstein polynomials. By looking on the binomial expansion

$$1 = (u + (1-u))^n = \sum_{i=0}^{n} \binom{n}{i} u^i (1-u)^{n-i}$$

we can deduce the definition of the Bernstein polynomials.

Definition 2.7. *We define the* BERNSTEIN POLYNOMIALS OF DEGREE n *with*

$$B_i^n(u) = \binom{u}{i} u^i (1-u)^{n-i}, \qquad (i = 0, \ldots, n). \tag{17}$$

Proposition 2.8. *(i) The Bernstein polynomials are linear independent.*

(ii) They satisfy the symmetry condition $B_i^n(u) = B_{n-i}^n(1-u)$.

(iii) All the roots are either 0 *or* 1, *in particular*

$$B_i^n(0) = \begin{cases} 1 & i = 0, \\ 0 & i > 0. \end{cases}$$

(iv) They form a partition of unity for all $u \in \mathbb{R}$.

(v) They are strictly positive for $u \in [0,1]$ *in the interior of the interval* $[0,1]$.

(vi) They satisfy the recurrence relation

$$B_i^{n+1}(u) = uB_{i-1}^n(u) + (1-u)B_i^n(u), \tag{18}$$

where $B_{-1}^n = B_{n+1}^n = 0$ *and* $B_0^0 = 1$.

These properties make the Bernstein polynomials appropriate for interpolation of curves. Especially the linear independence property says in particular that every polynomial p of degree $\leq n$ has also a unique representation using Bernstein polynomials as the basis of the space of these polynomials. This representation is called the nth degree **Bezier representation** of a polynomial. We write

$$p(u) = \sum_{i=0}^{n} c_i B_i^n(u).$$

With an affine coordinate transformation, $u = (1-t)a+tb$, we get the Bezier representation of a polynomial curve p over an arbitrary interval $[a, b]$,

$$p(u(t)) = \sum_{i=0}^{n} b_i B_i^n(t).$$

The coordinates b_i we call **Bezier points** and they determine the so-named **Bezier polygon** of the curve p over the interval.

Proposition 2.9. *(i) For the end points of the curve over the interval* $[a, b]$ *hold* $p(a) = b_0$ *and* $p(b) = b_n$.

(ii) Bezier representations are affine invariant.

(iii) For $u \in [a, b]$ *the point* $p(u)$ *is a convex combination of its Bezier points. In particular this means that the curve* p *over the interval* $[a, b]$ *lies in the convex hull determined by the corresponding Bezier points. This property implies that* $p([a, b]) \subseteq [\min_{i=0}^{n} b_i, \max_{i=0}^{n} b_i]$ *holds.*

Next we consider the recurrence relation mentioned above. This relation leads us to a very important algorithm when using Bezier patches, named by de Casteljau. We have

$$p(u) = \sum_{i=0}^{n} b_i^0 B_i^n(t) = \sum_{i=0}^{n-1} b_i^1 B_i^{n-1}(t) = \cdots = \sum_{i=0}^{0} b_i^n B_i^0(t) = b_0^n,$$

where

$$b_i^{k+1} = (1-t)b_i^k + tb_{i+1}^k.$$

With these relation we are easily able to evaluate a Bezier patch at a point u represented by its local coordinate t. This scheme for evaluation is called the **De Casteljau** algorithm.
For more information on Bernstein polynomials, Bezier techniques and interpolation of curves with Bezier splines we refer to the book Prautzsch et al. [20]. Now we continue with introducing Bernstein polynomials over triangles in a plane because we use them for interpolating the solution path.

Triangular Bezier-splines

First we introduce the concept of barycentric coordinates in $\mathbb{R}^2$.

Definition 2.10. *Let* $\triangle$ *be a triangle in* $\mathbb{R}^2$ *formed by the three vertices* $a_1, a_2, a_3 \in \mathbb{R}^2$. *Then we define the* NORMED BARYCENTRIC COORDINATES (u, v, w) *for a point* $p \in \mathbb{R}^2$ *with respect to* (a_1, a_2, a_3) *with the properties,*

(i) $u + v + w = 1$,

(ii) $(a_1 + a_2 + a_3)p = a_1u + a_2v + a_3w$.

Remarks. *(i) The point $p \in \mathbb{R}^2$ lies in the interior of the triangle $\triangle$ if and only if $u, v, w \geq 0$.*

(ii) We can describe the triangle $\triangle$ in the following way:

$$\triangle := \{ua_1 + va_2 + wa_3 \big| u, v, w \geq 0,\ u + v + w = 1\}.$$

(iii) The vertices (a_1, a_2, a_3) have the barycentric coordinates $(1, 0, 0), (0, 1, 0)$ and $(0, 0, 1)$.

(iv) The definition of barycentric coordinates from above can easily be extended to higher dimensional simplices.

Now we need formulas for the barycentric coordinates with respect to (a_1, a_2, a_3) of a point $p \in \mathbb{R}^2$. We write $a_i = \begin{pmatrix} a_i^1 \\ a_i^2 \end{pmatrix}$ for $i = 1, 2, 3$ and $p = \begin{pmatrix} p^1 \\ p^2 \end{pmatrix}$. Then we get these coordinates by solving

$$T \begin{pmatrix} u \\ v \end{pmatrix} = (p - a_3),$$
$$w = 1 - u - v,$$

where $T = \begin{pmatrix} a_1^1 - a_3^1 & a_2^1 - a_3^1 \\ a_1^2 - a_3^2 & a_2^2 - a_3^2 \end{pmatrix}$. These system of linear equations is uniquely solvable if the points (a_1, a_2, a_3) define a triangle in $\mathbb{R}^2$. Next we introduce the Bernstein polynomials of degree on such a triangular patch, because they build the basis of interpolation with triangular Bezier splines. Similarly to the introduction of the common Bernstein polynomials we consider the trinomial expansion

$$1 = (u + v + w)^n = \sum_{i,j,k} \frac{n!}{i!j!k!} u^i v^j w^k,$$

where clearly $i, j, k \geq 0$ and $i + j + k = n$. Again we can deduce the definition of the Bernstein polynomials over a triangle.

Definition 2.11. *Let $\triangle$ be a triangle in $\mathbb{R}^2$ defined with the vertices (a_1, a_2, a_3) and let (u, v, w) be the barycentric coordinate vector of a point $x \in \mathbb{R}^2$ with respect to (a_1, a_2, a_3) and let $n \in \mathbb{N}$ be fixed. Then the* Bernstein polynomials of degree *n over $\triangle$ are defined by*

$$B_{ijk}^n(u, v, w) = \frac{n!}{i!j!k!} u^i v^j w^k. \tag{19}$$

With multiindex notation we can write

$$B_{\mathbf{i}}^n(\mathbf{u}) = \binom{n}{\mathbf{i}} \mathbf{u}^{\mathbf{i}},$$

where $\mathbf{i} = (i, j, k)$, $\mathbf{u} = (u, v, w)$ and $|\mathbf{i}| = i + j + k = n$.
Only in this section we use bold symbols for multiindex notation. Elsewhere we mean interval vectors.
Below we use cubic Bernstein polynomials. These polynomials you can see in figure 1 as a canonical listing.

$$u^3$$

$$u^2v \quad u^2w$$

$$uv^2 \quad uvw \quad uw^2$$

$$v^3 \quad v^2w \quad vw^2 \quad w^3$$

Figure 1: Canonical listing of cubic Bernstein polynomials over a triangle.

Proposition 2.12. *(i) There exist* $\frac{1}{2}(n+1)(n+2)$ *Bernstein polynomials of degree* $\leq n$. *In particular, there are* 10 *cubic Bernstein polynomials.*

(ii) The bivariate Bernstein polynomials of degree n *are linearly independent and build a basis of the space of bivariate polynomials from degree* $\leq n$.

(iii) They are symmetric with respect to any permutation. This means $B_{\mathbf{i}}^n(\mathbf{u}) = B_{\pi(\mathbf{i})}^n(\pi(\mathbf{u}))$ *for any permutation* π.

(iv) A zero of these bivariate Bernstein polynomials must be at the edge of the triangle. In particular

$$B_{\mathbf{i}}^n(e_k) = \begin{cases} 1 & \mathbf{i}_k = n, \\ 0 & \mathbf{i}_k < n. \end{cases} \tag{20}$$

(v) They form a partition of unity,

$$\sum_{\mathbf{i}=n} B_{\mathbf{i}}^n(\mathbf{u}) = 1.$$

(vi) If a point $p \in \mathbb{R}^2$ *is in the interior of the triangle* $\triangle$, *i.e. if the barycentric coordinates* $\mathbf{u}$ *from* $p \in \mathbb{R}^2$ *are strictly positive then the Bernstein polynomials are strictly positive too.*

(vii) The Bernstein polynomials satisfy the following recursion formula

$$B^n_{i,j,k}(u,v,w) = uB^{n-1}_{i-1,j,k} + vB^{n-1}_{i,j-1,k} + wB^{n-1}_{i,j,k-1}, \tag{21}$$

where $B^0_{0,0,0} = 1$ initialises this recursion.

(viii) The derivative of a Bernstein polynomial over a triangle satisfies

$$\frac{\partial}{\partial \mathbf{u}_j} B^n_{\mathbf{i}} = nB^{n-1}_{\mathbf{i}-e_j}, \tag{22}$$

where $B_{\mathbf{j}} = 0$ if $\mathbf{j}$ is negative in one coefficient.

Remarks. *(i) All the properties from above are easily adaptable for multivariate polynomials over multivariate simplices.*

(ii) The positivity property (vi) *is the reason why the polynomials are mainly useful for interpolation over the triangle $\triangle$.*

Now we can introduce the so-called Bezier representation of a polynomial surface.

Definition 2.13. *Every polynomial bivariate surface $b(x,y)$ has a unique* Bezier representation *with respect to a triangle $\triangle$. The representation is looking in the following way,*

$$b(x,y) = \sum_{i+j+k=n} b_{ijk} B^n_{ijk}(u,v,w). \tag{23}$$

These coefficients b_{ijk} are called the Bezier points *of the surface b and they are the vertices of the* Bezier net *of b over the triangle $\triangle$.*

Such a mapping $b : \triangle \to \mathbb{R}^s$ we call a **triangular Bezier patch** over the triangle $\triangle$ where the points b_{ijk} are in $\mathbb{R}^s$ and $s \in \mathbb{N}$. When we use triangular Bezier patches to interpolate a surface the choice of the Bezier points $b_{ijk} \in \mathbb{R}^s$ is not trivial especially if the surface should satisfy some continuity conditions between various triangles.

Proposition 2.14. *(i) For lower dimensional faces of the triangle $\triangle$ the Bezier points restricted to this face form a Bezier patch on this face. In particular, this implies at the vertices (a_1, a_2, a_3) of our triangle $\triangle$ that the Bezier patch touches the corresponding points in $\mathbb{R}^s$ of the vertices, i.e.*

$$b(a_1) = b_{n00},\ b(a_2) = b_{0n0},\ b(a_3) = b_{00n}.$$

Further this implies

$$D_{a_2-a_1} b(a_1) = n(b_{n-1,1,0} - b_{n,0,0})$$

where D_v means Differential operator in direction v.

(ii) The surface $b(\triangle)$ lies in the convex hull of its Bezier points.

Remarks. *(i) Again these properties are generalisable for arbitrary simplices.*

(ii) The property (i) *means for cubic triangular Bezier patches that if we know the function values and the values of the derivatives at the vertices of the triangle we get directly every Bezier point of the patch beside of b_{111}. This Bezier points we get are unique.*

(iii) The property (ii) *implies*

$$b(\triangle) \subseteq \left([\min_{\mathbf{i}=n} b^1_{\mathbf{i}}, \max_{\mathbf{i}=n} b^1_{\mathbf{i}}], \ \dots \ , [\min_{\mathbf{i}=n} b^s_{\mathbf{i}}, \max_{\mathbf{i}=n} b^s_{\mathbf{i}}] \right)^T \in \mathbb{IR}^s,$$

where the superscript j means the jth coefficient of the Bezier points.

Next we need to think about an efficient algorithm to determine a Bezier patch at some point $x \in \triangle$. Here the well-known De Casteljau algorithm for Bezier curves is generalisable for Bezier patches over a triangle. This algorithm uses iteratively the recursion formula (21).

Algorithm 1 (De Casteljau algorithm)
Determine $b(x,y) = \sum_{i+j+k=n} b_{ijk} B^n_{ijk}(u,v,w)$ Bezier patch over the triangle $\triangle$ where (u,v,w) are the barycentric coordinates with respect to $\triangle$ and $(x,y) \in \triangle$.

The Algorithm

- Set $b^0_{ijk} \leftarrow b_{ijk}$ where $i+j+k=n$.
- For $l = 1, \dots, n$

  $$b^l_{ijk} \leftarrow u b^{l-1}_{i+1,j,k} + v b^{l-1}_{i,j+1,k} + w b^{l-1}_{i,j,k+1}$$

  where $i+j+k = n-l$ and $i,j,k \geq 0$.
- Set $b(x,y) \leftarrow b^n_{000}$.

This algorithm gives us additionally the Bezier net of at most three subtriangles formed by their vertices (a_1, a_2, x), (x, a_2, a_3) and (a_1, x, a_3). Here $x = (x,y)$ from above. This observation is useful later. If x is on an edge of

the initial triangle, we only get two subtriangles. If x is chosen as a vertex point then the algorithm is useless. We see that iterative application of this algorithm on every sub-triangle yields to a subdivision of the initial triangle. This procedure is commonly named subdivision algorithm. The subdivision algorithm always applied at appropriate points is useful if we want to plot the Bezier patch. A good strategy of dividing the triangles is to choose the middlepoint on the longest edge of the triangle.
Now we have to think about how to apply these cubic triangular Bezier patches for our problem.

Joining two triangular cubic patches

We consider two triangles $\triangle$ and $\triangle'$ sharing an edge. These are determined by its vertices (a_1, a_2, a_3) respectively (a_4, a_2, a_3). We know that all Bezier points beside of the center Bezier points b_{111} respectively b'_{111} are uniquely determined by the function values and gradients given at the vertices. So we have to think of how to choose the Bezier points such that the corresponding patches b and b' have a C^1 joint along the sharing edge.
Due to Proposition 2.14 (i) we know that the patches b and b' restricted to the sharing edge must be Bezier patches again. Additionally we know that the Bezier points of b and b' corresponding to this edge are uniquely determined by the function values and gradients at the vertices a_2 and a_3. So we can conclude that the Bezier patches b and b' restricted to the sharing edge must agree. So we have a C^0 joint between the triangles $\triangle$ and $\triangle'$. In particular this means

$$b(0, s, 1-s) = b'(0, s, 1-s),$$

where $s \in [0, 1]$. Now we additionally know that all derivatives in direction $a_2 - a_3$ of the Bezier patches b and b' along the edge agree. So we have to consider an arbitrary derivative with respect to a direction which is non-tangential to the sharing edge of the Bezier patches b and b' along the sharing edge to find conditions for a C^1 joint between the patches. We can write down a formula for directional derivatives for Bezier patches. We consider a line $x(t) = p + tz$ where $v = a_1 z_1 + z_2 a_2 + z_3 a_3$. Due to the construction of the barycentric coordinates for a direction must hold $z_1 + z_2 + z_3 = 0$. Then we have

$$\begin{aligned} D_z b(p) &= \frac{d}{dt} b(x(t))\Big|_{t=0} = z_1 \frac{\partial}{\partial u} b + z_2 \frac{\partial}{\partial v} b + z_3 \frac{\partial}{\partial w} b = \\ &= \sum_{\mathbf{j}=n-1} c_{\mathbf{j}} B_{\mathbf{j}}^{n-1}, \end{aligned}$$

where

$$c_{\mathbf{j}} = n(z_1 b_{\mathbf{j}+e_1} + z_2 b_{\mathbf{j}+e_2} + z_3 b_{\mathbf{j}+e_3}).$$

Now we can choose an arbitrary direction non-tangential to the sharing edge. We choose the direction $a_2 - a_1$ or in barycentric coordinates $(-1, 1, 0)$ and insert this in our formula. We only need to compare the Bezier net ordinates of $D_{a_1-a_2}b$ and $D_{a_1-a_2}b'$ corresponding to Bernstein polynomials which are not dependent of u respectively u' because we are only interested in the behaviour on the common edge. These Bezier net ordinates are c_{020}, c_{011} and c_{002} respectively c'_{020}, c'_{011} and c'_{002}. By inserting in the formula we get

$$\begin{aligned} c_{020} &= -b_{120} + b_{030}, \\ c_{011} &= -b_{111} + b_{021}, \\ c_{002} &= -b_{102} + b_{012}. \end{aligned}$$

In the next we have to determine the barycentric coordinates with respect to $\triangle'$ of the chosen direction $a_1 - a_2$. We say (u, v, w) are the barycentric coordinates of a_1 with respect to the triangle $\triangle'$. This means $a_1 = ua_4 + va_2 + wa_3$. Then we have to consider $D_{-u,-v+1,-w}b'$ on the common edge. Again inserting in the formula above gives

$$\begin{aligned} c'_{020} &= -ub'_{120} + (-v+1)b'_{030} - wb'_{021}, \\ c'_{011} &= -ub'_{111} + (-v+1)b'_{021} - wb'_{012}, \\ c'_{002} &= -ub'_{102} + (-v+1)b'_{012} - wb'_{003}. \end{aligned}$$

Now we compare c_{020} with c'_{020}, c_{002} with c'_{002} and c_{011} with c'_{011}. Then we get the conditions for a C^1 joint on the sharing edge,

$$b_{120} - ub'_{120} - vb'_{030} - wb'_{021} = 0, \tag{24}$$
$$b_{111} - ub'_{111} - vb'_{021} - wb'_{012} = 0, \tag{25}$$
$$b_{102} - ub'_{102} - vb'_{012} - wb'_{003} = 0. \tag{26}$$

It is easy to show that beside of the equation (25) the conditions are always true. This is not surprising because in these equations no unknown appears. Geometrically the equations from above are conditions for coplanarity of triangles. We see that we have restricted the choice of the inner Bezier points with these conditions, although (25) does not imply a unique choice for the inner Bezier point.
One possibility to restrict the choice of the inner Bezier points more is to consider the second derivatives along the common edge. Farin gives in Farin [9] and Farin [6] a formula for arbitrary C^k joints between triangular Bezier patches.

Theorem 2.15. *Let b be a Bezier patch defined over the triangle $\triangle$ determined by its vertices a_1, a_2, a_3 and c a Bezier patch over the triangle $\triangle'$ determined by the vertices a_4, a_2, a_3. Let a_1 have the barycentric coordinates (u, v, w) with respect to $\triangle'$. Then a necessary and sufficient condition for a C^r joint between b and c along the common edge is*

$$b_{sjk} = c^s_{0jk}(u, v, w) \qquad (s = 0, 1, \ldots, r), \tag{27}$$

where $c^s_{\mathbf{i}}(\mathbf{u})$ means the notation from the De Casteljau algorithm applied at the point $\mathbf{u}$.

By applying this theorem we see that our previous determinations are true and we get two additional equations for a C^2 joint between b and b',

$$b_{210} - b'_{210}u^2 - b'_{030}v^2 - b'_{012}w^2 - 2(b'_{120}uv + b'_{111}uw + b'_{021}vw) = 0, \tag{28}$$
$$b_{201} - b'_{201}u^2 - b'_{021}v^2 - b'_{003}w^2 - 2(b'_{111}uv + b'_{102}uw + b'_{012}vw) = 0. \tag{29}$$

We see we get two additional equations and this yields to an overestimated system of linear equations. The interpolation techniques often aim to satisfy the C^1 condition and try to minimize the C^2-discontinuity.

The Clough-Tocher interpolation scheme

Now we are assuming that we have a given triangulation $\mathcal{T}$ based on a grid $\mathfrak{G}$ in the region $\tilde{\Lambda} \times \tilde{\Sigma}$ we want to study. For every vertex $p \in \mathfrak{G}$ in this triangulation we assume that we know an approximate corresponding function value (u^p, ξ^p) and approximate corresponding gradients. For the gradients we write $(u^p_\lambda, \xi^p_\lambda)$ and $(u^p_\sigma, \xi^p_\sigma)$. We know already that for a single Bezier patch on a triangle we only have to think about how to choose the corresponding Bezier point b_{111}, because all the other corresponding Bezier points are uniquely determined by the function values and gradients at the vertices of the triangle.
The Clough-Tocher method was introduced in Clough and Tocher [4]. In this method a triangle, here called macro-triangle, is split into three mini-triangles. The split is determined by a splitting point C(see figure 3). As you see we changed the denotion for simplicity to describe the following topics. For symmetry reasons we use always the barycenter of the macro-triangle as the splitting point C, although all the other inner points of the triangle would be appropriate too. Due to the construction of the barycentric coordinates the barycenter always has the barycentric coordinates $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ over the corresponding triangle. This choice is reasonable because the x and y coordinates of the Bezier points over a triangle $\triangle$ determined by its

vertices (a_1, a_2, a_3) reference to $b_{ijk}[x, y] = \frac{1}{n}(ia_1 + ja_2 + ka_3)$ where n means the degree of the applied Bezier patch (see figure 11). This means the data values we get over C agree with b_{111}.

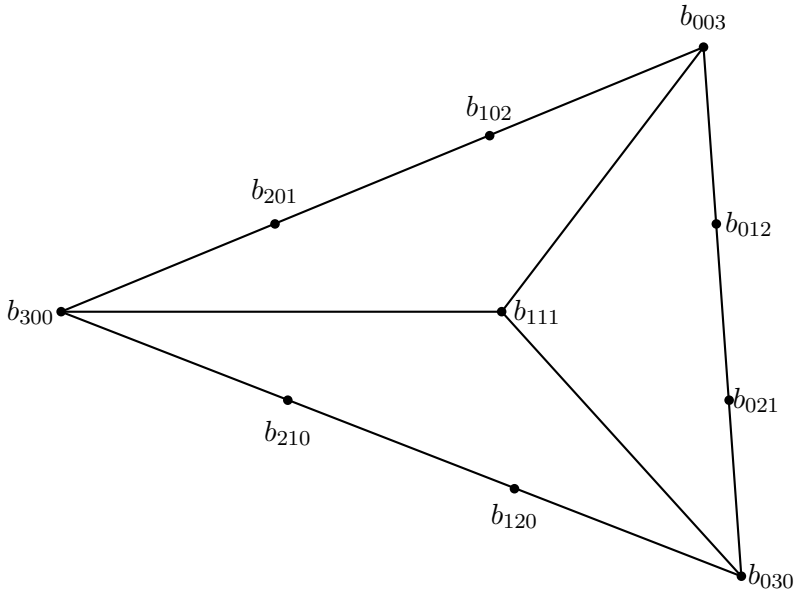

Figure 2: Bezier net in the triangular and cubic case

Now we present shortly the procedure of the Clough-Tocher scheme. We use the description of Mann [15]. For $ijk \in \{012, 120, 201\}$,

(a) Set V_i and T_{ij} like we have described above. To remember this choice is unique.

(b) Set I_{i1} to lie in the tangent plane at V_i. As a formula: $I_{i1} = \frac{1}{3}(V_i + T_{ij} + T_{ik})$.

(c) Now set C_i to be coplanar with T_{jk}, T_{kj} and C_i'. This choice has some freedom because this is exactly the C^1-joint condition. Clough and Tocher used a linear scheme for the choice. We use a modified scheme from Farin which minimizes the C^2 discontinuity and so should yield to a better result.

(d) Set I_{i2} to lie in the plane spanned by C_j, C_k and I_{i1}.

(e) Set S to lie in the plane spanned by I_{02}, I_{12} and I_{22}.

The method we use is named by Farin and is presented next.

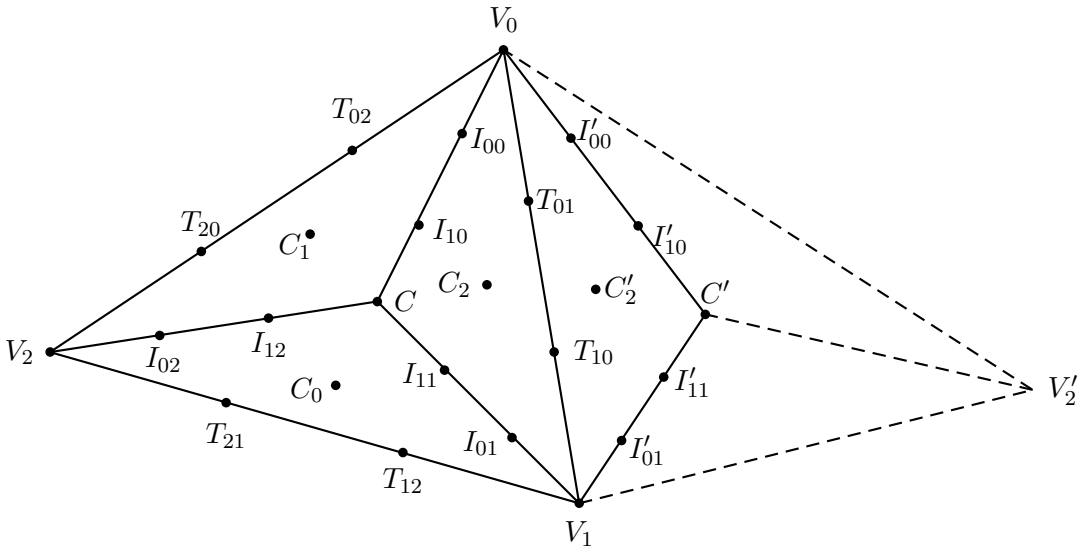

Figure 3: Clough-Tocher split

Farins method

Farins method yields to a C^1 joint and minimizes the C^2 discontinuities along the sharing edge of two neighbouring Bezier patches. This method yields to quadratic precision and improves the standard Clough-Tocher technique. The method uses the Clough-Tocher scheme beside of (c). So we replace (c) by the following steps.

(i) Set an initial C by

$$\frac{1}{4}(T_{20} + T_{02} + T_{10} + T_{01} + T_{21} + T_{12}) - \frac{1}{6}(V_0 + V_1 + V_2).$$

This formula was given in Farin [7] and ensures quadratic precision of the patch, but no C^1 joints between the sharing edges.

(ii) Next set the barycentric coordinates relative to the neighbouring triangles, i.e.

$$V_2 = u'V'_2 + v'V_1 + w'V_0,$$
$$V'_2 = uV_2 + vV_0 + wV_1.$$

(iii) Now use subdivision to get settings for C_i, I_{i1} and I_{i2} where $i = 0, 1, 2$. Here we use that due to our construction we get by applying the De

Casteljau algorithm at the barycenter $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ automatically the Bezier points of the mini-triangles. In particular, b^1_{110} corresponds to C_2 in our construction. Further we additionally see that b^1_{002} corresponds to I_{21}. In this way we get an initial setting for C_i and I_{i1}. By knowing this values we can get values for I_{i2}.

(iv) Next formulas are according to Mann [15]. The formula are obtained by minimizing the C^2 discontinuities with a standard Lagrange multiplier method. More detailed the least squares problem, solve (25) subject to minimizing the C^2 error equations (28) and (29) has to be solved. By using a standard least squares method Farin obtains the following formulas. Here we only write down the formulas and we refer for more details to Farin [8]. The notation in the formula is adapted to the currently used notation for Bezier patches.

$$\begin{aligned}
r_1 &= u'I'_{12} + v'I'_{11} - uI_{12} - wI_{11}, \\
r_2 &= u'I'_{02} + w'I'_{01} - uI_{02} - vI_{01}, \\
r_2 &= vT_{01} + wT_{10}, \\
a_{11} &= 2(v^2 + w^2), \\
a_{12} &= -2(vw' + wv'), \\
a_{22} &= 2(w'^2 + v'^2), \\
s_1 &= 2(vr_1 + wr_2), \\
s_2 &= -2(w'r_1 + v'r_2), \\
D &= 2ua_{12} + u^2a_{22} + a_{11}, \\
C'_2 &= \frac{us_1 + ua_{12}r_3 + u^2s_2 + r_3a_{11}}{D}, \\
C_2 &= \frac{C'_2 - vT_{01} - wT_{10}}{u}.
\end{aligned}$$

We can see that the last equation is exactly the C^1 condition from above. In an analogous way we can get values for C_0 and C_1.

Then continue with the Clough-Tocher scheme at (d). At the boundary of the area we are considering obviously we can not determine the C_i value adjacent to the boundary due to C^2 discontinuity minimization so we use the values we get for C_i due to the setting of the initial C. Maybe one can use the formula for the directional derivative of a patch to set the values for C_i corresponding to the boundary.

2.4 Existence and uniqueness theorems

Theorem 2.16 (Implicit function theorem)**.** *Let $f : \mathbb{R}^{n+m} \to \mathbb{R}^m$, $f(x, y)$ be a continuously differentiable function. Fix a point $(a_1, \ldots, a_n, b_1, \ldots, b_m) = (a, b) \in \mathbb{R}^{n+m}$ with $f(a, b) = c$ where $c \in \mathbb{R}^m$. If the matrix $D_y f(a, b) = \left(\frac{\partial f_i}{\partial y_j}(a, b) \right)_{(i,j)}$ where $i, j = 1, \ldots, , m$ is invertible, then there exists an open set $U \subseteq \mathbb{R}^n$ containing $a \in \mathbb{R}^n$, an open set $V \subseteq \mathbb{R}^m$ containing $b \in \mathbb{R}^m$, and a unique continuously differentiable function $g : U \to V$ such that*

$$f(x, g(x)) = c \text{ for all } x \in U. \tag{30}$$

Next we rephrase a theorem of Neumaier [18, p. 177] which gives us information about the existence of solution branches.

Theorem 2.17. *Let $G : D_0 \subseteq \mathbb{R}^n \to \mathbb{R}^n$ be Lipschitz continuous on $D \subseteq D_0$, and let $\mathbf{A} \in \mathbb{IR}^{n \times n}$ be a Lipschitz matrix for G on D. If $x \in \mathbf{x}_0 \in \mathbb{I}D$ then the following condition holds. If $x \in \operatorname{int}(\mathbf{x}_0)$ and $\emptyset \neq K(x, \mathbf{x}_0) \subseteq \operatorname{int}(\mathbf{x}_0)$ then G contains a unique zero in $\mathbf{x}_0$.*

The operator K is defined by

$$K(x, \mathbf{x}_0) = x - CG(x) - (C\mathbf{A} - I)(\mathbf{x}_0 - x) \tag{31}$$

and is called the **Krawczyk operator**.

Remark.
This theorem says that the analogue of a Newton-iteration step in interval analysis contracts.

Now we rephrase a theorem of Neumaier [18, p. 204] which gives us informations concerning the uniqueness of solution paths.

Theorem 2.18. *Let $G : \mathbb{R}^p \subseteq \Lambda \times \mathbb{R}^n \to \mathbb{R}^n$ be a Lipschitz continuous function, $(\lambda_0, x_0) \in \Lambda \times \mathbb{R}^n$ an approximate root, $D \subseteq \mathbb{R}^n, E \subseteq \Lambda$ closed and connected sets such that $D \times E \subseteq \Lambda \times \mathbb{R}^n$, and $\mathbf{A} : \mathbb{I}D \to \mathbb{IR}^{n \times n}$ is an interval function such that $\mathbf{A}(\mathbf{x})$ is a Lipschitz matrix of the function G fixed at λ for all $\mathbf{x} \in \mathbb{I}D, \lambda \in E$.*
Let $x_0 \in \mathbf{x}_0 \in \mathbb{I}D$ and suppose that

$$(C\mathbf{A}(\mathbf{x}) - I)(\mathbf{x}_0 - x_0) \subseteq \mathbf{d}, \tag{32}$$

$$\square \left\{ CG(\lambda, x_0) \;\middle|\; \lambda \in E \right\} \subseteq \mathbf{g}, \tag{33}$$

where $\mathbf{d}, \mathbf{g} \in \mathbb{IR}^n$ and $C \in \mathbb{R}^{n \times n}$ again a preconditioning matrix. Then

$$x_0 - \mathbf{g} - \mathbf{d} \subseteq \operatorname{int}(\mathbf{x}_0), \tag{34}$$

implies that, for all $\lambda \in E$, the equation $G(\lambda, x) = 0$ has a unique solution $x = H(\lambda) \in \mathbf{x}_0$, and the function $H : E \to \mathbb{R}^n$ defined in this way is continuous.
Moreover, the box $\mathbf{x}_1 = (x_0 - \mathbf{g} - \mathbf{d}) \cap \mathbf{x}_0$ is an enclosure of the solution set for all $\lambda \in E$.

Remark.
On closer inspection we see that Theorem 2.18 is just a straight-forward modification of Theorem 2.17. For a function $G : \mathbb{R}^p \subseteq \Lambda \times \mathbb{R}^n \to \mathbb{R}^n$, $G(\lambda, x)$ Theorem 2.18 says that if for all $\lambda \in E$, for the function $G_\lambda : \mathbb{R}^n \to \mathbb{R}^n$ where λ is held fixed the conditions in Theorem 2.17 hold then for all $\lambda \in E$ nearby a unique solution exists and one gets additionally the continuity of the so-constructed solution path.
In conclusion this means we do not have to prove the existence of a solution path separately.

3 Problem formulation

3.1 The reduction process

This section summarizes parts of Neumaier [19]. We first rephrase the fixed point equation (5),

$$\Phi_{\lambda,\sigma}\begin{pmatrix}u\\ \xi\end{pmatrix} := \begin{pmatrix}u\\ \xi\end{pmatrix} - A^{-1}\begin{pmatrix}F(\lambda,u)+A_1\xi\\ \mu(\lambda,u)-\sigma\end{pmatrix}.$$

Remark.
We repeat the proof of the following theorem which one can find in Neumaier [19] with very small changes because one can see that the proof also works with the weaker condition on the matrix A to have a left-inverse A^+ for which a bound $\alpha > 0$ for its norm can be found. If all columns of the matrix A are linear independent then the Moore-Penrose inverse is a left inverse of the matrix A. So the following discussion may works partly for this weaker condition. However we do not discuss this in detail.

Theorem 3.1. *Let $\delta \geq 0$ such that the ball*

$$B_\delta(u_0) = \left\{ u \in D \;\middle|\; \|u - u_0\| \leq \delta \right\}$$

is in $\operatorname{int}(D)$, *and for all $u \in B_\delta(u_0)$ we have*

$$\left\| \begin{pmatrix} D_u F(\lambda,u) - A_0 & 0 \\ D_u\mu(\lambda,u) - A_2 & 0 \end{pmatrix} \right\| \leq \frac{1}{2\alpha}. \tag{35}$$

If

$$\left\| \begin{pmatrix} F(\lambda,u_0) + A_1\xi_0 \\ \sigma - \mu(\lambda,u_0) \end{pmatrix} \right\| \leq \frac{\delta}{2\alpha}, \tag{36}$$

then $\Phi_{\lambda,\sigma}$ (see (5)) is a contraction in $B_\delta(u_0,\xi_0) = \left\{ (u,\xi) \;\middle|\; \left\| \begin{pmatrix} u-u_0 \\ \xi-\xi_0 \end{pmatrix} \right\| \leq \delta \right\}$.

Proof. Let $(u,\xi) \in B_\delta(u_0,\xi_0)$. Then the derivative of $\Phi_{\lambda,\sigma}$ with respect to (u,ξ) is

$$\Phi'_{\lambda,\sigma} = I - A^{-1}\begin{pmatrix} D_uF(\lambda,u) & A_1 \\ D_u\mu(\lambda,u) & 0 \end{pmatrix} = A^{-1}\begin{pmatrix} A_0 - D_uF(\lambda,u) & 0 \\ A_2 - D_u\mu\lambda,u) & 0 \end{pmatrix},$$

so that

$$\left\|\Phi'_{\lambda,\sigma}\right\| \leq \left\|A^{-1}\right\| \left\| \begin{pmatrix} A_0 - D_uF(\lambda,u) & 0 \\ A_2 - D_u\mu(\lambda,u) & 0 \end{pmatrix} \right\| \leq \frac{1}{2}.$$

Hence, by the mean value theorem

$$\left\|\Phi_{\lambda,\sigma}\begin{pmatrix}u\\ \xi\end{pmatrix}-\Phi_{\lambda,\sigma}\begin{pmatrix}v\\ \eta\end{pmatrix}\right\|\leq\frac{1}{2}\left\|\begin{pmatrix}u-v\\ \xi-\eta\end{pmatrix}\right\|\text{ for }\begin{pmatrix}u\\ \xi\end{pmatrix},\begin{pmatrix}v\\ \eta\end{pmatrix}\in B_\delta\begin{pmatrix}u_0\\ \xi_0\end{pmatrix}.$$

Thus $\Phi_{\lambda,\sigma}$ has a Lipschitz constant $\frac{1}{2}$. Moreover for the choice $v=u_0,\eta=\xi_0$, we find

$$\begin{aligned}\left\|\Phi_{\lambda,\sigma}\begin{pmatrix}u\\ \xi\end{pmatrix}-\begin{pmatrix}u_0\\ \xi_0\end{pmatrix}\right\| &\leq \frac{1}{2}\left\|\begin{pmatrix}u-u_0\\ \xi-\xi_0\end{pmatrix}\right\|+\left\|\Phi_{\lambda,\sigma}\begin{pmatrix}u_0\\ \xi_0\end{pmatrix}-\begin{pmatrix}u_0\\ \xi_0\end{pmatrix}\right\|\\ &\leq \frac{\delta}{2}+\left\|-A^{-1}\begin{pmatrix}F(\lambda,u_0)+A_1\xi_0\\ \sigma-\mu(\lambda,u_0)\end{pmatrix}\right\|\\ &\leq \frac{\delta}{2}+\left\|A^{-1}\right\|\left\|\begin{pmatrix}F(\lambda,u_0)+A_1\xi_0\\ \sigma-\mu(\lambda,u_0)\end{pmatrix}\right\|\leq\delta.\end{aligned}$$

Hence $\Phi_{\lambda,\sigma}$ is a contraction in $B_\delta(u_0,\xi_0)$ and has a unique fix point in $B_\delta(u_0,\xi_0)$ due to Banachs fixed point theorem. □

We apply this theorem in a box $(\boldsymbol{\lambda},\boldsymbol{\sigma})\subseteq\Lambda\times\mathbb{R}^l$. The verification then implies that for all $(\lambda,\sigma)\in(\boldsymbol{\lambda},\boldsymbol{\sigma})$ exists a point $(u,\xi)\in B_\delta(u_0,\xi_0)$ where $\xi_0=0$ for which $\tilde{F}(\lambda,\sigma,u,\xi)=0$.
We now consider the maximal simply connected, open set of $(\lambda,\sigma)\in\Lambda\times\mathbb{R}^l$ such that the mapping $\Phi_{\lambda,\sigma}$ is a contraction in some ball $B_\delta(u_0,\xi_0)$ with Σ. Further we write for these unique fixed point at $(\lambda,\sigma)\in\Lambda\times\mathbb{R}^l$ by $\begin{pmatrix}\tilde{u}(\lambda,\sigma)\\ \xi(\lambda,\sigma)\end{pmatrix}$.
The implicit function theorem (Theorem 2.16) now says that the mappings $\tilde{u}:\Sigma\to\mathbb{R}^n$ and $\xi:\Sigma\to\mathbb{R}^k$ defined in this way are k_0-times continuously differentiable and that they satisfy

$$F(\lambda,\tilde{u})+A_1\xi=0,\ \mu(\lambda,\tilde{u})=\sigma \tag{37}$$

in Σ. Now we have an implicit parametrization for the solution manifold M.

$$M=\left\{\begin{pmatrix}\lambda\\ u(\lambda,\sigma)\end{pmatrix}\;\middle|\;(\lambda,\sigma)\in\Sigma,\xi(\lambda,\sigma)=0\right\}. \tag{38}$$

Further is M the image under λ of a reduced manifold $\Sigma^*\in\Lambda\times\mathbb{R}^k$ given by

$$\Sigma^*=\left\{(\lambda,\sigma)\in\Sigma\;\middle|\;\xi(\lambda,\sigma)=0\right\}. \tag{39}$$

Next is a theorem which shows that the reduced manifold has the required topological property.

Theorem 3.2. *The mapping $\varphi : \Sigma \to \Lambda \times D$ defined by*

$$\varphi(\lambda, \sigma) = \begin{pmatrix} \lambda \\ \tilde{u}(\lambda, \sigma) \end{pmatrix}$$

is C^{k_0}-diffeomorphism to Σ^*.

The proof you can find in Neumaier [19].
We see that the singular behaviour of M is reflected in the reduced manifold Σ^*. In particular, the multi-valued function

$$\hat{\mu}(\lambda) = \left\{ \sigma \;\middle|\; (\lambda, \sigma) \in \Sigma,\, \xi(\lambda, \sigma) = 0 \right\} \tag{40}$$

describes the solution manifold.
Now we study what is to be observed by application of Theorem 3.1 and Theorem 3.2.
First we consider again our problem,

$$F : \Lambda \times D \subseteq \mathbb{R}^{p=1} \times \mathbb{R}^n \to \mathbb{R}^m, \quad F(\lambda, u) = 0, \tag{41}$$

where F is k_0-times continuously differentiable again and $n, k_0 \in \mathbb{N}$. If $p > 1$ we hope that our observations of the case $p = 1$ helps us to find easily a procedure for the $p > 1$ case. We discuss this case later. Now we start with the question of how to choose the matrix A.
Now we assume that we know an approximate zero $(\lambda_0, u_0) \in \Lambda \times D$ of the continuously differentiable function F where nearby is a (near) singularity. Then close to $D_u F(\lambda_0, u_0)$ exists a linear operator A_0 with $\dim \ker A_0 = m_0 > 0$. Then the following proposition helps us for a good choice of the linear operator A. We assume that A is chosen with the even mentioned A_0 and appropriate choices for the linear operators A_1 and A_2 such that A is a bijection and the norm of its inverse is bounded by $\alpha > 0$.

Proposition 3.3. *Let $v_1, \ldots, v_{m_0}$ be linearly independent null vectors of A_0, and let $w_1^*, \ldots, w_{m_0}^*$ be linearly independent null vectors of the adjoint $A_0^* : \mathbb{R}^m \to \mathbb{R}^n$.*

(i) If A is a bijection, then $A_2 v_1, \ldots, A_2 v_{m_0}$ are linearly independent and $A_1^* w_1^*, \ldots, A_1^* w_{m_0}^*$ are linearly independent.

(ii) If $m_0 = k$, then, conversely, these conditions imply that A is a bijection.

The proof you can find in Neumaier [19].
The conclusion of this proposition is that for the vectors $v_1, \ldots, v_{m_0} \in \mathbb{R}^n$ and $w_1, \ldots, w_{m_0} \in \mathbb{R}^m$ which span the invariant subspaces corresponding to the

small eigenvalues of $D_uF(\lambda_0, u_0)$ and its adjoint, A_1 and A_2 should be chosen such that the rank of the matrices $[A_2v_1, \ldots, A_2v_{m_0}]$ and $[A_1^*w_1, \ldots, A_1^*w_{m_0}]$ has rank m_0 and the rank is stable under perturbations of the size such that a corresponding eigenvalue gets zero.
Clearly we need $k \geq m_0$ to make the linear operator A a bijection and the number of unfolding functionals should be at least the algebraic number of small eigenvalues of $D_uF(\lambda_0, u_0) \in \mathbb{R}^{n\times n}$. If this is not true then $\alpha > 0$ is very large.
In common cases the choice of the linear operator A_2 depends on the unfolding functionals μ, that means $A_2 \approx D_u\mu(\lambda_0, u_0)$ and because of that the unfolding functionals μ must be chosen appropriate to use our procedure.
From now on we say $n = m$. This is the most common case we consider, although most of the discussion below can be adapted for the case $n \neq m$.
Now we assume that close to $(\lambda_0, u_0) \in \Lambda \times D$ which is a known approximate zero of the function F we have a point $(\bar{\lambda}, \bar{u}) \in \Lambda \times D$ such that for the matrix $\bar{A}_0 := D_uF(\bar{\lambda}, \bar{u})$,

$$\dim \ker \bar{A}_0 = m_0 = 1.$$

holds. In particular then the matrix $\bar{A}_0 \in \mathbb{R}^{n\times n}$ is singular. There shall be no further singular linear operator close to A_0 and $\bar{A}_0$. If we have $F(\bar{\lambda}, \bar{u}) = 0$ and $\dim \ker \bar{H}_0 = 1$ where $\bar{H}_0 := (D_\lambda F(\bar{\lambda}, \bar{u}), \bar{A}_0)$ then we have a turning point at $(\bar{\lambda}, \bar{u})$. If $\dim \ker \bar{H}_0 = 2$ then either a perfect or an imperfect bifurcation might occur. Now we assume that the more interesting latter case occurs.
For the following sections we want to reference mainly to Allgower and Georg [1].

3.2 The choice of A_1

We know that the matrix $\bar{A}_0$ is a singular and that means in particular that the corresponding linear operator $\bar{A}_0$ is not surjective. Due to the fundamental theorem of linear algebra we know that $\dim \operatorname{range} \bar{A}_0 = n - 1$ and in particular $\dim \mathbb{R}^n \backslash \operatorname{range} \bar{A}_0 = 1$. We also know that $(\operatorname{range} \bar{A}_0)^\perp = \ker \bar{A}_0^T$. So the mapping $\bar{A}_0u + A_1\xi = \begin{pmatrix} \bar{A}_0 & A_1 \end{pmatrix} \begin{pmatrix} u \\ \xi \end{pmatrix}$ is surjective when A_1 is chosen appropriately that means $0 \neq A_1 \in \ker \bar{A}_0^T$. So we know that a good choice of A_1 should be in $\ker \bar{A}_0^T$.
Unfortunately we do not know the exact point $(\bar{\lambda}, \bar{u}) \in \Lambda \times D$ where the singularity occur and so we have to try to approximate $\ker \bar{A}_0^T$ respectively $\ker \bar{H}_0^T$. We also can use $\ker \bar{H}_0$ because $A_1 \notin \operatorname{range} \bar{H}_0 \Rightarrow A_1 \notin \operatorname{range} \bar{A}_0$. So the problem we must solve is to determine an approximate vector $A_1 \in \mathbb{R}^n$

which approximately spans $\ker \bar{H}_0^T$ is

$$\min_e \left\{ \|H_0^* e\| \;\middle|\; \|e\| = 1 \right\}, \tag{42}$$

where $H_0 := \begin{pmatrix} D_\lambda F(\lambda_0, u_0) & A_0 \end{pmatrix}$. Here the norms mean the ℓ^2-norm or in the matrix-case the induced spectral norm. These norms are always meant in this section unless something else is said.
The solution of this problem is the eigenvector of $H_0 H_0^T$ corresponding to its absolute smallest eigenvalue.

3.3 The choice of A_2

Here we assume that no unfolding functional μ is given. Analogously to above we aim to find a vector $A_2 \in \mathbb{R}^n$ such that the mapping $\bar{A}_0^T v + A_2^T \eta : \mathbb{R}^{n+1} \to \mathbb{R}^n$ gets surjective because then we know that $\ker \begin{pmatrix} \bar{A}_0 \\ A_2 \end{pmatrix} = \{0\}$. This is exactly what we want to have. In an analogous way like above we get the problem

$$\min_\tau \left\{ \|A_0 \tau\| \;\middle|\; \|\tau\| = 1 \right\}. \tag{43}$$

The solution of this problem is the eigenvector corresponding to the absolute smallest eigenvalue of $A_0^T A_0$. Here we use the matrix $A_0 \in \mathbb{R}^{n \times n}$ instead of the matrix $H_0 \in \mathbb{R}^{n \times (n+1)}$ because we assume that the kernel of $\bar{H}_0$ has dimension two and so we do not know if we get the approximation which works to shrink the null space of the matrix $\bar{A}_0$.

3.4 The choice of A_1 with an additional assumption

Now we are assuming additionally that the functional $\mu : \mathbb{R}^{p=1} \times \mathbb{R}^n \to \mathbb{R}^{k=1}$ is given appropriately. We discuss later what appropriate means.
We consider the matrices

$$\begin{pmatrix} \bar{H}_0 \\ \bar{\mu}_{(\lambda,u)} \end{pmatrix} := \begin{pmatrix} \bar{F}_\lambda & \bar{A}_0 \\ \bar{\mu}_\lambda & \bar{\mu}_u \end{pmatrix} := \begin{pmatrix} D_\lambda F(\bar{\lambda}, \bar{u}) & D_u F(\bar{\lambda}, \bar{u}) \\ D_\lambda \mu(\bar{\lambda}, \bar{u}) & D_u \mu(\bar{\lambda}, \bar{u}) \end{pmatrix}$$

and $\begin{pmatrix} \bar{A}_0 \\ \bar{\mu}_u \end{pmatrix}$ instead of $\bar{H}_0$ and $\bar{A}_0$. Then $\begin{pmatrix} \bar{H}_0 \\ \bar{\mu}_{(\lambda,u)} \end{pmatrix}$ is a $(n+1) \times (n+1)$ matrix and if $\bar{\mu}_{(\lambda,u)}$ is not linearly dependent of the rows of $\bar{H}_0$ then

$$\dim \ker \begin{pmatrix} \bar{H}_0 \\ \bar{\mu}_{(\lambda,u)} \end{pmatrix} = 1 =: m_0.$$

Unfortunately we can not check the linear independence because we do not know the point $(\bar{\lambda}, \bar{u}) \in \Lambda \times D$. From now on we say the unfolding functional μ is chosen appropriately if the vector $\bar{\mu}_{(\lambda,u)}$ is linear independent of the rows of the matrix $\bar{H}_0$.
The next steps are very similar to the discussion above. We know that $\dim \mathbb{R}^n \setminus \operatorname{range} \begin{pmatrix} \bar{H}_0 \\ \bar{\mu}_{(\lambda,u)} \end{pmatrix} = 1$ and so we have to find analogously to above an approximation of $\ker \begin{pmatrix} \bar{H}_0 \\ \bar{\mu}_{(\lambda,u)} \end{pmatrix}^T$. Analogously the problem we must solve is

$$\min_e \left\{ \left\| \begin{pmatrix} H_0 \\ \mu_{(\lambda,u)}(\lambda_0, u_0) \end{pmatrix}^T e \right\| \;\middle|\; \|e\| = 1 \right\}, \tag{44}$$

and the solution is again the eigenvector corresponding to the absolute smallest eigenvalue of $\begin{pmatrix} H_0 \\ D_{(\lambda,u)}\mu(\lambda_0, u_0) \end{pmatrix} \begin{pmatrix} H_0 \\ D_{(\lambda,u)}\mu(\lambda_0, u_0) \end{pmatrix}^T$.
We see that the vector e is element of $\mathbb{R}^{n+1}$ but should be from $\mathbb{R}^n$. So we have to adapt our expanded problem to

$$\tilde{F}(\lambda, \sigma, u, \xi) = \begin{pmatrix} F(\lambda, u) + A_1'\xi \\ \mu(\lambda, u) - \sigma + A_1''\xi \end{pmatrix} \tag{45}$$

where $A_1 = (A_1',\, A_1'')^T := e$.
In this slightly adapted problem solutions with $\xi = 0$ are again solutions of our original problem and for the corresponding fixed point equation

$$\Phi_{\lambda,\sigma} \begin{pmatrix} u \\ \xi \end{pmatrix} := \begin{pmatrix} u \\ \xi \end{pmatrix} - A^{-1} \begin{pmatrix} F(\lambda, u) + A_1'\xi \\ \mu(\lambda, u) - \sigma + A_1''\xi \end{pmatrix} \tag{46}$$

Theorem 3.1 stays true with slight modifications. The condition (36) is looking now in the following way:

$$\left\| \begin{pmatrix} F(\lambda, u_0) + A_1'\xi_0 \\ \mu(\lambda, u_0) - \sigma + A_1''\xi_0 \end{pmatrix} \right\| \leq \frac{\delta}{2\alpha}. \tag{47}$$

The condition (35) does not change and the proof is easily to modify for this case.

3.5 Another ansatz to choose the extension of our problem

We assume that we know a LU-decomposition of the matrix A_0, in particular $PA_0 = LU$. Here P is the $n \times n$ pivoting matrix, L a $n \times n$ lower triangular

matrix with diagonal entries equal one and U is an $n \times n$ upper triangular matrix. By knowing this decomposition we consider a decomposition of the matrix A,

$$\begin{aligned}\begin{pmatrix} P & 0 \\ 0 & 1 \end{pmatrix} A &= \begin{pmatrix} PA_0 & PA_1 \\ A_2 & 0 \end{pmatrix} = \begin{pmatrix} LU & PA_1 \\ A_2 & 0 \end{pmatrix} = \\ &= \begin{pmatrix} L & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} U & L^{-1}PA_1 \\ A_2 & 0 \end{pmatrix} =: \tilde{L}\tilde{U}.\end{aligned}$$

We know that diagonal matrices like L and U are invertible if and only if their components on the diagonal are not equal zero. So we know that the matrix $\tilde{L}$ is invertible if L is too. We are assuming above that the matrix A_0 is invertible so also L must be invertible and as an implication $\tilde{L}$ too.
The matrix $U \in \mathbb{R}^{n\times n}$ is due to the same reasons invertible and we aim to choose $A_1 \in \mathbb{R}^n$ and $A_2 \in \mathbb{R}^n$ such that $\tilde{U} \in \mathbb{R}^{(n+1)\times(n+1)}$ stays invertible and is well-conditioned. We know that a diagonal matrix is well-conditioned if their diagonal entries are large compared with the entries in the same row and in the same column. So we choose $A_2 \in \mathbb{R}^n$ such that we try to correct the worst entry. We set

$$A_2^i = \begin{cases} \|U\| & \text{if } i = \underset{j}{\operatorname{argmin}} \left\{ \delta_j := \dfrac{|u_{jj}|}{\max_k \{|u_{jk}|, |u_{kj}|\}} \right\}, \\ 0 & \text{otherwise,} \end{cases} \tag{48}$$

where the superscript i means the ith component of A_2.
Here the norm is not the ℓ^2 - norm. The norm here shall be compatible with the chosen norms to verify the solution and in particular to determine the bound $\alpha > 0$ for the inverse of A.
Next we have to choose $A_1 \in \mathbb{R}^n$ in an appropriate way. Now we change the last column $\begin{pmatrix} L^{-1}PA_1 & 0 \end{pmatrix}^T$ with the column where the vector A_2 has the non-zero entry and consider this matrix. We can do this because such a transformation has no effect on the invertibility and well-conditioned of the matrix. Now we choose $A_1 \in \mathbb{R}^n$ such that the transformed matrix is a diagonal matrix and such that its diagonal entry is large compared with the entries in the same column and the same row. The entries in the same column we can choose and so we want them to be zero. The entry in the diagonal we choose to be $\|U\|$ again. So we get the linear equation system $L^{-1}PA_1 = 0$ in all components which are not equal to the component where $A_2 \in \mathbb{R}^n$ is zero and $L^{-1}PA_1 = \|U\|$ in the component where $A_2 \in \mathbb{R}^n$ non-zero. Easily we get the formula

$$A_1 = P^T L \left[0, \ldots, 0, \|U\|, 0, \cdots, 0\right]^T, \tag{49}$$

where the non-zero entry of the vector on the right-hand side is in the same entry as the non-zero entry of $A_2 \in \mathbb{R}^n$.
If we have some more $\delta_j \geq 0$ which are small then this indicates us that the initial problem has singularities or near singularities of higher degree than one. In this case we need an additional extension for the already extended problem to get a well-conditioned problem. This we can get by applying this procedure again.
Now we assume that we have a given unfolding functional μ with $A_2 := \mu(\lambda_0, u_0)$ or at least $A_2 \in \mathbb{R}^n$ determined with a method from above. Then we have $\tilde{A} := \begin{pmatrix} A_0 \\ A_2 \end{pmatrix}$ and we determine a LU-decomposition of $\tilde{A}$. That means

$$P\tilde{A} = LU,$$

where P is the $n+1 \times n+1$ pivoting matrix, L is a $n+1 \times n$ lower triangular matrix with $l_{ii} = 1 \ \forall i = 1, \ldots, n$ and U an upper triangular $n \times n$ matrix. For L there is only one choice of extension possible:

$$\tilde{L} := \begin{pmatrix} L & e_{n+1} \end{pmatrix},$$

where e_k is the kth unit vector.
We consider

$$\begin{aligned} P \begin{pmatrix} A_0 & A_1 \\ A_2 & 0 \end{pmatrix} &= \begin{pmatrix} P \begin{pmatrix} A_0 \\ A_2 \end{pmatrix} & P \begin{pmatrix} A_1 \\ 0 \end{pmatrix} \end{pmatrix} = \\ &= \begin{pmatrix} LU & P \begin{pmatrix} A_1 \\ 0 \end{pmatrix} \end{pmatrix} = \tilde{L} \begin{pmatrix} U & \tilde{L}^{-1} P \begin{pmatrix} A_1 \\ 0 \end{pmatrix} \end{pmatrix}. \end{aligned}$$

Due to equivalent argumentation like above we want $A_1 \in \mathbb{R}^n$ such that the $(n+1)$th entry of $\tilde{L}^{-1} P \begin{pmatrix} A_1 \\ 0 \end{pmatrix}$ is equal to $\|U\|$ and any other entry shall be zero. The system of linear equation which arises of this condition is in general not solvable because the matrix $\tilde{L}^{-1}P$ is nonsingular. Anyway we consider the formula

$$P^T \tilde{L} \begin{pmatrix} 0 \\ \|U\| \end{pmatrix} = \begin{pmatrix} A_1 \\ 0 \end{pmatrix}. \tag{50}$$

We know the pivoting matrix P^T is changing the rows of the lower diagonal matrix $\tilde{L}$. So we see if the lowest row of $\tilde{L}$ is moved by the pivoting matrix P^T then we really get zero at the right-hand side of the formula in the last entry of the vector. Equivalent to the condition of removing the lowest row to another entry is that the entry $p_{(n+1)(n+1)}$ of P must equal zero. If we have $p_{(n+1)(n+1)} = 1$ we need one more non-zero entry in the vector on the

left-hand side of the formula above too. We write $x \in \mathbb{R}^n$ for this vector. Now we consider the entry where the last column of $\tilde{L}$ respectively L has its maximum absolute value, because we want the additional non-zero entry in $x \in \mathbb{R}^n$ to be small. Let $(n+1, i)$ be the index of this entry in the matrix L. Then we set the further non-zero entry of the vector in the formula on the left-hand side by

$$x_i = -\frac{\|U\|}{l_{(n+1),i}}.$$

Now we get $A_1 \in \mathbb{R}^n$ by the formula

$$P^T \tilde{L} x = \begin{pmatrix} A_1 \\ 0 \end{pmatrix}. \tag{51}$$

Using this ansatz we automatically get a LU-decomposition of the matrix A. The matrix U gives us some hints about the condition of the extended problem we get. If we have some $\delta_i \geq 0$ defined like above which are small then either the unfolding functional μ might not be appropriately chosen and we have to find a different one or the initial problem might have singularities or near singularities of higher degree.

3.6 The $m_0 = 2$ case

Now we assume that close to $(\lambda_0, u_0) \in \Lambda \times D$ we have a point $(\bar{\lambda}, \bar{u}) \in \Lambda \times D$ where $\bar{A}_0 := D_u F(\bar{\lambda}, \bar{u})$ is a singular matrix and

$$\dim \ker \bar{A}_0 = m_0 = 2.$$

There shall be no further singular linear operator close to the linear operators A_0 and $\bar{A}_0$. If we have $F(\bar{\lambda}, \bar{u}) = 0$ and $\dim \ker \bar{H}_0 = 2$ where $\bar{H}_0 := \begin{pmatrix} D_\lambda F(\bar{\lambda}, \bar{u}) & \bar{A}_0 \end{pmatrix}$ then we probably have intuitively a simple bifurcation point with additionally turning point behaviour with respect to λ at $(\bar{\lambda}, \bar{u})$. If $\dim \ker \bar{H}_0 = 3$ then there might be a perfect or an imperfect bifurcation of higher order. Now we assume that the latter case occurs.
Exemplary we try to find a good choice for the linear operator A_1. With the same arguments as above is $\dim \operatorname{range} \bar{A}_0 = n - 2$. Obviously then we need two vectors $A_1' \in \mathbb{R}^n$ and $A_1'' \in \mathbb{R}^n$ to make the linear mapping $\bar{A}_0 u + A_1' \xi' + A_1'' \xi'' : \mathbb{R}^n \times \mathbb{R}^2 \to \mathbb{R}^n$ surjective. Ideally these vectors should be element of $\ker \bar{A}_0^T = (\operatorname{range} \bar{A}_0)^\perp$ and these space has exactly dimension 2 and so is spanned by two vectors. Now we set $A_1 = \begin{pmatrix} A_1' & A_1'' \end{pmatrix}$ and $\xi = \begin{pmatrix} \xi' \\ \xi'' \end{pmatrix}$. There is $v \notin \operatorname{range} \bar{H}_0 \Rightarrow v \notin \operatorname{range} \bar{A}_0$ true so that we get the known problem

again to approximate the kernel of $\bar{A}_0^T$ (respectively $\bar{H}_0^T$)

$$\min_e \left\{ \left\| H_0^T e \right\| \;\middle|\; \|e\| = 1 \right\},$$

where we already know how to find a solution.
Here we expect to find two vectors which approximately span the kernel of $\bar{A}_0^T$. Now we can either choose the two eigenvectors corresponding to the two smallest eigenvalues of $H_0 H_0^T$ or maybe we take at first one solution (the eigenvector corresponding absolute smallest eigenvalue) like above. Then we call this vector A_1'. Next we consider the problem

$$\min_e \left\{ \left\| \begin{pmatrix} H_0^T \\ A_1'^T \end{pmatrix} e \right\| \;\middle|\; \|e\| = 1 \right\}.$$

The solution of this problem shall get us an approximate nullvector of $\bar{H}_0^T$ which is additionally approximately orthogonal to the vector $A_1' \in \mathbb{R}^n$ we have found previously.
In the same way we modify the methods described above for the choice of the linear operator A_2 or (maybe) the choice of $A_1 \in \mathbb{R}^n$ when the unfolding functional μ is given.
We see that we do not need difficult modifications for the choice of $A_1 \in \mathbb{R}^n$ and $A_2 \in \mathbb{R}^n$ when $m_0 = 2$. If even $m_0 > 2$ the method to find the matrices A_1 and A_2 stays the same. There is to remark that if the unfolding functional $\mu : \mathbb{R}^p \times \mathbb{R}^n \to \mathbb{R}^l$ is given and $l < m_0$ then additional unfolding functionals must be found such that the modified functional μ maps into $\mathbb{R}^l$ with $l \geq m_0$. However, in general we have no information about the kind of singularity and we assume at first that a simple bifurcation occurs.

3.7 What happens when $p > 1$

We say now that $p = 2$ and $m_0 = 1$ again and the other assumptions stay the same. Then there are the possibilities that $\dim \ker \bar{H}_0$ is either 2 or 3. In the first case we do not have any bifurcation in these region. Our technique should work anyway like in all cases when singularities of lower degree occur. When $\dim \ker \bar{H}_0 = 3$ we need an appropriate linear operator $A_1 : \mathbb{R}^l \to \mathbb{R}^n$ and an appropriate unfolding functional $\mu : \mathbb{R}^p \times \mathbb{R}^n \to \mathbb{R}^l$ (respectively $A_2 : \mathbb{R}^n \to \mathbb{R}^l$). The choice is in analogous way possible as in the $p = 1$ case. The 'bifurcation diagram' in this case is 3-dimensional.
Now we look at the case when $p = 2$ and $m_0 = 2$. Here are three possibilities: either is $\dim \ker \bar{H}_0$ equal to 2, 3 or 4. When we have $\dim \ker \bar{H}_0 = 2$ again no bifurcation occurs. In the other situations we have bifurcations. In general

we do not know which situation is present so we have to assume the worst case. This means $\dim \ker \bar{H}_0 = 4$. Here the linear operator A_1 must map from $\mathbb{R}^2$ into the $\mathbb{R}^n$ and analogously the unfolding functional μ from $\mathbb{R}^p \times \mathbb{R}^n$ into $\mathbb{R}^2$. The determination of the vectors A_1 and A_2 can easily be adapted from above for this case. If the extension of the considered problem is appropriate then the matrix A must not be singular.
So theoretically the technique works for $p > 1$ and $m_0 > 1$ the same as for $p = 1$ and $m_0 = 1$. However we should try to keep the dimension of the vector space of σ as small as possible, because in the following we try to show that the fixed point equation (5) is a contraction in some region for all $(\lambda, \sigma) \in (\boldsymbol{\lambda}, \boldsymbol{\sigma})$ where $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$. This task is much more difficult if the dimension of the box is high because the volume of this box increases with the degree of the dimension of the box. However the dimension of the vector space of λ is given so we should try to avoid a high dimension for the vector space of σ.

3.8 A predictor-corrector method

We assume for simplicity the problem,

$$F : \mathbb{R}^p \times \mathbb{R}^n \supseteq \Lambda \times D \to \mathbb{R}^n, \quad F(\lambda, u) = 0. \tag{52}$$

Then an ordinary Euler-Newton-Method to solve (52) can be sketched shortly like this:

$$\tilde{x}_i = x_{i-1} + ht \qquad (\text{The predictor step}), \tag{53}$$

$$x_i = \tilde{x}_{i-1} - BF(\tilde{x}_i) \qquad (\text{The corrector step}), \tag{54}$$

where $x_i = (\lambda_i, u_i)$ respectively $\tilde{x}_i = (\tilde{\lambda}_i, \tilde{u}_i)$, $B = D_{(\lambda,u)}F(\tilde{x}_i)^+$ or in the Newton-chord case $B = D_{(\lambda,u)}F(x_{i-1})^+$. Here $D_{(\lambda,u)}F(x_{i-1})^+$ means the Moore-Penrose inverse of $D_{(\lambda,u)}F(x_{i-1})$. Further $h \in \mathbb{R}$ is the step-length and $t \in \mathbb{R}^n$ is the tangent vector at (λ_{i-1}, u_{i-1}) The vector is unique if and only if $\operatorname{rank} F_{(\lambda,u)}(x_{i-1}) = n$. The full rank condition means that no bifurcation point is at the point $(\lambda_{i-1}, u_{i-1}) \in \mathbb{R}^{p+n}$. Usually this conditions is true.
Now we make some assumptions. The box we want to study we call $(\boldsymbol{\lambda}, \mathbf{u}) \in \mathbb{IR}^{p+n}$ and we assume that we know an approximate root $(\lambda_0, u_0) \in (\boldsymbol{\lambda}, \mathbf{u})$ of the function $F : \Lambda \times D \to \mathbb{R}^n$. Then the point $(\lambda_0, \sigma_0, u_0, \xi_0)$ is an approximate root of the extended function $\tilde{F}$ where $\sigma_0 = \mu(\lambda_0, u_0)$ and $\xi_0 = 0$. Clearly we again mean by μ the previously invented unfolding functional.

We aim to find more approximate roots $(\tilde{\lambda}_0, \tilde{u}_0) \in (\boldsymbol{\lambda}, \mathbf{u})$. Our construction of the extended problem leads to find approximations for $\begin{pmatrix}\tilde{u}\\ \xi\end{pmatrix}(\tilde{\lambda}_0, \tilde{\sigma}_0)$ (see section 3.1) where $\tilde{\lambda}_0 \in \boldsymbol{\lambda}$ and $\tilde{\sigma}_0 \in \boldsymbol{\sigma} \in \mathbb{IR}^l$ and where $\boldsymbol{\sigma} \supseteq \mu(\boldsymbol{\lambda}, \mathbf{u})$.

The predictor step

When we assume that the function $\tilde{F}$ has no singularities in the box we have to study and due to $\tilde{F}$ is C^1 the mapping $\begin{pmatrix}\tilde{u}\\ \xi\end{pmatrix} : \mathbb{R}^2 \to \mathbb{R}^{n+1}$ which satisfy $\tilde{F}(\lambda, \sigma, \tilde{u}(\lambda, \sigma), \xi(\lambda, \sigma)) = 0$ for all $(\lambda, \sigma) \in (\boldsymbol{\lambda}, \boldsymbol{\sigma})$ is again C^1. Then the derivative functions $\begin{pmatrix}\tilde{u}\\ \xi\end{pmatrix}_\lambda$ and $\begin{pmatrix}\tilde{u}\\ \xi\end{pmatrix}_\sigma$ exist and are continuous in the whole box.
When we differentiate $\tilde{F}(\lambda, \sigma, \tilde{u}(\lambda, \sigma), \xi(\lambda, \sigma))$ with respect to λ where the function $\begin{pmatrix}\tilde{u}\\ \xi\end{pmatrix}(\lambda, \sigma)$ is defined like above we get

$$\begin{pmatrix} D_u F(\lambda, \tilde{u}(\lambda, \sigma)) & A_1 \\ D_u \mu(\lambda, \tilde{u}(\lambda, \sigma)) & 0 \end{pmatrix} \begin{pmatrix} \tilde{u}_\lambda(\lambda, \sigma) \\ \xi_\lambda(\lambda, \sigma) \end{pmatrix} + \begin{pmatrix} D_\lambda F(\lambda, \tilde{u}(\lambda, \sigma)) \\ D_\lambda \mu(\lambda, \tilde{u}(\lambda, \sigma)) \end{pmatrix} = 0.$$

Due to the definition of the mapping $\begin{pmatrix}\tilde{u}\\ \xi\end{pmatrix}$ we get

$$\begin{pmatrix} D_u F(\lambda, \tilde{u}(\lambda, \sigma)) & A_1 \\ D_u \mu(\lambda, \tilde{u}(\lambda, \sigma)) & 0 \end{pmatrix} \begin{pmatrix} \tilde{u}_\lambda(\lambda, \sigma) \\ \xi_\lambda(\lambda, \sigma) \end{pmatrix} = - \begin{pmatrix} D_\lambda F(\lambda, \tilde{u}(\lambda, \sigma)) \\ D_\lambda \mu(\lambda, \tilde{u}(\lambda, \sigma)) \end{pmatrix}. \tag{55}$$

This linear system of equation holds $\forall (\lambda, \sigma) \in L \times S$. In analogous way we get the the linear system of equations

$$\begin{pmatrix} D_u F(\lambda, \tilde{u}(\lambda, \sigma)) & A_1 \\ D_u \mu(\lambda, \tilde{u}(\lambda, \sigma)) & 0 \end{pmatrix} \begin{pmatrix} \tilde{u}_\sigma(\lambda, \sigma) \\ \xi_\sigma(\lambda, \sigma) \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{56}$$

We want to use these linear system of equations (55) and (56) for the predictor step. We need both vectors for the predictor step because contrarily to the sketched Euler-Newton where roots of a function $F : \mathbb{R}^{p=1} \times \mathbb{R}^n \to \mathbb{R}^n$ have to be found our problem maps from $\mathbb{R}^{p=2} \times \mathbb{R}^{n+1}$ to $\mathbb{R}^{n+1}$ and due to our assumptions the kernel of the Jacobi-Matrix near roots which are situated in the studied box $(\boldsymbol{\lambda}, \boldsymbol{\sigma})$ must have dimension 2. Then the kernel of this Jacobi-Matrix is spanned by the vectors $(1, 0, \tilde{u}_\lambda(\lambda, \sigma), \xi_\lambda(\lambda, \sigma))^T$ and $(0, 1, \tilde{u}_\sigma(\lambda, \sigma), \xi_\sigma(\lambda, \sigma))^T$. Further the matrix in (56) is nonsingular for all $(\lambda, \sigma) \in L \times S$ due to our assumptions and so the solution of the system of

linear equations (55) is unique for all $(\lambda, \sigma) \in (\boldsymbol{\lambda}, \boldsymbol{\sigma})$. Analogously the same is true for (56). To remark the right-hand side in (55) can be $0 \in \mathbb{R}^{n+1}$ and then the solution $(u_\lambda, \xi_\lambda) \in \mathbb{R}^2$ at this point naturally is $0 \in \mathbb{R}^{n+1}$.
Then a predictor step is

$$\begin{pmatrix} u^i \\ \xi^i \end{pmatrix} = \begin{pmatrix} u^{i-1} \\ \xi^{i-1} \end{pmatrix} + h_1 \begin{pmatrix} u_\lambda^{i-1} \\ \xi_\lambda^{i-1} \end{pmatrix} + h_2 \begin{pmatrix} u_\sigma^{i-1} \\ \xi_\sigma^{i-1} \end{pmatrix}, \tag{57}$$

where $\begin{pmatrix} u^i \\ \xi^i \end{pmatrix}$ approximates $\begin{pmatrix} \tilde{u}(\lambda_i, \sigma_i) \\ \xi(\lambda_i, \sigma_i) \end{pmatrix}$ for some $(\lambda_i, \sigma_i) \in (\boldsymbol{\lambda}, \boldsymbol{\sigma})$ and $\begin{pmatrix} u_\lambda^i \\ \xi_\lambda^i \end{pmatrix}$ respectively $\begin{pmatrix} u_\sigma^i \\ \xi_\sigma^i \end{pmatrix}$ approximates the derivative of the mapping $\begin{pmatrix} \tilde{u} \\ \xi \end{pmatrix}$ with respect to λ respectively σ at the point $(\lambda_i, \sigma_i) \in (\boldsymbol{\lambda}, \boldsymbol{\sigma})$. The reel values $h_i \, (i = 1, 2)$ are denoting step lengths. Like in every such method we have to care that the step length is not too large.
We also see that behind h_1 and h_2 hides the λ and σ direction. So if we know approximately $\begin{pmatrix} \tilde{u} \\ \xi \end{pmatrix} (\lambda_0, \sigma_0)$ and we want to use this predictor-corrector method to find an approximation for $\begin{pmatrix} \tilde{u}(\lambda_0', \sigma_0') \\ \xi(\lambda_0', \sigma_0') \end{pmatrix}$ then

$$\sum_{j=1}^{k} h_1^j = \lambda_0' - \lambda_0 \quad \text{and} \quad \sum_{j=1}^{k} h_2^j = \sigma_0' - \sigma_0 \tag{58}$$

have to hold. Here $k \in \mathbb{N}$ is the number of predictor steps which are used and $h_i^j \geq 0$ means the steplength of $h_i \geq 0$ in the jth predictor step $(i = 1, 2)$.
If we have $p > 1$ or $l > 1$ then λ and σ are p-dimensional vectors (respectively l-dimensional vectors). Then the linear systems of equations are

$$\begin{pmatrix} D_u F(\lambda, \tilde{u}(\lambda, \sigma)) & A_1 \\ D_u \mu_u(\lambda, \tilde{u}(\lambda, \sigma)) & 0 \end{pmatrix} \begin{pmatrix} \tilde{u}_{\lambda_r}(\lambda, \sigma) \\ \xi_{\lambda_r}(\lambda, \sigma) \end{pmatrix} = - \begin{pmatrix} D_{\lambda_r} F(\lambda, \tilde{u}(\lambda, \sigma)) \\ D_{\lambda_r} \mu(\lambda, \tilde{u}(\lambda, \sigma)) \end{pmatrix}$$

and

$$\begin{pmatrix} D_u F(\lambda, u(\lambda, \sigma)) & A_1 \\ D_u \mu(\lambda, u(\lambda, \sigma)) & 0 \end{pmatrix} \begin{pmatrix} u_{\sigma_r}(\lambda, \sigma) \\ \xi_{\sigma_r}(\lambda, \sigma) \end{pmatrix} = \begin{pmatrix} 0 \\ e_r \end{pmatrix},$$

where $r \in \mathbb{N}$ is the rth coefficient of the vector in $\mathbb{R}^l$ and e_r means the rth unit vector in $\mathbb{R}^l$. Obviously the steplengths h_1 and h_2 must be vectors in $\mathbb{R}^p$ now. At least the conditions of (58) must hold in every coefficient of λ respectively σ.
Some discussion on how to choose appropriate steplengths you can find in Allgower and Georg [1].

The corrector step

Due to condition (58) we do not want to change the λ and σ coordinate in the corrector step. So we consider

$$\tilde{F}_{\lambda,\sigma} : \mathbb{R}^{n+1} \to \mathbb{R}^{n+1}, \quad \tilde{F}_{\lambda,\sigma}(u,\xi) := \tilde{F}(\lambda,\sigma,u,\xi), \tag{59}$$

where we assume $\lambda \in \mathbb{R}^p$ and $\sigma \in \mathbb{R}^l$ fixed.
Then we can use the probably previously determined matrix A^{-1} in the Newton step. This matrix approximates $D_{(u,\xi)}\tilde{F}(\lambda_0,\sigma_0,u_0,\xi_0)^{-1}$ and we assume here that due to our construction $A^{-1} \approx D_{(u,\xi)}\tilde{F}(\lambda,\sigma,u,\xi)^{-1}$ holds more or less for all $(\lambda,\sigma) \in (\boldsymbol{\lambda},\boldsymbol{\sigma})$ and for all $u \in \mathbf{u}$. If this assumption is not true we have to consider a smaller box and to compute a new extension for our originally problem. Now we can set the corrector step,

$$\begin{pmatrix} u^i \\ \xi^i \end{pmatrix} = \begin{pmatrix} \tilde{u}^i \\ \tilde{\xi}^i \end{pmatrix} - A^{-1}\tilde{F}_{\lambda_i,\sigma_i}(\tilde{u}^i,\tilde{\xi}^i). \tag{60}$$

Maybe we should iterate this step more often because in general the matrix A^{-1} is no very good approximation of $F_u(\lambda_i,\sigma_i,\tilde{u}^i,\tilde{\xi}^i)^{-1}$ and so the convergence is slower. Then we write for the corrector iteration equation

$$\Psi_{\lambda,\sigma}\begin{pmatrix} v \\ \eta \end{pmatrix} = \begin{pmatrix} v \\ \eta \end{pmatrix} - A^{-1}\tilde{F}_{\lambda,\sigma}(v,\eta). \tag{61}$$

The corrector iteration is initialised with $(\tilde{u}^i,\tilde{\xi}^i)$ which probably was determined with a previously described predictor step. We use the already determined approximation A^{-1} because in general it is expensive to compute an inverse in every step. However, we could try probably to get a better approximation with rank-one updates of A^{-1}.

Remark.
We often have the task to determine $\begin{pmatrix} \tilde{u} \\ \xi \end{pmatrix}(\lambda,\sigma)$ *for some* $(\lambda,\sigma) \in (\boldsymbol{\lambda},\boldsymbol{\sigma})$ *and we know a good approximation to initialise the corrector iteration such that we do not need a predictor step. However one has to be aware of the fact that a corrector iteration changes in particular the vector* $u \in \mathbb{R}^n$ *and this implies a change of the* $\sigma \in \mathbb{R}^l$ *value (vector) because* $\sigma = \mu(\lambda,u)$*. So one has to care that the* σ *value (vector) does not change too much and maybe correct the error with a correcting predictor step.*

4 The verification process

This section is based on the previous sections and tries to apply the results. In particular we want to reference for this section to Kearfott [12].
In this section we apply the previous discussion to provide algorithms which shall give us rigorously verified solution paths with respect to λ and σ. Additionally we want to get enclosures where bifurcation or near bifurcation points occur. To remind we consider a $k_0 \geq 1$ times continuously differentiable nonlinear mapping,

$$F : \Lambda \times D \subseteq \mathbb{R}^p \times \mathbb{R}^n \to \mathbb{R}^m, \quad F(\lambda, u),$$

where we want to find enclosures for all roots in a bounded area $\tilde{\Lambda} \times \tilde{D} \subseteq \Lambda \times D$. We assume that we know additionally the derivatives of F with respect to u and λ. Here in this section we say that $n = m$. Further for the either given or determined unfolding functional $\mu : \mathbb{R}^p \times \mathbb{R}^n \to \mathbb{R}^l$ and the linear operator $A_1 : \mathbb{R}^l \to \mathbb{R}^n$.
During this section all norms are assumed to be the common maximum norm or the induced matrix norm, the absolute row sum. This assumption we make for simplicity reasons. If we would use weighted maximum norms then the changes in the discussion are small (see section 2.2).

4.1 Verifying a box

We start with how to apply Theorem 3.1 in our context. First we rewrite (36),

$$\|F(\lambda, u_0) + A_1\xi_0\| \leq \frac{\delta}{2\alpha}, \qquad \|\sigma - \mu(\lambda, u_0)\| \leq \frac{\delta}{2\alpha}.$$

We use these conditions for determining an adequate value for $\delta \geq 0$. By considering the condition (35) we see that a small value for $\delta \geq 0$ provides better chances to verify a solution box using Theorem 3.1.
Here we need the results of section 2.1 to compute good enclosures of the range of the terms on the left-hand side of the inequations above.

Algorithm 2 (Analysing a single box)
Find interval vectors $\mathbf{u} \in \mathbb{IR}^n$ and $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$ such that the following condition holds, $\forall(\lambda, \sigma) \in (\boldsymbol{\lambda}, \boldsymbol{\sigma})\ \exists!(u, \xi) \in (\mathbf{u}, \boldsymbol{\xi})$ with a suitable box $\boldsymbol{\xi} \in \mathbb{IR}^l$ such that $\tilde{F}_{\lambda,\sigma}(u, \xi) = 0$.

INPUT

- The interval vectors $\boldsymbol{\lambda} \in \mathbb{IR}^p, \boldsymbol{\sigma} \in \mathbb{R}^l$

- An initial approximate solution $(\lambda_0, \sigma_0, u_0, \xi_0)$ of the extended problem.

OUTPUT

- A boolean variable b_1 which is true when the verification was successful.
- A boolean variable b_2 which is true when the verification was successful and $0 \notin \boldsymbol{\xi}$.
- The interval box $\mathbf{u} \in \mathbb{IR}^n$.

THE ALGORITHM

- Set b_1, b_2 as false.
- Set δ.

$$\delta \leftarrow 2\alpha \max\left\{\sup\left(\left\|F(\boldsymbol{\lambda}, u_0) + A_1\xi_0\right\|, \left\|\mu(\boldsymbol{\lambda}, u_0) - \boldsymbol{\sigma}\right\|\right)\right\}$$

- IF $\|\xi_0\| > \delta$
  - Set $\mathbf{u}$.

$$\mathbf{u} \leftarrow \mathrm{midrad}\left(u_0, \delta\right).$$

  - Check the condition (35),

$$\left\|\begin{pmatrix} D_u F(\boldsymbol{\lambda}, \mathbf{u}) - A_0 & 0 \\ D_u \mu(\boldsymbol{\lambda}, \mathbf{u}) - A_2 & 0 \end{pmatrix}\right\| \leq \frac{1}{2\alpha}.$$

    IF the check is successful
    - Set $b_1 \leftarrow \mathrm{true}$ and $b_2 \leftarrow \mathrm{true}$.
- ELSE (Set $\xi_0 = 0$).
  - Reset δ.

$$\delta \leftarrow 2\alpha \max\left\{\sup\left(\left\|F(\boldsymbol{\lambda}, u_0)\right\|, \left\|\mu(\boldsymbol{\lambda}, u_0) - \boldsymbol{\sigma}\right\|\right)\right\}$$

  - Set $\mathbf{u}$.

$$\mathbf{u} \leftarrow \mathrm{midrad}\left(u_0, \delta\right).$$

  - Check the condition (35),

$$\left\|\begin{pmatrix} D_u F(\boldsymbol{\lambda}, \mathbf{u}) - A_0 & 0 \\ D_u \mu(\boldsymbol{\lambda}, \mathbf{u}) - A_2 & 0 \end{pmatrix}\right\| \leq \frac{1}{2\alpha}.$$

    IF the check is successful
    - Set $b_1 \leftarrow \mathrm{true}$.

Remarks. *(i) We always have to take care that we do not loose rigour in our computations.*

(ii) To remind the value for the bound $\alpha > 0$ we get by the condition (4).

(iii) The value for $\delta \geq 0$ shall be small. We see that a good initial approximative solution $(u_0, \xi_0) \in \mathbb{R}^{n+l}$ and a small absolute value for $\alpha > 0$ provide us this small value for $\delta \geq 0$. The value of $\alpha > 0$ shall be appropriately small if the extended problem is well-conditioned, i.e. if the original problem is not too bad-conditioned near $(\lambda_0, u_0) \in \mathbb{R}^{p+n}$ and if the choice of the extension of the problem is appropriate. Additionally an appropriate choice of the used norms can improve the value for $\alpha > 0$.

(iv) Due to our aim to find a solution path for the original problem we are only interested in solutions of the extended problem where $\xi = 0$. However if the condition $\|\xi_0\| > \delta$ is true then we try to verify a box which restricts the solution path. Obviously the correctness of this condition would yield to a box $\boldsymbol{\xi} \in \mathbb{IR}^l$ such that $0 \notin \boldsymbol{\xi}$. In particular if such a box has been verified b_2 is true *and such a box restricts the area where the solution path possibly crosses.*

If this condition does not hold then we set $\xi_0 = 0$ which yields clearly to $A_1 \cdot \xi_0 = 0$ and so this term does not occur anymore in the determination of $\delta \geq 0$. So we only verify boxes where solutions of the original problem can exist, i.e. if the algorithm is successful $0 \in \boldsymbol{\xi}$ holds.

(v) In this algorithm and the following algorithms we assume that the common maximum norm and its induced matrix norm is used. However if one uses weighted maximum norms to improve the results then one only has to adapt the setting of the box $\mathbf{u} \in \mathbb{IR}^n$ by the midrad *operator. The changes one has to make are described in section 2.2.*

(vi) In the same way like we set $\mathbf{u} \in \mathbb{IR}^n$ we could set $\boldsymbol{\xi} \leftarrow \text{midrad}\,(0, \delta)$. However in general we do not need this enclosure. We only need to know if $0 \in \boldsymbol{\xi}$ or $0 \notin \boldsymbol{\xi}$.

The next question which arises is how to proceed the continuation process.

4.2 Continuation

For this subsection we want to reference to Chow and Hale [3], Golubitsky et al. [10] and Rheinboldt [22].

We have assumed that the real function $F : \mathbb{R}^p \times \mathbb{R}^n \to \mathbb{R}^n$ is $k_o \geq 1$ times continuously differentiable. So the implicit function theorem (Theorem 2.16) says that for a non-singular root $(\lambda_0, u_0) \in \mathbb{R}^{p+n}$ of the function F, neighbourhoods U of λ_0, V of u_0 and a unique continuously differentiable function $u : U \to V$ such that $F(\lambda, u(\lambda)) = 0$. So the function u maps the unique solution branch in the open set U.
For the extended problem we can similarly derive such a unique continuously differentiable function $\begin{pmatrix} \tilde{u} \\ \xi \end{pmatrix} : \Sigma \to U$. The open sets Σ, U must be neighbourhoods of $(\lambda_0, \sigma_0) \in \mathbb{R}^{p+l}$ respectively $(u_0, \xi_0) \in \mathbb{R}^{n+l}$ where $\tilde{F}_{\lambda_0,\sigma_0}(u_0, \xi_0) = 0$. Clearly the original function problem must have been extended appropriately, i.e. there are no singularities in $\Sigma \times U$.
If we have successfully applied Algorithm 2 then we know boxes $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$ and $(\mathbf{u}, \boldsymbol{\xi}) \in \mathbb{IR}^{n+l}$ for which the statement $\forall (\lambda, \sigma) \in (\boldsymbol{\lambda}, \boldsymbol{\sigma}) \, \exists! (u, \xi) \in (\mathbf{u}, \boldsymbol{\xi})$ such that $\tilde{F}_{\lambda,\sigma}(u, \xi) = 0$ holds. In particular then the invertibility of $D_{(u,\xi)}\tilde{F}_{\lambda,\sigma}(u, \xi)$ for all $(\lambda, \sigma, u, \xi) \in (\boldsymbol{\lambda}, \boldsymbol{\sigma}, \mathbf{u}, \boldsymbol{\xi})$ can be implied because Theorem 3.1 holds just there. So there are open sets $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \subset \Sigma, (\mathbf{u}, \boldsymbol{\xi}) \subset U$ and a function $\begin{pmatrix} \tilde{u} \\ \xi \end{pmatrix} : \Sigma \to U$ such that $\tilde{F}_{\lambda,\sigma}(\tilde{u}(\lambda, \sigma), \xi(\lambda, \sigma)) = 0$ holds for all $(\lambda, \sigma) \in \Sigma$.
Actually we are interested in the solution branches of the original problem. In (38) and (39) we see that we have to solve $\xi(\lambda, \sigma) = 0$ where $\xi : \Sigma \to U_\xi$ where $U_\xi \supset \boldsymbol{\xi}$ is an open set. The notation U_ξ means the restriction of U to the coefficients corresponding to ξ. If we try to apply the implicit function theorem to this reduced problem $\xi(\lambda, \sigma) = 0$ some cases can occur. We are assuming that $\xi(\lambda_0, \sigma_0) = 0$ for some $(\lambda_0, \sigma_0) \in (\boldsymbol{\lambda}, \boldsymbol{\sigma})$. This assumption is reasonable because if the function ξ would not have any roots in the considered box then we would not be interested in it anyway. There are three different cases to consider. In figure 4 you can see illustrations how the bifurcation diagram defined by the function $\hat{\mu}$ defined in (40) can be for the corresponding case. The illustrations in figure 4 for simplicity cover the case when $p = l = 1$.

(A) The simplest case is if for all $(\lambda_0, \sigma_0) \in (\boldsymbol{\lambda}, \boldsymbol{\sigma})$ with $\xi(\lambda_0, \sigma_0) = 0$ exist neighbourhoods $\tilde{\Sigma}_\lambda \subseteq \Sigma_\lambda$ of λ_0, $\tilde{\Sigma}_\sigma \subseteq \Sigma_\sigma$ of σ_0 and a unique continuously differentiable function $\eta : \tilde{\Sigma}_\lambda \to \tilde{\Sigma}_\sigma$ exist such that $\xi(\lambda, \eta(\lambda)) = 0$ for all $\lambda \in \tilde{\Sigma}_\lambda$, i.e. if the matrix $D_\sigma \xi(\lambda_0, \sigma_0) \in \mathbb{R}^{l \times l}$ is invertible for all such (λ_0, σ_0). Then obviously $F(\lambda, \tilde{u}(\lambda, \eta(\lambda))) = 0$ holds for all $\lambda \in \tilde{\Sigma}_\lambda$ and the function $\tilde{u} \circ \eta : \tilde{\Sigma}_\lambda \to \tilde{V}_u$ is continuously differentiable for $\tilde{V}_u \subseteq V_u$ suitable.

In the figure 4 the graphs (a) - (d) cover this case.

(B) In this case there is a $(\lambda_0, \sigma_0) \in (\boldsymbol{\lambda}, \boldsymbol{\sigma})$ where the matrix $D_\sigma \xi(\lambda_0, \sigma_0) \in \mathbb{R}^{l \times l}$ has not full rank, i.e. is not invertible. Then due to the implicit function theorem no such neighbourhoods and function like in the first case can be found for (λ_0, σ_0). On the other hand if the matrix $D_{(\lambda,\sigma)} \xi(\lambda_0, \sigma_0)$ has full rank then we can re-sort the variables and find such a continuously differentiable function which parametrizes the solution branch near (λ_0, σ_0) with respect to some $\tilde{\lambda} \in \mathbb{R}^p$. Similarly as above then $F(\tilde{\lambda}, \tilde{u}(\lambda, \eta(\tilde{\lambda}))) = 0$ holds. The variables of the function F must have been re-sorted in the same way as for the reduced problem. For example in the simplest case $p = l = 1$ then a function $\eta : \tilde{\Sigma}_\sigma \to \tilde{\Sigma}_\lambda$ such that $\xi(\eta(\sigma), \sigma)) = 0$ in the suitable chosen sets $\tilde{\Sigma}_\lambda$ and $\tilde{\Sigma}_\sigma$. For $p = l = 1$ this phenomena is called turning point. Again we can follow that the corresponding solution branch of the original problem is continuously differentiable in the corresponding neighbourhood when the branch is appropriately parametrized.

Graphs with common turning points you can see in figure 4, graphs (e), (f) and (g).

(C) This case differs from the previous case only because here the matrix $D_{(\lambda,\sigma)} \xi(\lambda_0, \sigma_0) \in \mathbb{R}^{l \times l}$ has not full rank too. Here no such function exists to parametrize the solution branch, even if the variables are re-sorted, with respect to some $\tilde{\lambda} \in \mathbb{R}^p$. Then we have a singularity at $(\lambda_0, \sigma_0) \in \mathbb{R}^{p+l}$, in the reduced problem, probably a bifurcation point. Due to the construction of the extended problem $\tilde{F}_{\lambda,\sigma}(u, \xi) = 0$ this implies that there is a point $u_0 := u(\lambda_0, \sigma_0) \in \mathbf{u}$ such that the original problem has a singularity at $(\lambda_0, u_0) \in \Lambda \times D$. See for this implication the discussion in section 3.1, especially Theorem 3.2.

In the graphs (h),(i) in figure 4 you can see bifurcation points. In (h) the bifurcation point is additionally a turning point too. This situation is called often pitchfork bifurcation.

For the continuation we are using the continuity of the solution branches. We see in the illustrations that if a solution branch of the reduced problem $\xi(\lambda, \sigma) = 0$ crosses the considered box $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$ then the considered branch must cross the boundary of $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$ at least twice. The continuity of the solution path implies this. If there are more than two areas in the considered box $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$ then this implies that the considered box contains either a bifurcation point or that at least two distinct solution paths which cross $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$ without bifurcating. The last case is if we have a solution branch which is completely contained in the considered box

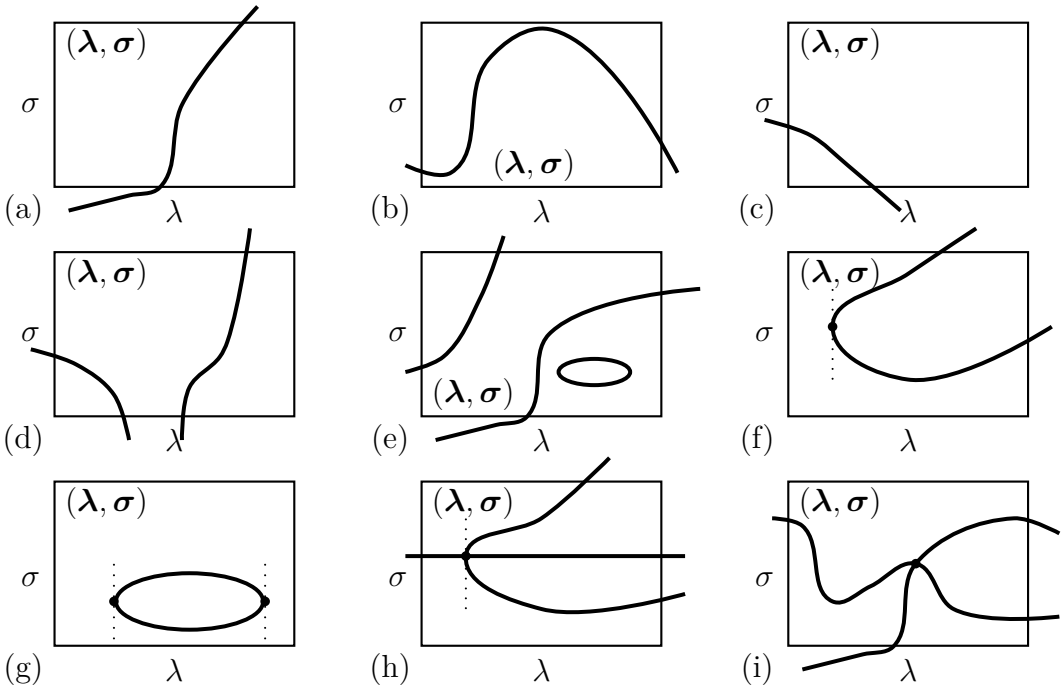

Figure 4: Some possible kinds of bifurcation diagrams when $p = l = 1$.

$(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$. Then the solution branch does not cross the boundary of $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$.

Remarks. *(i) A solution branch of the reduced problem can be pieced together continuously through the cases* (A), (B) *and* (C)*. In particular if we have the simplest case $p = 1$ then each branch is a curve, for $p > 1$ each branch is a p-dimensional manifold.*

(ii) It is reasonable to assume that the considered box $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$ is narrow. So we call the case when more than one solution branch crosses the box without bifurcating an imperfect bifurcation. This situation we have in the graphs (d) and (e) in figure 4.

(iii) Until now we have not defined a bifurcation point. Here we give an informal definition for the problem $F(\lambda, u) = 0$ where the function $F : \mathbb{R}^p \supseteq \Lambda \times \mathbb{R}^n \to \mathbb{R}^n$ is continuously differentiable. In a bifurcation point (λ_0, u_0) locally the number of solutions of the considered problem changes with $\lambda \in \mathbb{R}^p$. Further this definition shall not be dependent on re-sorting the variables.

Necessarily the matrix $D_{(\lambda,u)}F(\lambda_0, u_0) \in \mathbb{R}^{n\times(p+n)}$ must not have full

rank at a bifurcation point $(\lambda_0, u_0) \in \Lambda \times \mathbb{R}^n$. *However this condition is not sufficient.*

(iv) When we speak of re-sorting the variables $(x_1, \ldots, x_n) \in \mathbb{R}^n$ *then we mean to get variables* $(\tilde{x}_1, \ldots, \tilde{x}_n) \in \mathbb{R}^n$ *where* $(\tilde{x}_1, \ldots \tilde{x}_n) = (x_{\pi(1)}, \ldots, x_{\pi(n)})$ *and where* π *is a permutation of the set* $\{1, \ldots, n\}$.

(v) Again assuming that the box we are considering is narrow there are maximally $l+1$ *solution branches in this box because otherwise the extended problem still is ill-conditioned and as a consequence the verification of Theorem 3.1 does not succeed. The graph (e) in figure 4 may be an example for this situation. Then the extended problem must be extended again. See for this section 3.6.*

As a conclusion we see that analysing the boundary of the box $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$ gives us useful informations for the continuation. We have to find areas of the boundary of the considered box where a solution branch crosses and the number of such areas give us a good clue about which situation we have. For the simple $l = 1$ case this means that two such areas indicates that we have no singularity in the considered box. By contrast four such areas are indicating a bifurcation or at least an imperfect bifurcation. If there is no such area in the boundary then this indicates that the whole branch is contained in this box. This is case is unlikely if the box is narrow. For $l > 1$ the thoughts are very similar.
Further we use the knowledge of the area where the solution branch crosses to choose the next box we want to verify (see figure 5). So the next topic is how to analyse the boundary of a verified box $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$.

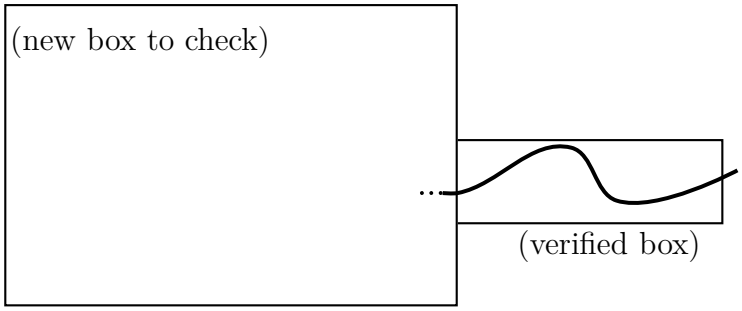

Figure 5: Choice of another box to check. Case $p = l = 1$.

4.3 Analysing the boundary of a verified box

Here we are considering interval vectors $\boldsymbol{\lambda} \in \mathbb{IR}^p, \boldsymbol{\sigma} \in \mathbb{IR}^l, \mathbf{u} \in \mathbb{IR}^n$ which has been successfully verified by the previously presented algorithm. Now we want to analyse the solution of the extended problem for all $(\lambda, \sigma) \in \partial(\boldsymbol{\lambda}, \boldsymbol{\sigma})$, the boundary of $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$.
For simplicity now we say $p = l = 1$. First we are considering the left boundary edge (see figure above). This edge can be written as an interval $(\underline{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^2$ where to recall the notation $\boldsymbol{\lambda} = [\underline{\lambda}, \overline{\lambda}]$. Here we consider $\underline{\lambda}$ as an interval. Obviously the conditions

$$\|F(\underline{\lambda}, u_0) + A_1\xi_0\| \leq \sup \|F(\boldsymbol{\lambda}, u_0) + A_1\xi_0\|$$

and

$$\sup \|\mu(\underline{\lambda}, u_0) - \boldsymbol{\sigma}\| \leq \sup \|\mu(\boldsymbol{\lambda}, u_0) - \boldsymbol{\sigma}\|,$$

where $u_0 = \operatorname{mid} \mathbf{u}$ are true because $\underline{\lambda} \subseteq \boldsymbol{\lambda}$. So we can set $\delta \geq 0$ absolute smaller than in the previous verifying algorithm for our considered interval vectors. When we set the corresponding interval vectors, now named by $\tilde{\mathbf{u}} \in \mathbb{IR}^p$, in the same way like in the previous verifying algorithm we have $\tilde{\mathbf{u}} \subseteq \mathbf{u}$. So the condition

$$\sup \left\| \begin{pmatrix} D_u F(\underline{\lambda}, \tilde{\mathbf{u}}) - A_0 & 0 \\ D_u \mu(\underline{\lambda}, \tilde{\mathbf{u}}) - A_2 & 0 \end{pmatrix} \right\| \leq \sup \left\| \begin{pmatrix} D_u F(\boldsymbol{\lambda}, \mathbf{u}) - A_0 & 0 \\ D_u \mu(\boldsymbol{\lambda}, \mathbf{u}) - A_2 & 0 \end{pmatrix} \right\| \leq \frac{1}{2\alpha}$$

always holds. Unfortunately, we do not get additionally information her because $\xi_0 = 0$ always in verified boxes. So we have to think of a way to verify that no solution branch of the original problem passes the considered edge.
So we aim to find an interval vector $(\tilde{\mathbf{u}}, \tilde{\boldsymbol{\xi}}) \in \mathbb{IR}^{n+l}$ corresponding to the boundary edge $(\underline{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$ where $0 \notin \tilde{\boldsymbol{\xi}}$. So we start with determining an approximation for the solution of the extended problem $(\tilde{u}_0, \tilde{\xi}_0) \in \mathbb{R}^{n+l}$ at $(\underline{\lambda}, \operatorname{mid} \boldsymbol{\sigma}) \in \mathbb{R}^{p+l}$. In general here $\tilde{\xi_0} \neq 0$, what is important for us. Assuming we have found such a good approximation $(\tilde{u}_0, \tilde{\xi}_0) \in \mathbb{R}^{n+l}$ we first determine

$$\delta_1 := \left\| \tilde{\xi}_0 \right\|. \tag{62}$$

For the verification that no solution path passes the boundary edge we need $\delta < \delta_1$ because then we have certainly $0 \notin \tilde{\boldsymbol{\xi}}$. Next we determine $\delta \geq 0$ in almost the same way like in the previous algorithm,

$$\delta = 2\alpha \max \left\{ \sup \left(\left\| F(\underline{\lambda}, \tilde{u}_0) + A_1 \tilde{\xi}_0 \right\|, \|\mu(\underline{\lambda}, \tilde{u}_0) - \boldsymbol{\sigma}\| \right) \right\}.$$

If the condition $\delta < \delta_1$ is true then we reset $\delta \geq 0$ such that $\delta < \delta_1$ and maximum. Then we can set $(\tilde{\mathbf{u}}, \tilde{\boldsymbol{\xi}}) \in \mathbb{IR}^{n+l}$ in the same way like in the

previous algorithm. By this construction the condition $(\tilde{\mathbf{u}}, \tilde{\boldsymbol{\xi}}) \subseteq (\mathbf{u}, \boldsymbol{\xi})$ is not necessarily holding. However, clearly we are only interested in the area $(\tilde{\mathbf{u}}, \tilde{\boldsymbol{\xi}}) \cap (\mathbf{u}, \boldsymbol{\xi})$. Then we consider

$$\sup \left\| \begin{pmatrix} D_u F(\underline{\lambda}, \tilde{\mathbf{u}} \cap \mathbf{u}) - A_0 & 0 \\ D_u \mu(\underline{\lambda}, \tilde{\mathbf{u}} \cap \mathbf{u}) - A_2 & 0 \end{pmatrix} \right\| \leq \sup \left\| \begin{pmatrix} D_u F(\boldsymbol{\lambda}, \mathbf{u}) - A_0 & 0 \\ D_u \mu(\boldsymbol{\lambda}, \mathbf{u}) - A_2 & 0 \end{pmatrix} \right\| \leq \frac{1}{2\alpha}.$$

Obviously this condition automatically is true. So we do not have to check this condition explicitly. So we see we only have to check the condition $\delta < \delta_1$. The set $(\tilde{\mathbf{u}}, \tilde{\boldsymbol{\xi}}) \cap (\mathbf{u}, \boldsymbol{\xi})$ must not be empty if the approximate root $(\tilde{u}_0, \tilde{\xi}_0) \in \mathbb{R}^{n+l}$ is appropriate. If this set is empty then the approximate root $(\tilde{u}_0, \tilde{\xi}_0) \in \mathbb{R}^{n+l}$ is contained in another wrong area and our arguments are not applicable because they are locally.
For the boundary edge where the σ value is held constant the procedure works in almost the same way.

Algorithm 3 (Boundary analysing)
Try to verify that no solution branch of the reduced problem passes the considered part of the boundary $(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\sigma}}) \subseteq \partial(\boldsymbol{\lambda}, \boldsymbol{\sigma})$.

INPUT

- A part of the boundary of the considered interval vector, here called $(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\sigma}}) \in \mathbb{IR}^{p+l}$.
- An approximate solution $(\tilde{u}_0, \tilde{\xi}_0) \in \mathbb{R}^{n+l}$ of the extended problem at $\mathrm{mid}(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\sigma}}) \in \mathbb{R}^{p+l}$.

OUTPUT

- A boolean variable b initialised with false which indicates if the verification has been successful or not.

THE ALGORITHM

- Set δ.

$$\delta \leftarrow 2\alpha \max \left\{ \sup \left(\left\| F(\tilde{\boldsymbol{\lambda}}, u_0) + A_1 \xi_0 \right\|, \left\| \mu(\tilde{\boldsymbol{\lambda}}, u_0) - \tilde{\boldsymbol{\sigma}} \right\| \right) \right\}$$

- IF $\delta < \left\| \tilde{\xi}_0 \right\|$.
  - Set $b \leftarrow \mathrm{true}$.

Remarks. *(i) We consider the part of the boundary* $(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\sigma}}) \in \mathbb{IR}^{p+l}$ *of the considered interval box* $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$ *as an interval vector. Obviously in the simplest case when* λ *and* σ *are one-dimensional variables then either* $\tilde{\boldsymbol{\lambda}} \in \mathbb{IR}^p$ *or* $\tilde{\boldsymbol{\sigma}} \in \mathbb{IR}^l$ *is a thin interval, i.e. a real constant represented by its interval representation.*

(ii) In general the boundary of a box $\mathbf{x} \in \mathbb{IR}^n$ *can be seen as the union of all interval vectors in the set*

$$\left\{ (\mathbf{x}_1, \ldots, \underline{x}_i, \ldots, \mathbf{x}_n), (\mathbf{x}_1, \ldots, \overline{x}_i, \ldots, \mathbf{x}_n) \;\middle|\; i = 1, \ldots, n \right\}.$$

(iii) When λ *and* σ *are higher-dimensional variables the algorithm remains valid.*

Until now we did not think about what is happening when the verifying algorithm or the algorithm to analyse a part of a boundary is not successful. We start considering this next.
Obviously we have to consider a smaller interval vector when Algorithm 3 fails. So we introduce the concept of bisection.

Algorithm 4 (Bisection)
Split a interval vector $\mathbf{x} \in \mathbb{IR}^n$ into a list of intervals $\mathbf{x}^i$ for which $\mathbf{x} = \bigcup_i \mathbf{x}^i$ holds by bisecting in every coordinate.

INPUT

- An interval box $\mathbf{x} \in \mathbb{IR}^n$.

OUTPUT

- A list of interval vectors $\mathcal{L}$ for which $\mathbf{x} = \bigcup_{\mathbf{w} \in \mathcal{L}} \mathbf{w}$ holds.

THE ALGORITHM

- Add $\mathbf{x}$ to the list $\mathcal{L}$.
- FOR $k = 1, \ldots, n$.
  - Initialise a list $\mathcal{K}$ with the same intervals as in $\mathcal{L}$.
  - Delete all elements in $\mathcal{L}$.
  - WHILE $\mathcal{K}$ is not empty.
    - Take the first interval box $\mathbf{y}$ from $\mathcal{K}$ and delete it in the list.

* IF $\operatorname{rad} \mathbf{y}_k \neq 0$
  - Set two new interval vectors $\mathbf{y}^1, \mathbf{y}^2 \in \mathbb{IR}^n$. If $j \neq k$ holds then $\mathbf{y}_j^p \leftarrow \mathbf{y}_j$ $(p = 1, 2)$ and else

$$\mathbf{y}_k^1 \leftarrow [\underline{y}_k, \frac{\underline{y}_k + \overline{y}_k}{2}], \qquad \mathbf{y}_k^2 \leftarrow [\frac{\underline{y}_k + \overline{y}_k}{2}, \overline{y}_k].$$

  - Add these two interval boxes $\mathbf{y}^1, \mathbf{y}^2$ to the list $\mathcal{L}$.
* ELSE
  - Add $\mathbf{y}$ to the list $\mathcal{L}$.

Remarks. *(i) By applying this algorithm for $\mathbf{x} \in \mathbb{IR}^n$ we see the maximal number of intervals which contain $\mathcal{L}$ is increasing exponentially with n. So this algorithm is not suitable for a large n. However, we apply this algorithm either for the interval vectors $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$ or $\boldsymbol{\lambda} \in \mathbb{IR}^p$ and due to our construction then p and l is fairly small.*

(ii) In application the condition $\operatorname{rad} \mathbf{y}_k \neq 0$ one should replace by $\operatorname{rad} \mathbf{y}_k \geq \epsilon$ where $\epsilon > 0$ bounds the minimal radius of the interval vectors one wants to consider. Additionally one may avoid infinite loops.

(iii) See the figure 6 for an illustration when $n = 2$.

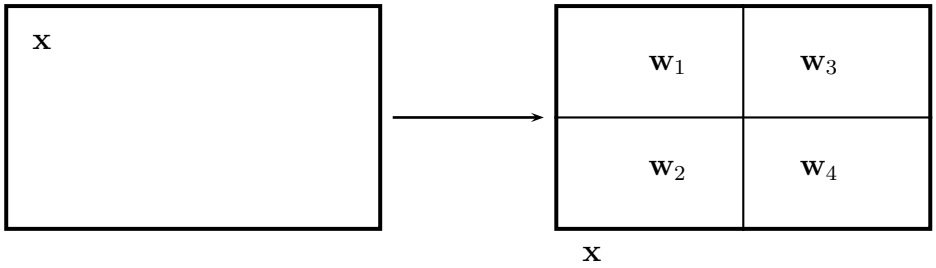

Figure 6: Bisection for $n = 2$.

Further if we have the case that a solution branch passes the considered party of boundary $(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\sigma}}) \in \mathbb{IR}^{p+l}$ then Algorithm 3 must not be successful. So we introduce a bound for the maximal radius of a part of the boundary $(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\sigma}}) \in \mathbb{IR}^{p+l}$ on which we apply Algorithm 3. If we have such a part of the boundary where the radius is lower as the introduced bound we assume that the solution branch crosses the boundary of the box there. So we can write down the algorithm which analyses the complete boundary of $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$.

Algorithm 5 (Complete boundary analysing)
Find parts of the boundary of $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$ where a solution branch traverses probably.

INPUT

- A list of interval vectors $\mathcal{L}$ where each interval vector represents a part of the boundary of the interval vector $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$.
- The corresponding enclosure $\mathbf{u} \in \mathbb{R}^n$ to the box $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$.
- A function $g(\lambda, \sigma)$ which allows to calculate an approximate solution of the extended problem $\tilde{F}_{\lambda,\sigma}(u, \xi) = 0$.

OUTPUT

- A list $\mathcal{V}$ which contains interval vectors $(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\sigma}}) \subseteq \partial(\boldsymbol{\lambda}, \boldsymbol{\sigma})$ where a solution branch may traverse and a corresponding enclosure $\tilde{\mathbf{u}} \in \mathbb{IR}^n$.

THE ALGORITHM

- WHILE $\mathcal{L}$ is not empty.
  - Take an interval vector $(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\sigma}})$ from the list $\mathcal{L}$ and delete it in $\mathcal{L}$.
  - Set $(\tilde{u}_0, \tilde{\xi}_0) \leftarrow g(\mathrm{mid}(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\sigma}}))$.
  - IF $\max \mathrm{rad}(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\sigma}}) > \epsilon$ where $\epsilon > 0$ and suitable.
    - Apply Algorithm 3 with input $(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\sigma}})$ and $(\tilde{u}_0, \tilde{\xi}_0)$.
    - IF Algorithm 3 was not successful, i.e. b is false.
      - Apply Algorithm 4 on $(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\sigma}})$ Then we get a list $\mathcal{K}$. Now add all the interval vectors contained in the list $\mathcal{K}$ to the list $\mathcal{L}$.
  - ELSE (Here is assumed that a part of the boundary is found where a solution branch crosses).
    - Set δ.

$$\delta \leftarrow 2\alpha \max \left\{ \sup(\left\| F(\boldsymbol{\lambda}, u_0) \right\|, \left\| \mu(\boldsymbol{\lambda}, u_0) - \boldsymbol{\sigma} \right\|) \right\}$$

    - Set $\tilde{\mathbf{u}}$.

$$\tilde{\mathbf{u}} \leftarrow \mathrm{midrad}\,(u_0, \delta)$$
$$\tilde{\mathbf{u}} \leftarrow \tilde{\mathbf{u}} \cap \mathbf{u}.$$

    - Add $(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\sigma}})$ and $\tilde{\mathbf{u}}$ to the list $\mathcal{V}$.

Remarks. *(i) We have to take care of the fact that this algorithm does not provide boxes* $(\tilde{\boldsymbol{\lambda}}, \tilde{\mathbf{u}}) \in \mathbb{IR}^{p+l}$ *where a corresponding unique solution branch of the original problem is verified. However, if* $\epsilon > 0$ *is chosen appropriately small then it is very likely that such a solution branch exists uniquely in these box and is.*

(ii) The restricting setting $\tilde{\mathbf{u}} \leftarrow \tilde{\mathbf{u}} \cap \mathbf{u}$ *is necessary to ensure rigour but in general the considered boxes are very narrow so it is very likely that* $\mathbf{u} \in \mathbb{IR}^n$ *without this restriction is an appropriate corresponding enclosure too. Without this restriction we would have to check the condition* (35) *instead for* $\tilde{\mathbf{u}} \in \mathbb{IR}^n$.

4.4 Complementation

Another procedure we need is taking the complement of an interval box $\mathbf{x} \in \mathbb{IR}^n$ in an interval box $\mathbf{y} \in \mathbb{IR}^n$.

Algorithm 6 (Complementation)
Take the complement of an interval vector $\mathbf{x} \in \mathbb{IR}^n$ in an interval vector $\mathbf{y} \in \mathbb{IR}^n$. See figure 7 for an illustration when $n = 2$.

INPUT

- The interval vectors $\mathbf{x}, \mathbf{y} \in \mathbb{IR}^n$.

OUTPUT

- A list of interval vectors $\mathcal{L}$ such that $\bigcup_{\mathbf{w} \in \mathcal{L}} \mathbf{w} = \mathbf{y} \backslash \mathbf{x}$.

THE ALGORITHM

- Initialise an empty list $\mathcal{L}$.
- IF $\mathbf{x} \cap \mathbf{y}$ is empty.
  - Insert $\mathbf{y}$ into the list $\mathcal{L}$.
- ELSE
  - FOR $i = 1, \ldots, n$.
    - Set the interval $\mathbf{z} \leftarrow \mathbf{x}_i \cap \mathbf{y}_i$.
    - IF $\underline{z} > \underline{y}_i$.

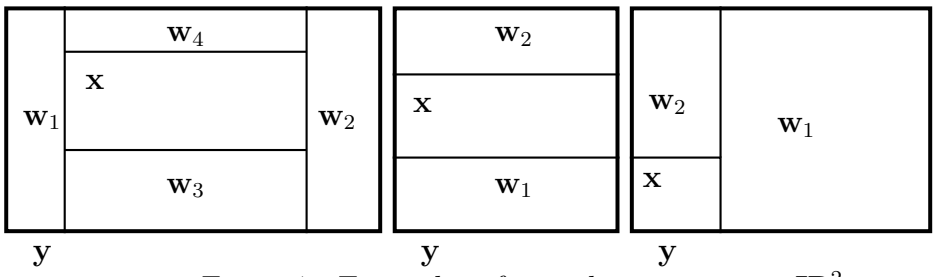

Figure 7: Examples of complementation in $\mathbb{IR}^2$

- Set an interval vector $\mathbf{w} \in \mathbb{IR}^n$ such that its ith coordinate is $[\underline{y}, \underline{z}]$ and its other coordinates are the same as those of $\mathbf{y}$.
- Insert $\mathbf{w}$ into $\mathcal{L}$.

* If $\overline{z} < \overline{y}_i$.
  - Set an interval vector $\mathbf{w} \in \mathbb{IR}^n$ such that its ith coordinate is $[\overline{z}, \overline{y}]$ and its other coordinates are the same as those of $\mathbf{y}$.
  - Insert $\mathbf{w}$ into $\mathcal{L}$.
* Replace the ith coordinate of $\mathbf{y}$ by $\mathbf{z}$.

Remark.
The complementation algorithm does not need necessarily $\mathbf{x} \subseteq \mathbf{y}$, *but we need only this case so the illustration in figure 7 covers only this case. The illustration does not cover all possible cases of complementation in* $\mathbb{R}^2$.

Now we can write down the procedure how to completely analyse a box $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+n}$ concerning a crossing solution branch of the reduced problem $\xi(\lambda, \sigma) = 0$.

4.5 Completely analysing a box

Algorithm 7 (Complete box analysing)
Here either a smaller box to verify or a verified box is found. Additionally boxes to continue the algorithm are provided and the boundary of the possibly verified box is analysed.

Input

- The interval boxes $\boldsymbol{\lambda} \in \mathbb{IR}^p$ and $\boldsymbol{\sigma} \in \mathbb{IR}^l$.

- The lists of interval vectors $\mathcal{K}$ and $\mathcal{L}$.
- (optional) An approximate root $(\lambda_0, u_0) \in \Lambda \times \mathbb{R}^n$ of the function $F : \Lambda \times \mathbb{R}^n \to \mathbb{R}^n$.

OUTPUT

- Modified lists $\mathcal{K}, \mathcal{L}$.
- If the algorithm is successful the verified box $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$ with a corresponding box $\mathbf{u} \in \mathbb{IR}^n$ and its corresponding boundary information in the list $\mathcal{V}$.

THE ALGORITHM

- Set $(u_0, \xi_0) \leftarrow g(\mathrm{mid}(\boldsymbol{\lambda}, \boldsymbol{\sigma}))$ unless the approximate root is given.
- Apply Algorithm 2. The input is $(\boldsymbol{\lambda}, \boldsymbol{\sigma})$ and u_0.
- IF the verification in Algorithm 2 was successful.
  (a) Now the interval vectors $\boldsymbol{\lambda} \in \mathbb{IR}^p, \boldsymbol{\sigma} \in \mathbb{IR}^l, \mathbf{u} \in \mathbb{IR}^n$ are known.
  (b) Form interval vectors representing the boundary of $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$ and initialise a list $\mathcal{B}$ containing all these interval vectors.
  (c) Apply Algorithm 5. The input is the list $\mathcal{B}$ and the output is a list $\mathcal{V}$ which contains the information where the solution branches cross probably the boundary of the considered box $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$.
  (d) Find all interval vectors $(\hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\sigma}}) \in \mathbb{IR}^{p+l}$ from the list $\mathcal{K}$ which have a non-empty intersection (i.e. is 'neighboured') with one of the interval vectors $(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\sigma}}) \in \mathbb{IR}^{p+l}$ from the list $\mathcal{V}$. If such a box is found, delete this box in $\mathcal{K}$ and add it to the list $\mathcal{L}$.
- ELSE
  (A) Set an interval vector $(\hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\sigma}}) \in \mathbb{IR}^{p+l}$ such that $(\hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\sigma}}) \subseteq (\boldsymbol{\lambda}, \boldsymbol{\sigma})$ and narrower. Further the solution branch shall cross $(\hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\sigma}}) \in \mathbb{IR}^{p+l}$ certainly. See figure 8.
  If an approximative root is given then set
  $(\hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\sigma}}) \leftarrow \mathrm{midrad}\left((\lambda_0, \sigma_0), \frac{1}{2}\mathrm{rad}(\boldsymbol{\lambda}, \boldsymbol{\sigma})\right) \cap (\boldsymbol{\lambda}, \boldsymbol{\sigma})$.
  (B) Take the complement of $(\hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\sigma}}) \in \mathbb{IR}^{p+l}$ in $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$ Use the complementation, Algorithm 6. Then add the interval vectors of the list which is set through the complementation algorithm to the

list $\mathcal{K}$ or $\mathcal{L}$ depending on if a box has already been verified from which a solution path traverses into the box. Further add the box $(\hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\sigma}}) \in \mathbb{IR}^{p+l}$ to the list $\mathcal{L}$.

Remarks. *(i) The list $\mathcal{L}$ shall contain interval vectors $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$ which shall be verified with Algorithm 2. Unless at the beginning of the overall algorithm a box is added to $\mathcal{L}$ when a different box has been verified with Algorithm 2 and these two boxes have a common boundary 'edge' and additionally Algorithm 3 has shown that a solution branch crosses this common boundary. So the list $\mathcal{L}$ shall store additionally this corresponding boundary information for each box in it.*

Then there is a list $\mathcal{S}$ where verified boxes $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$ and the corresponding box $\mathbf{u} \in \mathbb{IR}^n$ is stored. Further information on the boundary of each element in $\mathcal{S}$ may be stored in the list.

In the overall algorithm we are considering the solution branch of the reduced problem in a box $(\boldsymbol{\lambda}^0, \boldsymbol{\sigma}^0) \in \mathbb{IR}^{p+l}$. Then the list $\mathcal{K}$ contains interval vectors such that the union of this interval vectors is the complement of the union of all interval vectors $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$ of the lists $\mathcal{S}, \mathcal{L}$ in the box $(\boldsymbol{\lambda}^0, \boldsymbol{\sigma}^0)$.

(ii) For the setting of $(\hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\sigma}}) \in \mathbb{IR}^{p+l}$ in (A) *one must ensure that the solution path crosses this box (see figure 8). By combining bisection and the* midrad *operator such a box can be found easily.*

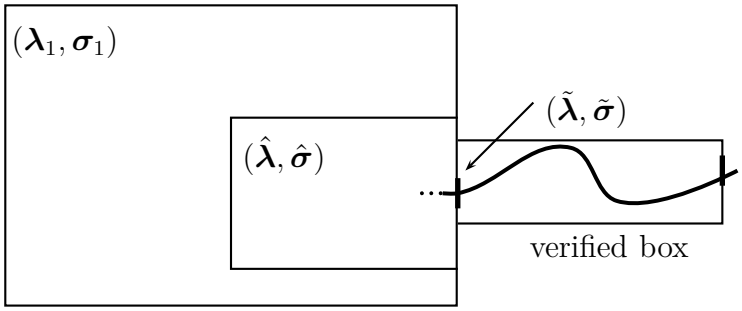

Figure 8: How to choose an appropriately narrower box.

4.6 Overall algorithm

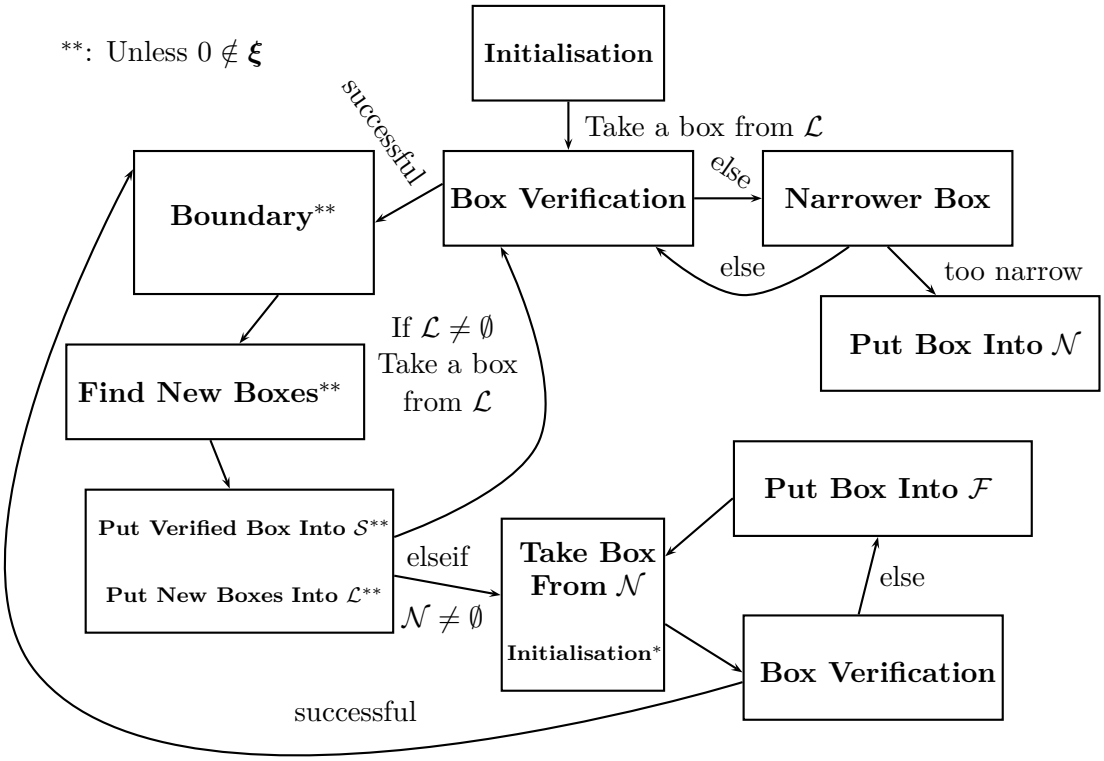

Figure 9: Sketch of overall algorithm

Algorithm 8 (Overall algorithm)
Find a verified enclosure for the solution paths of the original problem in a box $(\boldsymbol{\lambda}^0, \mathbf{u}^0) \subseteq \Lambda \times \mathbb{R}^n$ starting at an approximate root $(\lambda_0, u_0) \in (\boldsymbol{\lambda}^0, \mathbf{u}^0)$.

INPUT

- The interval boxes $\boldsymbol{\lambda}^0 \in \mathbb{IR}^p$, $\mathbf{u}^0 \in \mathbb{IR}^n$ in which the verified solution paths of the problem must be found.
- An initial approximate solution $(\lambda_0, u_0) \in (\boldsymbol{\lambda}^0, \mathbf{u}^0)$ of the original problem.
- (optional) A given unfolding functional $\mu : \mathbb{R}^p \times \mathbb{R}^n \to \mathbb{R}^l$.
- (optional) Some lists $\mathcal{K}, \mathcal{L}$.

OUTPUT

- Some lists $\mathcal{S}, \mathcal{F}$.

THE ALGORITHM

(a) Use the discussion from sections 3.2 to 3.7 to determine the extension of the problem. Now $A_1 \in \mathbb{R}^{n\times l}, A_2 \in \mathbb{R}^{n\times l}$ are known and if necessary the unfolding functional μ is also determined.

(b) Compute the bound $\alpha > 0$ for the norm of the inverse of $A = \begin{pmatrix} A_0 & A_1 \\ A_2 & 0 \end{pmatrix}$ where $A_0 = D_u F(\lambda_0, u_0)$.

(c) Set an initial box $\mathbb{IR}^l \ni \boldsymbol{\sigma}^0 \leftarrow \mu(\boldsymbol{\lambda}^0, \mathbf{u}^0)$.

(d) Determine an approximate solution function $g(\lambda, \sigma)$ of the extended problem around the known approximate solution. See sections 3.8 and 2.3.

(e) If necessary initialise the lists $\mathcal{L}, \mathcal{K}$ of interval boxes both with $(\boldsymbol{\lambda}^0, \boldsymbol{\sigma}^0)$. Initialise the lists $\mathcal{S}, \mathcal{F}$ as empty lists.

(f) WHILE $\mathcal{L}$ is not empty.

- Take a box $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$ and possibly its corresponding boundary information, another box $(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\sigma}}) \in \mathbb{IR}^{p+l}$, from $\mathcal{L}$ and delete it in $\mathcal{L}$.
- IF $\operatorname{rad}(\boldsymbol{\lambda}, \boldsymbol{\sigma}) > \epsilon$.
  - Apply Algorithm 7 to completely analyse $(\boldsymbol{\lambda}, \boldsymbol{\sigma})$. The input is $(\boldsymbol{\lambda}, \boldsymbol{\sigma})$, the lists $\mathcal{K}, \mathcal{L}$ and if available $(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\sigma}})$ and the approximate root (λ_0, u_0).
  - If Algorithm 7 has been successful then add the resulting verified box $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$, the corresponding box $\mathbf{u} \in \mathbb{IR}^n$ and the corresponding boundary information stored in a list $\mathcal{V}$ to the list $\mathcal{S}$.

  ELSEIF $\mathcal{L}$ is not empty OR $\mathcal{S}$ is not empty.

  - Add the box $(\boldsymbol{\lambda}, \boldsymbol{\sigma})$ and its corresponding boundary information, another box $(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\sigma}})$, to the list $\mathcal{N}$.
  - Add an approximate root $\mathbb{R}^{p+n} \ni (\lambda_1, u_1) \in \mathbb{R}^{p+n}$ corresponding to the box $(\boldsymbol{\lambda}, \boldsymbol{\sigma})$ to the list $\mathcal{N}$. Set $\lambda_1 \leftarrow \operatorname{mid} \boldsymbol{\lambda}_1$ and set $u_1 \in \mathbb{IR}^n$ by determining $g(\operatorname{mid}(\boldsymbol{\lambda}_1, \boldsymbol{\sigma}_1))$.

  ELSE

– Add the box $(\boldsymbol{\lambda}, \boldsymbol{\sigma})$ to the list $\mathcal{F}$.

(g) WHILE $\mathcal{N}$ is not empty.

- Take a box $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$, its corresponding boundary information and the corresponding approximate root $(\lambda_1, u_1) \in \mathbb{R}^{p+l}$ from the list $\mathcal{N}$ and delete this box in the list $\mathcal{N}$. If this box has not been verified already, this means the box is not contained in $\mathcal{S}$ then initialise the list $\mathcal{L}_{\text{new}}$ with this box and the corresponding boundary information. Otherwise take a new box from $\mathcal{N}$ and try again.
- Apply recursively Algorithm 8. The input is $(\boldsymbol{\lambda}^0, \mathbf{u}^0)$, the lists $\mathcal{K}, \mathcal{L}_{\text{new}}$ as well as the corresponding approximate root $(\lambda_1, u_1) \in \mathbb{R}^{p+n}$ of the function F. Furthermore is the possibly determined functional μ input too.
- Concatenate the resulting lists $\tilde{\mathcal{S}}, \tilde{\mathcal{F}}$ with the lists $\mathcal{S}, \mathcal{F}$.

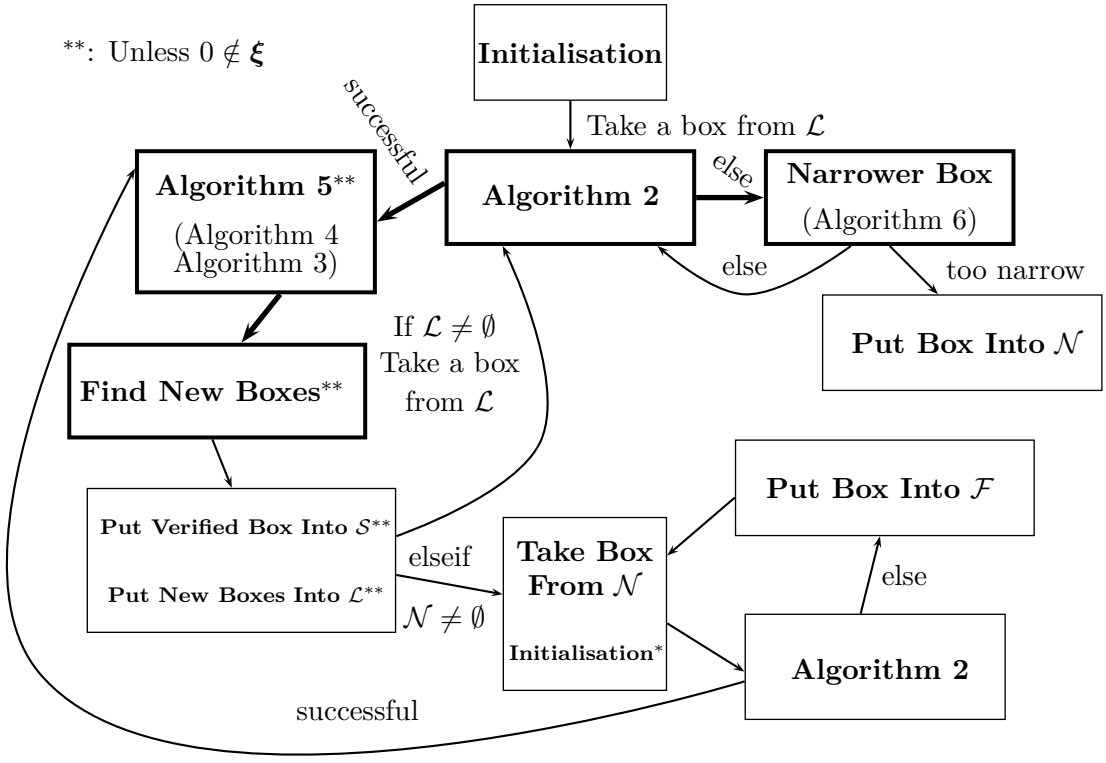

Figure 10: Overall algorithm, Algorithm 8

Remarks. *(i) The bold marked frames and arrows mark the steps of Algorithm 7. The bracketed algorithms are needed amongst others in this step.*

(ii) By initiation we mean the steps (a) *to* (e) *in Algorithm 8. In the initiation* step the steps* (c) *and* (d) *has to be skipped. Further then no new unfolding functional must be determined.*

(iii) The list $\mathcal{S}$ obviously shall contain all verified boxes $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$ with the corresponding box $\mathbf{u} \in \mathbb{IR}^n$ and corresponding boundary information. Then the condition for all $(\lambda, \sigma) \in (\boldsymbol{\lambda}, \boldsymbol{\sigma})$ exists $(u, \xi) \in (\mathbf{u}, \boldsymbol{\xi})$ such that $\tilde{F}_{\lambda,\sigma}(u, \xi) = 0$ holds for an appropriately large box $\boldsymbol{\xi} \in \mathbb{IR}^l$ for which $0 \in \boldsymbol{\xi}$ must hold.

In the list $\mathcal{N}$ boxes are stored for which the verification of Theorem 3.1 fails with the determined matrix A and $\alpha > 0$. For boxes in $\mathcal{N}$ we assume that the reason for the failure is that the interval matrix $A \in \mathbb{R}^{(n+l)\times(n+l)}$ does not approximate the Jacobian matrix in this box well enough anymore (see condition (35)*). So the algorithm aims to verify Theorem 3.1 for boxes in $\mathcal{N}$ with a more suitable matrix A and corresponding bound $\alpha > 0$ (see* (g)*). Additionally corresponding boundary information shall be stored in $\mathcal{N}$. Further a corresponding approximate root is stored in $\mathcal{N}$ to initialise Algorithm 8.*

In the list $\mathcal{F}$ boxes are stored in which the verification of Theorem 3.1 fails even if there is a new matrix A and corresponding bound $\alpha > 0$ determined. In general there are two possible reasons for this failure. Firstly there could be a singularity of higher order than expected. Secondly the unfolding functional μ is not appropriately. Maybe one can try again with a different unfolding functional. Actually the initially considered box $(\boldsymbol{\lambda}^0, \mathbf{u}^0) \in \mathbb{IR}^{p+n}$ is narrow in general. So if we have determined the unfolding functional using the discussion in section 3.8 then it is unlikely that the reason of failure is an inappropriate unfolding functional.

Principally a cause of failure to verify Theorem 3.1 always can be too much overestimation when determining the range of the functions F or its derivative function $D_u F$, but here we must assume that the overestimation is low enough.

(iv) If the unfolding functional $\mu : \mathbb{R}^p \times \mathbb{R}^n \to \mathbb{R}^l$ is given then we assume to either know $D_u\mu : \mathbb{R}^p \times \mathbb{R}^n \to \mathbb{R}^{l\times n}$. Then clearly A_2 is set by $A_2 = D_u\mu(\lambda_0, u_0)$. Otherwise A_2 is determined using the discussion in

section 3 and the unfolding functional is defined by the linear functional $\mu(u) = A_2 \cdot u$. As a consequence then clearly $D_u\mu(u) = A_2$ for all $(\lambda, u) \in \Lambda \times D$.

(v) If $n \geq 0$ is not too large then one may determine the matrix A and the corresponding bound $\alpha > 0$ for every box which has to be analysed. However, for $n \geq 0$ large the explicit computation of the inverse of a large matrix A is very expensive, so one must try to avoid too much of these computations.

(vi) We need an interval matrix $\mathbf{A}^{-1} \in \mathbb{IR}^{(n+l)\times(n+l)}$ such that there exists a real matrix $B \in \mathbf{A}^{-1}$ such that $AB = BA = I$, where $I \in \mathbb{R}^{(n+l)\times(n+l)}$ means the identity matrix. For details see Kearfott [12] or Neumaier [18].

(vii) For the computing of a grid using our discussion in section 3.8 a matrix which approximates the inverse of A is useful. Easily we can get such a matrix by $\mathbb{R}^{(n+l)\times(n+l)} \ni A^{-1} = \operatorname{mid} \mathbf{A}^{-1}$.

(viii) In (a) *and* (b) *it is recommendable to determine appropriate weights $w^1, w^2 \in \mathbb{R}^{n+l}$ for the norms to be used. See section 2.2.*

(ix) If the bound $\alpha > 0$ is not appropriately small then the extended problem may be ill-conditioned. Maybe one should monitor the value $\alpha > 0$.

(x) The function $g(\lambda, \sigma)$ shall give qualitatively good approximations of the solution of the extended problem near $(\lambda_0, \sigma_0) \in \mathbb{R}^{p+l}$. Beside of the appropriate extension of the original problem the good quality of the approximate solution is important for the verification of a box. So maybe the approximation we get by the function $g(\lambda, \sigma)$ should be improved with a Newton-Chord iteration. See section 3.8. The matrix $A^{-1} \in \mathbb{R}^{(n+l)\times(n+l)}$ shall be applicable for Newton-Chord iteration.

By using a Newton-Chord iteration we should be aware that in general a change u implies a change in $\sigma \in \mathbb{R}^l$ because $\sigma = \mu(\lambda, u)$. The situation we have, is to determine an approximation of the solution at some $(\lambda_0, \sigma_0) \in \mathbb{R}^{p+n}$. So we have to take care that for the corresponding approximate solution $(u_0, \xi_0) \in \mathbb{R}^{n+l}$ the value (or vector) $\tilde{\sigma}_0 = \mu(\lambda_0, u_0)$ does not differ too much from the desired $\sigma_0 \in \mathbb{R}^l$. Maybe sometimes a correcting predictor step is needed to get the approximate root for the desired $\sigma_0 \in \mathbb{R}^l$.

We use the discussion of the sections 3.8 and 2.3 to determine the function $g(\lambda, \sigma)$ which approximates the solution of the extended problem with respect to λ and σ. Any other method to determine such a

function which provides a good enough approximation of the solution is applicable too.

(xi) The value $\epsilon > 0$ *in the condition* $\operatorname{rad}(\boldsymbol{\lambda}, \boldsymbol{\sigma}) > \epsilon$ *must be set corresponding to the quality of the approximation of solutions through the function* $g(\lambda, \sigma)$ *and possibly additional Newton-Chord iteration. If the quality of the approximation of a solution* $(u_0, \xi_0) \in \mathbb{R}^{n+l}$ *at* $\operatorname{mid}(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{R}^{p+l}$ *in general do not yield to* $\tilde{\sigma_0} = \mu(\lambda_0, u_0) \in \boldsymbol{\sigma}$ *then it is not usefully to consider boxes of this size.*

(xii) The condition '$\mathcal{L}$ *is not empty or* $\mathcal{S}$ *is not empty' in the second case of* (g) *is needed to avoid an infinite loop in the case when a box can not get verified.*

(xiii) The extension we use provides locally that singularities of the original problem disappear in the extended problem. However we have to be aware that for a root $(\lambda_0, u_0) \in (\boldsymbol{\lambda}^0, \mathbf{u}^0)$ *of the real function* F *we must not obviate the existence of another root* $(\lambda_0, u_1) \in (\boldsymbol{\lambda}^0, \mathbf{u}^0)$ *such that* $\mu(\lambda_0, u_0) = \mu(\lambda_0, u_1)$.

4.7 Existence and uniqueness of the solution path

Here we reference to Neumaier [18] and to Rheinboldt [22].
Here we first assume that we have found a verified box $(\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{IR}^{p+l}$, the corresponding box $\mathbf{u} \in \mathbb{IR}^n$ and we know certainly that a solution branch passes the box. Further corresponding boundary information, two boxes $(\tilde{\boldsymbol{\lambda}}_1, \tilde{\boldsymbol{\sigma}}_1), (\tilde{\boldsymbol{\lambda}}_2, \tilde{\boldsymbol{\sigma}}_2) \in \mathbb{IR}^{p+l}$ and the corresponding boxes $\tilde{\mathbf{u}}_1, \tilde{\mathbf{u}}_2 \in \mathbb{IR}^n$ we assume to know. By applying Algorithm 8 this situation occurs very likely after the first box has been verified. By remembering section 4.3 we know that such a box $(\tilde{\boldsymbol{\lambda}}_1, \tilde{\boldsymbol{\sigma}}_1) \in \mathbb{IR}^{p+l}$ and the corresponding $\tilde{\mathbf{u}}_1 \in \mathbb{IR}^n$ does not provide that a solution branch of the reduced problem $\xi(\lambda, \sigma) = 0$ certainly crosses $(\tilde{\boldsymbol{\lambda}}_1, \tilde{\boldsymbol{\sigma}}_1) \in \mathbb{IR}^{p+l}$ unless it is very likely. As a consequence we must not obviate the case (g) of figure 4 in the simple $p = l = 1$ case. For higher dimensional cases solution manifolds can also occur which do not cross the boundary of the corresponding box. For simplicity and illustration reasons we discuss mainly the simple case, although the consequences shall hold for the higher-dimensional case too.
So we need to prove that a solution branch of the reduced problem is unique and exists which traverses such a box $(\tilde{\boldsymbol{\lambda}}_1, \tilde{\boldsymbol{\sigma}}_1) \in \mathbb{IR}^{p+l}$. We apply Theorem 2.17 to prove existence.

Existence

Here we consider Theorem 2.17 which you can find in section 2.4
The interval matrix $\mathbf{A} \in \mathbb{IR}^{n\times n}$ is defined by $\mathbf{A} = F'(\mathbf{x})$ and the preconditioning matrix $C \in \mathbb{R}^{n\times n}$ shall approximate the inverse of $\mathbf{A} \in \mathbb{IR}^{n\times n}$. This interval matrix $\mathbf{A}$ satisfies the condition to be a Lipschitz matrix (see for details Neumaier [18]). Generally $C \in \mathbb{R}^{n\times n}$ is a real matrix. The term $\mathrm{int}(\mathbf{x}_0)$ means the interior of a set represented by $\mathbf{x}_0 \in \mathbb{IR}^n$.
We have to take care on rigour, so in application we interpret $x \in \mathbb{R}^n$ as an interval.

Remark.
The Krawczyk operator defined in Theorem 2.17 also could be used to improve the enclosure $\mathbf{u} \in \mathbb{IR}^n$ *in Algorithm 2. There are also better operators than the Krawczyk operator, for example the* ***Hansen-Sengupta operator*** *which is similar to the ordinary Gauss-Seidel method. For more see Neumaier [18] and Kearfott [12].*

We apply this theorem on our original problem, this means on a $k_0 \geq 1$ times continuously differentiable function $F : \mathbb{R}^p \supseteq \Lambda \times D \to \mathbb{R}^n$. Actually on the function $F_\lambda : D \to \mathbb{R}^n$ where $\lambda \in \tilde{\boldsymbol{\lambda}}_i$ is held fixed and the interval vector $\mathbf{x}_0$ in the theorem we replace by $\tilde{\mathbf{u}}_i \in \mathbb{IR}^n$ and x by $\operatorname{mid} \tilde{\mathbf{u}}_i \in \mathbb{R}^n$ $(i = 1, 2)$.
Actually if the interval matrix $\mathbf{A}$ is not (near) singular then this existence verification shall not fail due to overestimation of the range of the functions F or $D_u F$ because the construction of Algorithm 5 provides very narrow boxes $\tilde{\mathbf{u}}_1 \in \mathbb{IR}^n$.
The next question to consider is the local uniqueness of solution branches.

Uniqueness

We apply the Theorem 2.18 which you can find in section 2.4 on the original problem, more exactly on the function $F : \mathbb{R}^p \supseteq \Lambda \times D \to \mathbb{R}^n$ at $(\tilde{\boldsymbol{\lambda}}_i, \tilde{\mathbf{u}}_i) \in \mathbb{IR}^{p+l}$ $(i = 1, 2)$.
If such a continuous function H like in Theorem 2.18 exists for F then we call this continuous function u and $F(\lambda, u(\lambda)) = 0$ holds for all λ in the considered area. Then this clearly implies $\tilde{F}(\lambda, \mu(\lambda, u(\lambda)), \tilde{u}(\lambda), \xi(\lambda))) = 0$ where $\tilde{u}(\lambda) = \tilde{u}\,(\lambda, \mu(\lambda, u(\lambda))) = 0$ and $\xi(\lambda) = \xi(\lambda, \mu(\lambda, u(\lambda))) = 0$ (by remembering the notation of section 4.2) for all these λ and the functions $\tilde{u}$ and ξ are continuous as compositions of continuous functions.
Now we consider the application of this theorem for the simple case, $p = l = 1$. More we say that the part of the boundary we are considering is $[\overline{\lambda}, \tilde{\boldsymbol{\sigma}}] \in \mathbb{IR}^2$ where we remember $\boldsymbol{\lambda} = [\underline{\lambda}, \overline{\lambda}]$. Obviously $\overline{\lambda}$ is interpreted as an interval. Intuitively we see that $\overline{\lambda}$ is a point or a thin interval so the function

$u : \overline{\lambda} \to \mathbb{R}^n$ maps to only one point in $\mathbb{R}^n$. However, the theorems above are implications of the implicit function theorem (see proofs in Neumaier [18]), so the validity of $F(\lambda, u(\lambda)) = 0$ remains in an open neighbourhood of $\overline{\lambda} \in \mathbb{R}$. As a consequence the solution branch of the reduced problem $\xi(\lambda, \sigma) = 0$ is $\mu(\lambda, u(\lambda))$ in this neighbourhood and must traverse $[\overline{\lambda}, \tilde{\boldsymbol{\sigma}}] \in \mathbb{IR}^2$.
So by remembering the situation described at the beginning of this section we can imply the traversing of the solution branch of $(\tilde{\boldsymbol{\lambda}}_2, \tilde{\boldsymbol{\sigma}}_2) \subseteq \partial(\boldsymbol{\lambda}, \boldsymbol{\sigma})$ if we have proven that the solution branch traverses $(\tilde{\boldsymbol{\lambda}}_1, \tilde{\boldsymbol{\sigma}}_1) \subseteq \partial(\boldsymbol{\lambda}, \boldsymbol{\sigma})$ (see figure 11) due to the continuity of the solution branch (see section 4.2). Colloquially spoken a solution branch which traverses the boundary of a box must not disappear in the box.
Unluckily we can not imply the uniqueness in $(\tilde{\boldsymbol{\lambda}}_2, \tilde{\boldsymbol{\sigma}}_2)$, although it is very unlikely that there is another solution branch which traverses $(\tilde{\boldsymbol{\lambda}}_2, \tilde{\boldsymbol{\sigma}}_2)$ because this branch would have to traverse $(\tilde{\boldsymbol{\lambda}}_2, \tilde{\boldsymbol{\sigma}}_2)$ twice for not harming the condition of not disappearing in the verified box.

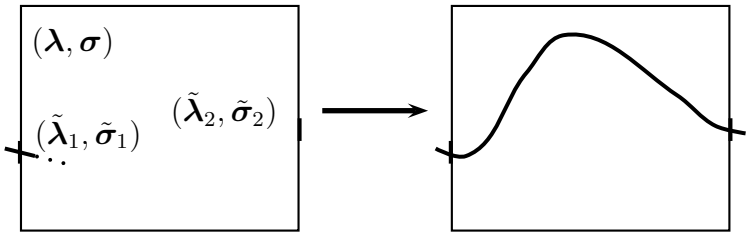

Figure 11: Continuation of a solution branch I

As long as the boxes we verify always provide only one additional part of its boundary where the solution branch might cross we can imply that the solution branch continues through this parts of the boundary and moreover we assume uniqueness. Next if we find a box where we suspect a (near) bifurcation, i.e. the boundary information we compute is more than one additional part of the boundary the box (see figure 12). Then we have to apply Theorem 2.18 for all areas $(\mathbf{2}), (\mathbf{3}), (\mathbf{4}), (\mathbf{5})$ where the solution branch might traverse. If we verify Theorem 2.18 at $(\mathbf{2})$ and we still know that this theorem holds at $(\mathbf{1})$ then this implies that a solution branch between $(\mathbf{1})$ and $(\mathbf{2})$. Clearly we have assumed that between $(\mathbf{1})$ and $(\mathbf{2})$ we have not verified a box where a singularity is suspected. However we can not say anything about the uniqueness of the solution path between $(\mathbf{1})$ and $(\mathbf{2})$.
If Theorem 2.18 is verified at $(\mathbf{2}), (\mathbf{3}), (\mathbf{4}), (\mathbf{5})$ this implies a (imperfect) bifurcation in the considered box for the reduced problem and due to construction of the extended problem also a (imperfect) bifurcation in the original problem. Unluckily there is no way to decide if we have the situation of a perfect

or an imperfect bifurcation.

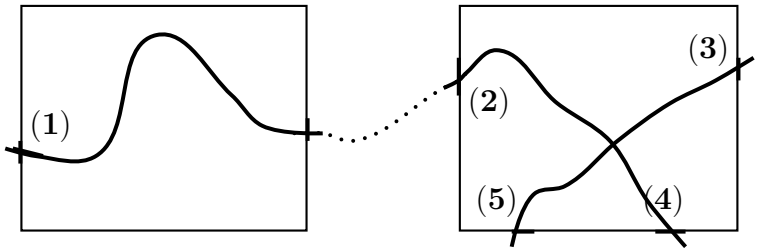

Figure 12: Continuation of a solution branch II

We know that the Jacobian-matrix $D_u F(\lambda_0, u_0)$ at a bifurcation point $(\lambda_0, u_0) \in \Lambda \times D$ is not invertible, so the Jacobian near such a singular Jacobian might be ill-conditioned. So it would not be surprisingly if the verification of Theorem 2.18 near a bifurcation point fails, because of an ill-conditioned Lipschitz-matrix **A**. Without loss of generality we say that the verification of Theorem 2.18 fails at (**3**). However, then we can try again at the next box we verify (see figure 13).

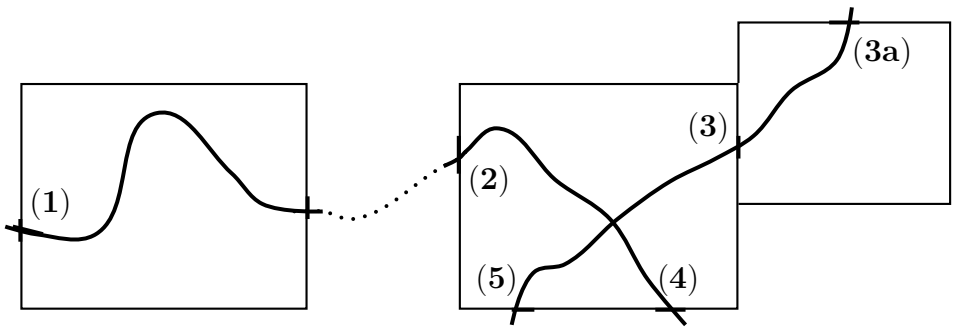

Figure 13: Continuation of a solution branch III

The verification of Theorem 2.18 at (**3a**) instead of at (**3**) then also implies an (imperfect) bifurcation.
We see by additionally using Theorem 2.18 we get enclosures for areas where a solution branch of the reduced and in the following of the original problem is unique and exists continuously and we get additionally enclosure for areas where an (imperfect) bifurcations occur.

Remarks. *(i) We could apply this theorem also on each verified box. Then the uniqueness can be shown everywhere. However the verification of uniqueness in this way would not work if a turning point occurs.*

(ii) The same arguments should also work in a very similar way for the higher-dimensional case when $p > 1$ or $l > 1$. If there is $l > 1$ then

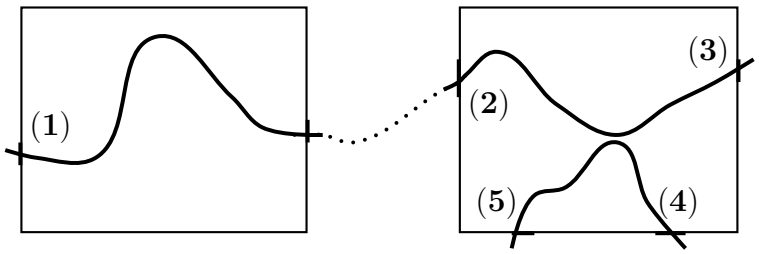

Figure 14: Continuation of a solution branch IIa

more than four parts of the boundary can exist where a solution branch traverses, in particular $2(l+1)$ *such parts are possible if the considered box is narrow enough.*

(iii) If the box is not narrow the implication of a perfect or imperfect bifurcation would not be reasonable, but we previously stated in the last preceding section that the assumption of a narrow box is reasonable.

(iv) For the illustration of the non-decidability of perfect or imperfect bifurcation see figure 14.

5 Implementation and examples

For the implementation we use the software MATLAB. Here we do not go into the features which MATLAB provides because this would be far too much. We assume that the reader is familiar with the basic usage of MATLAB. However we introduce the basic features of a toolbox for MATLAB which provides rigorous interval computations. This toolbox is called INTLAB.

5.1 Implementation

First we start with the introduction of INTLAB. This toolbox from Rump provides the needed rigorous interval computations. In particular the computations use the outward rounding which is described in section 2.1 to beware rigour.

INTLAB

We start with the commands to define an interval. In MATLAB this type of data structure is called *intval*. These commands are `infsup(a,b)` as well as `midrad(m,r)` and `intval(x)`. The `infsup` command provides an interval matrix with component-wise lower bound `a` and upper bound `b`. Obviously then `a,b` must be matrices, in MATLAB of data-type *double*, with the same size and `b` must be greater or equal than `a` component-wise. The `midrad` command essentially provides the same as the previously defined operator with the same name. The input of type *double* `m` is the midpoint of the provided interval and `r` the radius. Obviously `r` must be positive and either a scalar, i.e. a *double* of size 1, or of the same size as `m`. Moreover the command `intval` gives us the possibility to convert a matrix or better a *double* `x` into its interval representation. This conversion must not lead principally to a thin interval due to rounding issues (see section 2.1). The input can either be a *double* or a *string* which represents a *double* object.
In our main reference for INTLAB Rump [23] is suggested that the command `intval` leads to a different result for *double* and *string* as input, but according to the corresponding help in MATLAB in the actual version of INTLAB the type of input should not lead to a different result anymore.
Further INTLAB provides the commands `inf, sup, mid, rad` where the result is quite obvious and we do not write this down in detail. One has to be aware of the fact that the command `inf` in another context in MATLAB means the representation of infinity.
Further important commands are `intersect(a,b)`, `emptyintersect(a,b)` and `hull(a,b)`. The first two commands provide the component-wise in-

tersection of two interval matrices of same size and `emptyintersect` gives additionally a boolean matrix of the same size. If an entry in the boolean matrix is true then the intersection of the intervals corresponding to this entry is empty. In particular if we consider two interval vectors, i.e. two interval boxes then their intersection must not be empty in any entry so that the intersection of the sets represented by the interval vectors is not empty. For the command `hull` one has to be aware that the hull of the sets represented by the two interval boxes is not equal to the union of two intervals boxes even if the union actually is simply connected. So one has to be very careful with this command. The command `hull` for two intervals `a,b` yields to an interval where the lower bound is the minimum of both lower bounds and the upper bound is the maximum of the upper bounds.
Most important INTLAB provides rigorous enclosures of the range for the elementary operations as well as for the range of the most common elementary functions (see section 2.1). Moreover INTLAB has two modes of interval matrix multiplication. The first mode, called *FastIVMatrixMultiplication*, provides a fast but no very sharp interval matrix multiplication. The more advisable mode in our application is called *SharpIVMatrixMultiplication* and provides a much sharper result with the expense of a slower execution. However in the case of global optimization the sharpness is more important. Especially if both multiplicands are thick interval matrices the sharper interval matrix multiplication is advisable. The mode are initialised by the command `intvalinit('SharpIVMatrixMultiplication')` respectively `intvalinit('FastIVMatrixMultiplication')`. The standard mode is *FastIVMatrixMultiplication*.
Further a call like

$$\texttt{x} = \texttt{intval}(\texttt{0.125}) + \texttt{1/10};$$

results into `x` to be a *intval*. However the result is not correct in the sense we probably want it to be because `0.225` is not contained in the interval represented by `x`. By calling

$$\texttt{x} = \texttt{0.125} + \texttt{intval}(\texttt{1})/\texttt{10};$$

we would get a correct result. However to ensure correct results we should care that all quantities in a call are of type *intval* by converting *double* objects with the command `intval` into *intval* objects. We mentioned in section 2.1 that naive interval evaluation of function can lead to large overestimation of the range. As a consequence the common interval evaluation with INTLAB can also lead to such a large overestimation. To improve this problem INTLAB contains toolboxes for slope arithmetic and automatic differentiation which yields to mean value forms. Here we do not discuss this toolboxes

because the examples chosen to illustrate the previously provided method would not be improved by this toolboxes. However in general centred forms are advisable if one needs to evaluate the enclosure of the range of more complicated functions. Then slope arithmetic or automatic differentiation are useful.
At least we need the bound $\alpha > 0$ and for determining this bound an enclosure of the inverse matrix of the matrix A is needed. INTLAB can solve interval systems of linear equations, this means clearly that an enclosure of the solution is provided. The command is called `verifylss` and by using this procedure the common MATLAB command `inv` is overloaded for *intval* objects. So we can commonly call the command `inv(A)` which results in an enclosure of the inverse of the matrix A.
For more details on INTLAB we first refer to Rump [23] and the corresponding homepage. There you can find a lot of references on the usage of INTLAB. Further if you install INTLAB there one can find some demos which are useful when you begin to use INTLAB. More information on the mentioned commands you can find with the `help` command in MATLAB.

Own Implementation

Here I list classes and functions which I have written in MATLAB using INTLAB to apply the previously discussed topics with a short description of the usage of the corresponding class or function.

`funk`:
This class stores the function name *name* and the vectors A_1 and A_2 which one is using at the moment. The considered evaluation of the function F and its derivatives must be stored in the following way `f`*name*`.m` respectively `dx`*name*`.m` respectively `dlambda`*name*`.m`. If such a *funk* object is called `f` then we can evaluate the wished original or extended function or its derivative by the commands `f.f(·), f.dx(·), f.dlambda(·), f.F(·), f.Dx(·), f.Dlambda(·), f.Dsigma(·)`. A capital letter always means the extended function.

`findA1`, `findA2`:
This functions find appropriate vectors A_1 respectively A_2 using the discussion in the sections 3.2 and 3.3.

`Gridd`:
This class finds in a box defined by an interval vector `lsbox` for some points `gridpoints` (λ, σ) corresponding function values or (vectors) `y` and values or

(vectors) for the corresponding derivatives `dlambda, dsigma`. In particular a corrector step method `corrStep` and a predictor step methods `predStep` corresponds to this class. This class bases on the discussion in sections 3.8.

`BezPolynom, TriKubBezPolynom, TriKubBezInterp`:
A *BezPolynom* object represents a Bezier polynomial on an arbitrary simplices with its coefficients and the vertices of the corresponding simplices. The class `BezPolynom` has a method to evaluate the polynomial (see Algorithm 1, de Casteljau). A *TriKubBezPolynom* object represents a triangular cubic Bezier polynomial like in the discussion of section 2.3. Such an object is determined by the vertices of the corresponding triangle and the coefficients which are illustrated in figure 3. The class `TriKubBezPolynom` is a subclass of `BezPolynom`. The class `TriKubBezInterp` patches such *TriKubBezPolynom* objects together to a Bezier surface. The coefficients are determined by using Farins method from section 2.3 based on the information we get by a *Gridd* object. Here we use the class `DelaunayTri` and its methods of MATLAB. This class provides a Delaunay triangulation. At the boundary of the triangulation we determine additional gradient vectors to find good coefficients for the corresponding Bezier polynomials.
These are all the preprocessing functions I use in my implementation.

Remark.
Due to some reasons I mention later I do not analyse the boundary edges of every box the algorithm is verifying. I check the boundary of bigger boxes only which are completely checked (see figure 15). Here the boxes filled green and

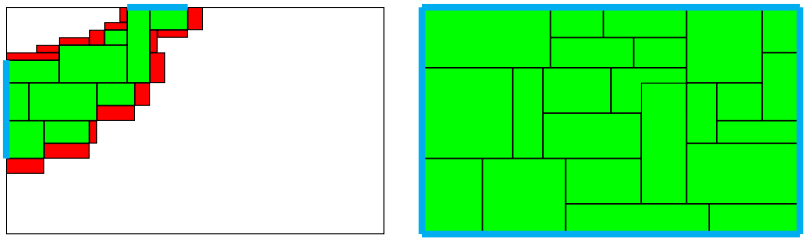

Figure 15: Illustrations of possible situations through my implementation

red has been verified by Algorithm 2. Through the green boxes a solution path crosses very likely and the red boxes restricting the area where the solution path traverses. In particular in the red boxes we know $0 \notin \boldsymbol{\xi}$ *(see Algorithm 2). The cyan marked segment of the boundary of the big box marks the area we have to analyse with Algorithm 5. Based on the information we get by Algorithm 5 we may choose the next bigger box to consider.*
We see that depending on the choice of the size of such a bigger box either

the first or the second situation occurs. The reason why I do not check the boundary of each sub-box is that on the one hand this analysing of the boundary is quite expensive and needs an even higher accuracy than the common check for verification of a small box. On the other hand the separately considering of bigger boxes helps to keep the administration of the small subboxes smaller. Otherwise this administration gets very large and as a consequence the procedure gets slower and slower.
So I assume inside a bigger box that every small box which shares an edge with a already verified small box where the solution path might crosses, must be checked by Algorithm 2 without analysing the boundary of each verified small box.
For the continuation with further bigger boxes see figure 8. The parameters respectively the size of this bigger box depends on the considered problem and one must try to find a suitable size.

`BoxManagement`:
This class is responsible for the administration of all sub-boxes in a bigger box. This means there are the main properties `verBoxes` and `notVerBoxes`. Not surprisingly in `verBoxes` are the verified boxes with the corresponding enclosure is stored. In `notVerBoxes` every box is stored which have not been verified yet. The boxes stored in these two properties cover the whole bigger box which is considered. I use the MATLAB class `containers.Map` for these properties. Further the class `BoxManagement` stores the information which box shares a boundary edge with another box. As a consequence of this the whole algorithm knows which boxes must be verified yet. At least this class `BoxManagement` also stores and manages the boundary information of the considered bigger box. The mentionable methods which accord to this class are `modify` and `plot`. The command `modify` modifies a *BoxManagement* object for the different situations which can occur (see Algorithm 7). Further the command `plot` provides a plot in a very similar way like in the figure 2 above.

`VerifiedBox`:
This class is the main and most important part of the whole procedure. This class aims to completely analyse such a bigger box similar to Algorithm 8. The most important properties of this class are `initbox` which is a *intval* object which restricts the considered box. Then there is a property `BoxM`, a *BoxManagement* object. Above I have describe the function of such an object. Further there is a property `BezSurface`, a *TriKubBezInterp* object. The function of such an object has been described above too. Further there are lists of integers `flist`, `blist` which are connected to the prop-

erty `notVerBoxes` in `BoxM`. The integers in `blist` connect to boxes which are marked red in figure 2. In the list `flist` integers are stored which corresponds to boxes in `notVerBoxes` in the object `BoxM` for which our verification algorithm has not been successfully. The reasons why this can happen have been discussed section 4.
The key methods this class contain are `try2Verify` and `boundaryCheck`. The key method `try2Verify` does the same like the Algorithm 7 without analysing the boundary edges of the considered small box. The method `boundaryCheck` analyses the boundary edge of the considered bigger box. Here see figure 2. This methods checks the cyan marked edges and tries to restrict the area where a solution path traverses more.
Further there is a method for calculating the bound $\alpha > 0$ and the matrix A, `calcAlpha`, a simple method to find appropriate weights, `findWeights`, a method for a possibly needed corrector iteration, `corrStep`, and methods for bisection and complementation, `bisect` and `takeComplement`.

`Verifying`:
This class starts the whole algorithm. As input we need the function name to initialise a object of type *func*, a starting point, i.e. a approximate root of the considered function, and an *intval* object to restrict the area in which we want to determine the solution path. Optionally one can give a unfolding functional. This class manages the bigger boxes and initialises the class `VerifiedBox` for appropriate boxes. The main property is called `verList`. In `verList` objects of the type *VerifiedBox* are listed. The class uses the boundary information which is stored in this *VerifiedBox* objects to find new boxes which must be analysed. The class also tries to avoid that some areas are double checked.
Further the class `Verifying` contains a method `plot` which executes `plot` for all *VerifiedBox* objects stored in `verList`.

Remarks. *(i) At least there is the possibility to initialise a new vector A_1 for each box we want to analyse with* `VerifiedBox` *or we can initialise at the beginning of* `Verifying` *once a vector A_1 and keep this. If the considered area is large it is possibly recommendable to initialise A_1 each time new else once determining such a vector is enough. The unfolding functional μ should be kept in the whole process because otherwise the bifurcation diagram is not be meaningful. Often there is a linear unfolding functional, maybe determined by the function* `findA2`*, and then as a consequence the vector A_2 does not change during the algorithm.*

(ii) In the implementation we have to try that the administration of the boxes does not slow down the program. The analysing of the bound-

ary need high accuracy to give useful results and as a consequence the `boundaryCheck` *method is quite expensive. The cost of the command to verify a box is dependent on the domain of the function we are considering. Here we need to compute enclosures of the range of the function by using interval analysis. These computations are expensive if we are considering a function with a domain of high dimension.*

(iii) For the Bezier surface to interpolate the solution manifold we should not use a grid with a large amount of points because otherwise the evaluation of the Bezier surface gets expensive and slows down the program.

5.2 Examples

First we start with the simple example of the common pitchfork bifurcation.

Example 1. *The problem we are considering is*

$$x^3 - \lambda x = 0 \tag{63}$$

where $x, \lambda \in \mathbb{R}$. Obviously one solution path can be written down explicitly by $x(\lambda) = x = 0$. The second solution path we can write down by $x(\lambda) = \pm\sqrt{\lambda}$. We can not give an explicit expression for the whole solution path of the second branch because at $(\lambda, x) = (0, 0)$ we have a turning point with respect to λ. However our algorithm should not have a problem through the turning point as well as the bifurcation point at the same point.
Obviously we can also write down a explicit expression for the second branch with respect to x. This is $\lambda(x) = x^2$. So it is reasonable to choose the unfolding functional $\mu : \mathbb{R} \to \mathbb{R}$ such that $\mu(x) = x$. Then we have $\sigma = x$ and $A_2 = 1$.
By applying this algorithm and plotting the result we get the following figures. As starting point we chose $\lambda = 1$ and a random number $x \approx 0$.
We see clearly that the algorithm has found the wished solution branches. In the green marked area the solution path proceeds and further the illustration says that the solution path does not traverse red marked areas and lines. Moreover the blue points (areas) we got by analysing the boundary of such bigger boxes. So the blue areas restrict the continuation of the solution paths very much. In figure 17 I zoomed in near the bifurcation point.
By applying successfully the results of section 4.7 on the blue marked parts of the edge of the middle box we can show that a (near) bifurcation point is contained in this box. The smaller the box is the likelier a reel bifurcation point is contained in this box. By applying the results of section 4.7 on more such parts of edges of boxes which are marked blue one can show existence and uniqueness of the solution branches beside the box where the (near) bifurcation point occurs.

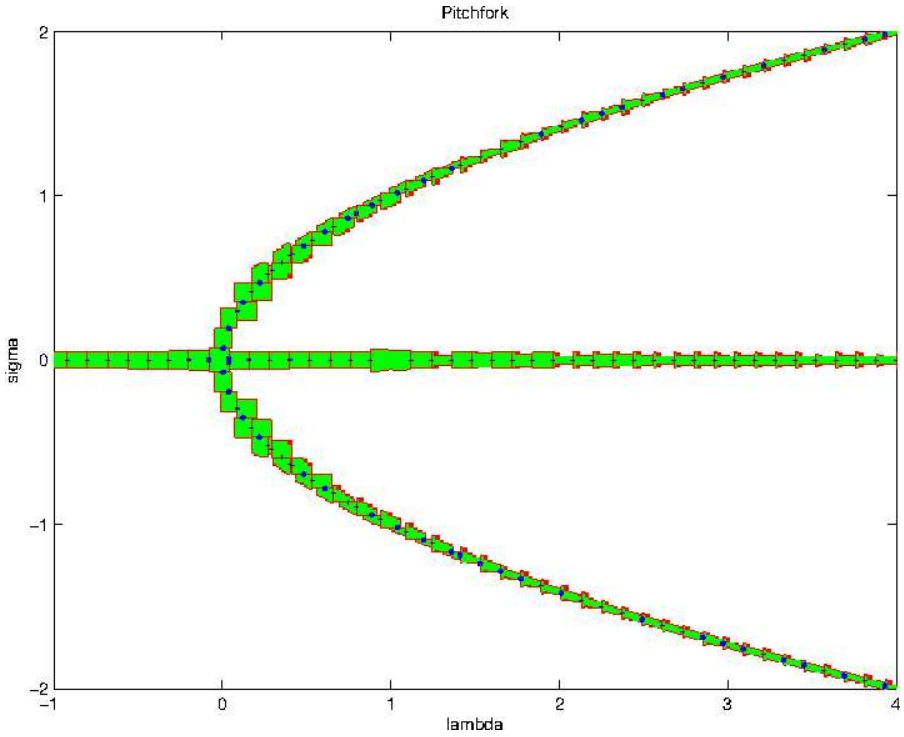

Figure 16: Pitchfork

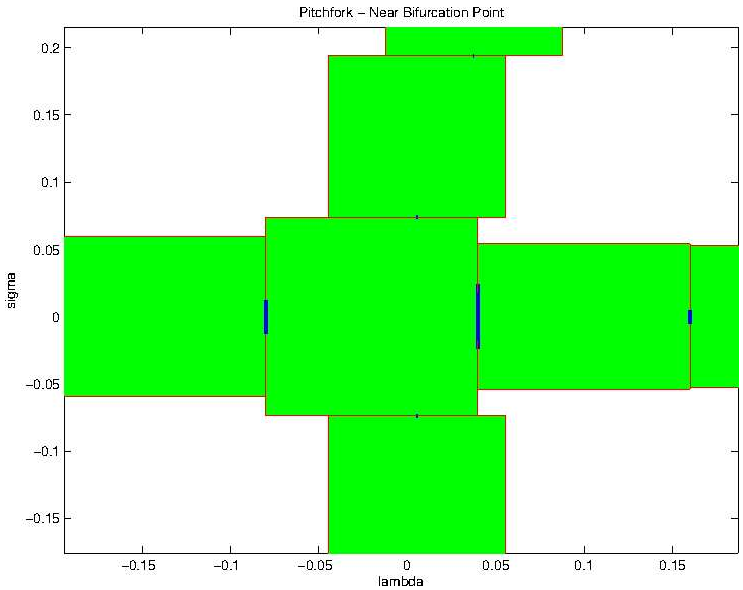

Figure 17: Pitchfork - near bifurcation

Here we determined at the beginning of the algorithm the linear operator respectively the vector A_1 and did not change it during the algorithm.

To prepare the next example we rephrase first a theorem of Chow and Hale [3].

Theorem 5.1. *Let $F : \mathbb{R} \times \mathbb{R}^n \supseteq \Lambda \times D \to \mathbb{R}^n$ a $k \geq 2$ times continuously differentiable function. Suppose that*

$$F(\lambda, x) = Bx - \lambda x + N(\lambda, x) \tag{64}$$

$$N(\lambda, 0) = 0, \quad N_x(\lambda, 0) = 0 \tag{65}$$

where $B \in \mathbb{R}^{n \times n}$ and N a nonlinear function. If λ_0 is a simple eigenvalue of B with eigenvector $y_0 \neq 0$, then $(\lambda, x) = (\lambda_0, 0)$ is a bifurcation point of $F(\lambda, x) = 0$.

Example 2. *Here we are considering the function $F : \mathbb{R} \times \mathbb{R}^3 \to \mathbb{R}^3$,*

$$F(\lambda, x) = Ax - \lambda x + \lambda \begin{pmatrix} b_{11}x_1^3 + b_{12}x_1x_2 + b_{13}x_1x_3 \\ b_{21}x_1x_2 + b_{22}x_2^3 + b_{23}x_2x_3 \\ b_{31}x_1x_3 + b_{32}x_2x_3 + b_{33}x_3^3 \end{pmatrix} \tag{66}$$

where

$$A = \begin{pmatrix} -1 & 0 & 1.2 \\ -1 & 1 & 0.01 \\ 0.22 & 1 & -1.3 \end{pmatrix}, \quad B = \begin{pmatrix} 1.23 & 0.8 & -1.01 \\ -0.04 & 0.4 & 0 \\ -4 & 1 & -0.9 \end{pmatrix}.$$

Not surprisingly this function fits for the theorem above and so we know by determining the eigenvalues of the matrix A where we must expect simple bifurcations. The eigenvalues are approximately $-1.982, 0.5374$ and 0.1446. So we apply the implementation on this function starting at $\lambda = 0.4$ and $x \approx 0 \in \mathbb{R}^3$ and hoping to find the solution branches bifurcating approximately at $(0.5374, 0) \in \mathbb{R} \times \mathbb{R}^n$ and $(0.1446, 0) \in \mathbb{R} \times \mathbb{R}^n$. Here the vector A_1 changes for every bigger box the algorithm considers. The unfolding functional we determined by the discussion of section 3 and more exactly with the method `findA2` *is*

$$\mu(u) = \begin{pmatrix} -0.7265 \\ -1.1438 \\ -0.7995 \end{pmatrix} \cdot u.$$

In the figures below the σ-value is determined by this unfolding functional. In the figure 18 we see that the expected behaviour of the solution branches is detected by our algorithm. Moreover we see that the same solution branch bifurcates approximately at $(0.5374, 0) \in \mathbb{R} \times \mathbb{R}^n$ and $(0.1446, 0) \in \mathbb{R} \times \mathbb{R}^n$ of the trivial solution branch $u = 0$.

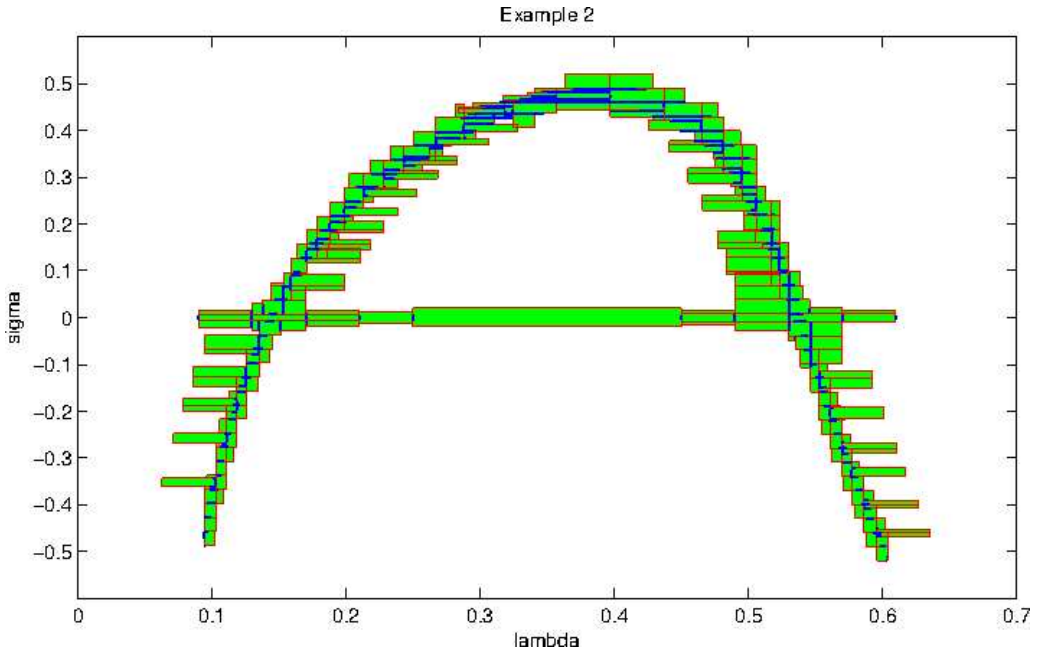

Figure 18: Example 2

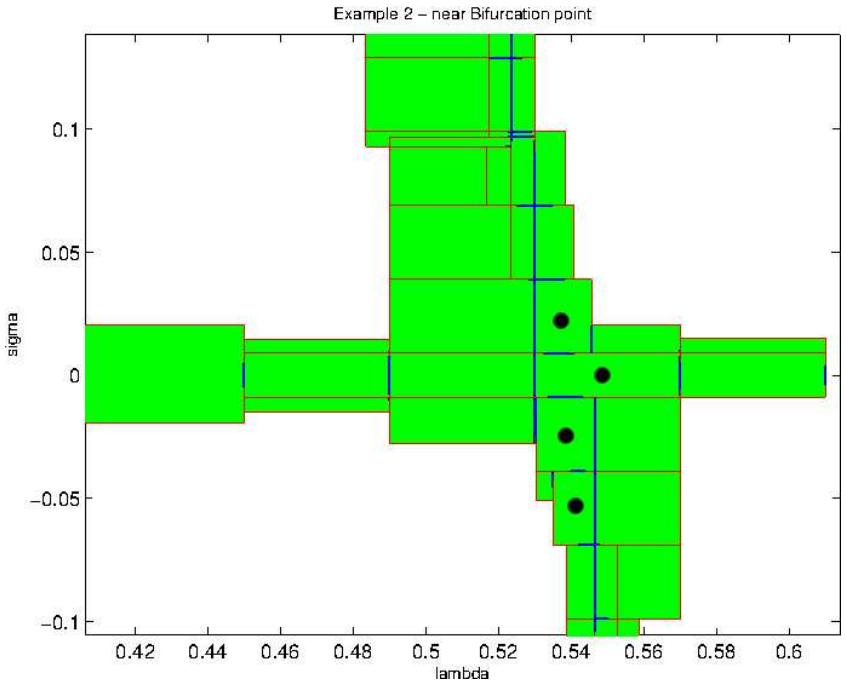

Figure 19: Example 2 - near bifurcation point

In figure 19 we zoom in near the bifurcation point $(0.5374, 0) \in \mathbb{R} \times \mathbb{R}^n$. *Here I marked the boxes with black points where four parts of the boundary edge has been found where possibly a solution branch traverses. However on closer examination we see that beside of the box where we know through Theorem 5.1 that a bifurcation point occurs the possible solution branches would traverse into boxes where no further part of the boundary is found where a solution branch can traverse. By this observation it gets very unlikely that a solution branch really crosses this edge and as a consequence it is very unlikely that a bifurcation point occurs in such a box.*

By increasing the accuracy of the computations one could get even narrower enclosures and better guesses for the areas where a solution branch traverses the edge of a box. However I think the illustrations make clear how the method works and shows the applicability.

6 Further comments and prospects

First we mention that all results are possibly improvable by using an appropriate branch and bound scheme. Due to the fact that in general branch and bound is very expensive one may have to think about good splitting strategies. For more we refer exemplary to Ratz and Csendes [21] or Csendes and Ratz [5].
Next there is the possibility to use the operators we have mentioned in the previous section, the Krawczyk operator or the Hansen-Sengupta operator, to improve the enclosures we get by our algorithms. More details one can find in Neumaier [18] or Kearfott [12].
Further the discussion of section 2.3 can be used to interpolate lower and upper bound surfaces for the solution paths.
At the beginning of the work we claimed the matrix A to be invertible and the norm the inverse of A to be bounded by an $\alpha > 0$ (see (4)). We mentioned that our main Theorem 3.1 still works if the matrix only has a left-inverse for which the norm must be bounded. In the main results of section 4 we claimed the considered function F to have the form $F : \mathbb{R}^p \times \mathbb{R}^n \to \mathbb{R}^n$ and as a consequence then $\mu : \mathbb{R}^p \times \mathbb{R}^n \to \mathbb{R}^k$ and $A_1 : \mathbb{R}^l \to \mathbb{R}^n$ with $k = l$. The condition $k = l$ is necessary because otherwise the matrix A would not be quadratic and as a consequence not invertible. However, the presented results should work for a non-quadratic matrix A too if the matrix A is left-invertible and the norm of its inverse is bounded and one can show that the considered solution paths are still continuous.
In the original work, Neumaier [19], the considered function F is not claimed to be reel. The domain $D \times \Lambda \subseteq \mathbb{R}^p \times \mathbb{R}^n$ and the co-domain $\mathbb{R}^m$ of F can be replaced by $D \times \Lambda \subseteq \mathbb{R}^p \times X$ and Y where X, Y are finite- or infinite-dimensional Banach spaces. Then the same results like in section 1 remain true with the additional assumption that the Frechet derivative $D_u F(\lambda_0, u_0)$ is a Fredholm operator. In general this means that the results of section 1 are applicable also on a wider range of problems. For example on suitable differential equations.

7 Header

7.1 class funk

```
classdef funk
    properties ( SetAccess = protected )
        name % function names
             % function evaluations in m-files:
             % fname.m, dxname.m, dlambdaname
    end
    properties
        A1 % vector to extend A1
        A2 % vector to extend A2
    end

    methods
        function obj = funk(n)
            % to initialise a funk object
        end

        y = f(obj,lambda,x)
        % function evaluation

        y = dx(obj,lambda,x)
        % jacobian evaluation w.r. to x
        y = dlambda(obj,lambda,x)
        % jacobian evaluation w.r. to lambda

        y = F(obj,lambda,sigma,x,xi)
        % extended function evaluation

        y = Dx(obj,lambda,x)
        % extended jacobian w.r. to x

        y = Dlambda(obj,lambda,x)
        % extend jacobian w.r. to lambda

        y = Dsigma(obj)
        % extend jacobian w.r. to sigma
        % no input needed if unfolding functional is linear
    end
end
```

7.2 function findA1

```
function A1 = findA1(fun, lambda, x)

```

```
% Computes vector A1 to extend the problem
% Uses eigenvector method

% INPUT
% fun: funk object to evaluate the considered function
% lambda: value of lambda (at approximate root)
% x: value of x (at approximate root)

% OUTPUT
% A1: vector to extend the problem
```

7.3 function findA2

```
function A2 = findA2(fun,lambda,x)

% Computes vector A2 to extend the problem
% Uses eigenvector method

% INPUT
% fun: funk object to evaluate the considered function
% lambda: value of lambda (at approximate root)
% x: value of x (at approximate root)

% OUTPUT
% A2: vector to extend the problem
```

7.4 class Gridd

```
classdef Gridd
    properties
        gridpoints % (lambda,sigma) gridpoints
        y % corresponding function values
        dlambda % corresponding gradients w.r. to lambda
        dsigma % corresponding gradients w.r. to sigma
        fun % funk obj  to evaluate considered function
        invA % preconditioning matrix
        lsbox % interval box in which the grid is
    end
    properties ( Hidden )
        epsilo % bound for the min steplength
        epsilonsigm % bound for the change of sigma
        varepsilo % bound
        inith % initial steplength
    end

    methods
```

```
19          function obj = Gridd(fu, box, AA,  start, xi0)
20              % initialise Gridd object
21          end
22
23          [succ, obj] = corrStep(obj, lambda, sigma, x, xi)
24          % corrector step
25          % INPUT
26          % lambda, sigma, x, xi: initial value
27          % OUTPUT
28          % stored int the object obj
29          % succ: succesfull or not
30
31
32          [lambda, sigma, x, xi, obj] =
33          predStep(obj, lambda, sigma, x, xi, h1, h2, mod)
34          % predictor step
35          % INPUT
36          % lambda, sigma, x, xi: initial value
37          % OUTPUT
38          % stored int the object obj
39
40          obj = predCorrS(obj,lambda,sigma,x,xi,mod,neg)
41          % predictor-corrector step, sigma direction
42          % INPUT
43          % lambda, sigma, x, xi: initial value
44          % mod: last step or not
45          % neg: steplength positive or negative
46          % OUTPUT
47          % stored int the object obj
48
49          obj = predCorrW(obj,lambda,sigma,x,xi,mod,neg)
50          % predictor-corrector step, lambda direction
51          % INPUT
52          % lambda, sigma, x, xi: initial value
53          % mod: last step or not
54          % neg: steplength positive or negative
55          % OUTPUT
56          % stored int the object obj
57
58          obj = predCorrWs(obj,lambda,sigma,x,xi,mod,neg1, neg2)
59          % predictor-corrector step, diagonal direction
60          % INPUT
61          % lambda, sigma, x, xi: initial value
62          % mod: last step or not
63          % neg1: steplength w.r. to lambda positive or negative
64          % neg1: steplength w.r. to sigma positive or negative
65          % OUTPUT
66          % stored int the object obj
67
```

```
68          obj = calcGrid(obj, lambda, sigma, x, xi)
69          % initialises the calculation
70          % INPUT
71          % lambda, sigma, x, xi: initial value
72          % OUTPUT
73          % stored int the object obj
74
75          [succ, obj] = initStep(obj, lambda, sigma, x, xi)
76          % first step of calculation
77          % INPUT
78          % lambda, sigma, x, xi: initial value
79          % OUTPUT
80          % stored int the object obj
81          % succ: succesfull or not
82      end
83  end
```

7.5 class BezPolynom

```
1   classdef BezPolynom
2
3       properties
4           coeff
5           % coeff ... coefficients of a Bezier patch
6           % [i,j,k], ... indices of coeff,
7           % here keys correspoding to a vector
8       end
9       properties (SetAccess = protected)
10          tri %simplex defined by its vertices stored in tri
11          degree %degree of the patch
12      end
13      properties (Dependent, SetAccess = protected)
14          dimstart % dimension of the domain
15          dimziel %dimension of the co-domain
16      end
17
18      methods
19          function obj = BezPolynom(c,t,deg)
20              % initialises the BezPolynom object
21              % c defines the coefficients
22              % t ... dimension of the domain
23              % deg ... degree of the patch
24          end
25
26          val = get.coeff(obj)
27          % get method for coeff
28
```

```
        val = get.degree(obj)
        % get method for degree

        val = get.dimstart(obj)
        % get method for dimstart

        val = get.dimziel(obj)
        % get method for dimziel

        val = bezVal(obj,x);
        % evaluation of the patch at a point x

        ind = makeindex(obj);
        % get indices of the patch
    end
end
```

7.6 class TriKubBezPolynom

```
classdef TriKubBezPolynom < BezPolynom
    properties
        Ci % coefficients  C_i
        I1 % coefficients I_01, I_11, I_21
        I2 % coefficients I_02, I_12, I_22
    end

    methods
        function obj = TriKubBezPolynom(c,t,dlambda,dsigma)
            % initialiss TriKubBezPolynom
            % c ... values at the vertices
            % t ... vertices of the triangle
            % dlambda ... gradient w.r. to lambda at vertices
            % dsigma ... gradient w.r. to sigma at vertices
        end

        val = get.Ci(obj)
        %get method for Ci

        obj = CalcInitCoeff(obj,c,dlambda,dsigma)
        % calculates coefficients at the edge of the triangle
        % INPUT like above

        obj = CalcInitInteriorCoeff(obj)
        % calculates interior coefficient b_111
    end
    methods ( Static )
        val = CalcInitCm(c)
```

```
29          % calculates b_111 with Ci coefficients
30      end
31  end
```

7.7 class TriKubBezInterp

```
1   classdef TriKubBezInterp
2       properties
3           tkbp % TriKubBezPoly corresponding to dtri
4           dtri % a delaunay triangulation, DelaunayTri object
5       end
6
7       methods
8           function obj = TriKubBezInterp(v)
9               % initialises a TriKubBezInterp object
10              % v ... a Gridd object
11          end
12
13          obj = CalcBezPoly(obj,y,dlambda,dsigma,ff)
14          % starts the computation
15          % y ... function value
16          % dlambda ... gradient w.r. to lambda
17          % dsigma ... gradient w.r. to sigma
18          % ff ... funk object for evaluation of
19          % the considered function
20
21          obj = Farin(obj,e,tt)
22          % Farins method
23          % indices of vertices of an edge
24          % index of an edge of a triangle
25
26          obj = Farinend(obj);
27          % Last step of Farins method
28
29          obj = BorderCi(obj,e,t,ff);
30          % computes Bezier points at the boundary of the triangulation
31
32          [val, isin] = bezSurfVal(obj,x);
33          % evaluates the Bezier surface
34          % INPUT
35          % x ... point at which the surface is evaluated
36          % val ... resulting value
37          % isin ... boolean variable which says if x is inside of the
38          % triangulation
39      end
40  end
```

7.8 class BoxManagement

```
classdef BoxManagement
    properties
        initBox % (lambda, x) intval box. area which is considered
        lbox % corresponding lambda intval box
        sbox % corresponding sigma intval box

        verBoxes
        % containers.Map object. (lambda, sigma) intval boxes with
        % corresponding enclosure x intval box
        notVerBoxes
        % container.Map object. (lambda, sigma) intval boxes which are not
        % yet considered

        vNeighbourV
        % containers.Map object. Neighbourhood between verified Boxes
        vNeighbourN
        % containers.Map object. Neighbourhood between verified Boxes and
        % not verified Boxes
        nVNeighbourV
        % containers.Map object. Neighbourhood between not verified Boxes
        % and verified Boxes
        nVNeighbourN
        % containers.Map object. Neighbourhood between not verified Boxes

        Boundary % information which part of the boundary must be analysed
        Boundary2 % information where the path probably traverses
        BoundaryNo % information on which edge the boundary traverses
    end

    methods
        function obj = BoxManagement(box, sbo)
            % initialise a BoxManagement object
            % box: initialises initbox
            % sbo initilialises sbox
        end

        plotall(obj,list,farbe)
              % plotting method
              % list: list of restricting boxes
              % farbe: colour of the restricting boxes

        is = isNeighbour(obj, mod1, index1, mod2, index2)
        % Check if two boxes are neighboured
        % mod1, mod2: type of box, verified or not verified box
        % index1, index2: index of the boxes according to mod1,mod2

        [obj, nkeys] = modify(obj, mod, oldindex, list)
```

```
48          % method which updates all informations if something changes
49          % mod: which kind of change
50          % oldindex: index of the box in which a change happens
51          % list: lists the new boxes
52      end
53  end
```

7.9 class VerifiedBox

```
1   classdef VerifiedBox
2
3       properties
4           initbox %(lambda u) intval box to consider
5           BoxM % corresponding BoxManagement object for adminstration
6           func % funk object, function which is considered, extension
7           BezSurface % Bezier surface to evaluate approximate roots
8           flist
9           % list of indices according to boxes where the verification failed
10          blist
11          % list of indices according to boxes to restrict the enclosure
12      end
13
14      properties ( Hidden )
15          sweights % weight vector according to the domain
16          zweights % weight vector according to the co-domain
17
18          alpha % current bound alpha
19          invA % current preconditioning matrix inv(A)
20          A % current jacobian matrix
21
22          currlbox % current lambda box which is considered
23          currsbox % current sigma box which is considered
24
25          todolist % list of indices according which shall be verified
26          nlist
27          % list of indices according to boxes where a new alpha ís needed
28
29          stolnewbox % tolerances according to the size of a box
30          stolnewboxl
31          tolnewbox
32          tolnewboxl
33          inittolnewbox
34          inittolnewboxl
35          stboxs
36          stboxl
37          mintolnewbox
38          mintolnewboxl
```

```
39
40          minbsradius % tolerances according to the boundary check
41          minblradius
42          minbradius
43
44          actkey % key of the actually considered box
45          nactkey
46          % key of the box for which a new alpha has been computed
47      end
48
49      methods
50          function obj = VerifiedBox(mod, funkt, start, boxa)
51              % initialises a VerifiedBox object
52              % mod: modus, first or normal
53              % start: initial approximate root
54              % boxa: (u lambda) intval box which has to be analysed
55
56          end
57
58          obj = try2Verify(obj, mod1, mod2, start)
59          % Box verification algorithm
60          % mod1: mode first or normal
61          % mod2: weighted or not weighted norms
62          % start: approximate root if mod1 = first
63
64          obj = boundaryCheck(obj, mod)
65          % Analysing the boundary
66          % mod: weighted or not weighted
67
68          obj = calcAlpha(obj, mod1, mod2, start)
69          % Calculate a new alpha
70          % mod1: with corrector iteration or without
71          % mod2: weighted or not weighted norms
72          % start: approximate root
73
74          [val, succ] = corrStep(obj, start, invA);
75          % Corrector iteration
76          % start: approximate root
77          % invA: approximate inverse, Newton-Chord
78          % OUTPUT
79          % val: value of the root after iteration
80          % succ: successful or not
81
82          obj = findWeights(obj, start, AA, invA);
83          % simple method to get weights for the domain and co-domain
84          % start: approximate root
85          % AA: jacobian of F
86          % invA: approximate inverse
87
```

```
88          list = bisect(obj, box);
89          % bisection of the intval object box
90          % list: intval vector with the resulting boxes
91      end
92      methods (Static)
93          list = takeComplement(AA, BB);
94          % take the complement of a box AA in a box BB
95          % list: resulting boxes in a intval vector
96      end
97  end
```

7.10 class Verifying

```
1   classdef Verifying
2       properties
3           initStart % initial approximate root
4           initbox % initial box to consider
5
6           toDoList % keys according to boxes to analyse
7           verList % list of VerifiedBox objects which are analysed
8
9           func % funk object, function to consider
10          A12 % A1, A2 is here stored if it does not change
11      end
12      properties ( Hidden )
13          radbox % about the size of new chosen boxes
14          extbox
15
16          keylist % list of keys to boxes for analysation
17
18          boundarylist % stores edges where no solution path traverses
19      end
20
21      methods
22          function obj = Verifying(funkt, start, boxa, A2)
23              % initialises a Verifying object
24              % funkt: funk object -> func
25              % boxa: (lambda x) box to consider
26              % A2: if unfolding functional given and linear then A2
27          end
28
29          plot(obj)
30          % plotting method, uses plot method of BoxManagement
31
32          nbox = thecheck(obj,nbox,obox, i)
33          % checks if a new box for verification is appropriately chosen
34          % nbox: new intval box
```

```
35          % obox: old intval box at which the new box connects
36          % i: says at which edge the boxes connect
37
38      end
39      methods ( Static )
40          list = takeComplement(A,B)
41          % complementation method
42      end
43  end
```

References

[1] E.L. Allgower and K. Georg. *Numerical continuation methods*. Springer, Berlin, 1990.

[2] E. Baumann. Optimal centered forms. *BIT*, 28(1):80–87, 1988.

[3] S.N. Chow and J.K. Hale. *Methods of bifurcation theory*. Springer, New York, 1982.

[4] R.W. Clough and J.L. Tocher. Finite element stiffness matrices for analysis of plates in bending. In *Proceedings of conference on matrix methods in structural analysis*, pages 515–545, 1965.

[5] T. Csendes and D. Ratz. Subdivision direction selection in interval methods for global optimization. *SIAM Journal on Numerical Analysis*, 34(3):922–938, 1997.

[6] G. Farin. Bezier polynomials over triangles and the construction of piecewise C^r polynomials. *Brunel University Mathematics Technical Papers collection*, 1980.

[7] G. Farin. Smooth interpolation to scattered 3D data. *Surfaces in Computer-Aided Geometric Design*, pages 43–63, 1983.

[8] G. Farin. A modified Clough-Tocher interpolant. *Computer Aided Geometric Design*, 2(1-3):19–27, 1985.

[9] G. Farin. Triangular Bernstein-Bezier patches. *Computer Aided Geometric Design*, 3(2):83–127, 1986.

[10] M. Golubitsky, D.G. Schaeffer, and I. Stewart. *Singularities and groups in bifurcation theory. Volume I*. Springer, New York, 1985.

[11] P. Kashyap. Improving Clough-Tocher interpolants. *Computer aided geometric design*, 13(7):629–651, 1996.

[12] R.B. Kearfott. *Rigorous global search: continuous problems*. Kluwer, Dordrecht, 1996.

[13] R.B. Kearfott, M.T. Nakao, A. Neumaier, S.M. Rump, S.P. Shary, and P. Van Hentenryck. Standardized notation in interval analysis. In *Proc. XIII Baikal International School-seminar "Optimization methods and their applications"*, pages 106–113, 2005.

[14] R. Klein. *Algorithmische Geometrie*. Addison-Wesley-Longman, Bonn, 1997.

[15] S. Mann. Cubic precision Clough-Tocher interpolation. *Computer aided geometric design*, 16(2):85–88, 1999.

[16] R.E. Moore, R.B. Kearfott, and M.J. Cloud. *Introduction to interval analysis*. SIAM, Philadelphia, 2009.

[17] A. Neumaier. The enclosure of solutions of parameter-dependent systems of equations. In *Reliability in computing: the role of interval methods in scientific computing*, pages 269–285. Academic Press, 1988.

[18] A. Neumaier. *Interval methods for systems of equations*, volume 37. Cambridge University Press, Cambridge, 1990.

[19] A. Neumaier. Generalized Lyapunov-Schmidt reduction for parametrized equations at near singular points. *Linear Algebra and its Applications*, 324(1):119–131, 2001.

[20] H. Prautzsch, W. Boehm, and M. Paluszny. *Bezier and B-spline techniques*. Springer, Berlin, 2002.

[21] D. Ratz and T. Csendes. On the selection of subdivision directions in interval branch-and-bound methods for global optimization. *Journal of Global Optimization*, 7(2):183–207, 1995.

[22] W.C. Rheinboldt. *Numerical analysis of parametrized nonlinear equations*. Wiley-Interscience, New York, 1986.

[23] S.M. Rump. INTLAB - INTerval LABoratory. In Tibor Csendes, editor, *Developments in Reliable Computing*, pages 77–104. Kluwer Academic Publishers, Dordrecht, 1999. `http://www.ti3.tu-harburg.de/rump/`.

[24] SM Rump. Interval computations with INTLAB. *Brazilian Electronic Journal on Mathematics of Computation*, 1, 1999.

[25] R. Seydel. *Practical bifurcation and stability analysis*. Springer, New York, 2010.

Abstract (Deutsch)

In dieser Arbeit werden nichtlineare, reelle, parameter-abhängige Funktionen betrachtet. Hier sind die Nullstellen von solchen Funktionen gesucht. An nicht-singulären Stellen der Lösungsmannigfaltigkeit des Problems, welches wir betrachten, impliziert der Satz über implizite Funktionen, dass eine Funktion existiert, welche die Lösungsmannigfaltigkeit lokal in Abhängigkeit des gewählten Parameters beschreibt. Im Allgemeinen sind nun singuläre Stellen der Lösungsmannigfaltigkeit schwerer behandelbar. Unsere Methode erweitert das Problem, so dass Singularitäten in der Lösung des ursprünglichen Problems im erweiterten Problem korrigiert werden und so nicht mehr vorkommen. Des weiteren bekommen wir ein niedrig - dimensionales Problem, dessen Lösung die Lösungsmannigfaltigkeit des ursprünglichen Problems reflektiert, im Speziellen in der Nähe von Bifurkationspunkten. Die theoretische Grundlage für diese Methode wurde in Neumaier [19] erarbeitet. Wir wollen hier eine rigorose Einschließung der Lösungsmannigfaltigkeit finden. Für diesen Zweck wird das Konzept der Intervallanalysis und Intervallarithmetik in einer Sektion behandelt. Intervallanalysis und Intervallarithmetik wird hier hauptsächlich verwendet um Einschließungen von Funktionsbereichen zu finden um die erarbeitete Methode rigoros anzuwenden. Die Einführung enthält die wichtigsten Begriffe, Notationen und Sätze aus dem Bereich der Intervallanalysis.
Die Methode benötigt auch eine Interpolation der Lösungsmannigfaltigkeit. Für diesen Zweck werden Bezier Patches über Dreiecken betrachtet. Genauer gesagt wird zur Interpolation der Lösung eine Methode verwendet, die erstmals in Clough and Tocher [4] präsentiert wurde und dann von Farin in [8], [9] noch weiter entwickelt wurde.
Der Hauptteil der Arbeit präsentiert die zwei Theoreme, die der Methode zugrunde liegen, und den Schlüssen die daraus zu ziehen sind. Es wird diskutiert wie das urpsprüngliche Problem erweitert werden muss um die Singularitäten zu korrigieren. Auch eine Prediktor-Corrector Methode für die betrachtete Form des Problems wird erarbeitet. Weiters werden Algorithmen erarbeitet um rigorose Lösungseinschließungen zu bekommen. Der Hauptteil schließt mit einer Diskussion über Existenz und Eindeutigkeit von Lösungszweigen innerhalb von Lösungseinschließungen.
Die Implementation der erarbeiteten Methode erfolgt in MATLAB. Speziell verwendet wird die Toolbox INTLAB von Rump. Mehr dazu in Rump [23]. Am Ende der Arbeit werden noch Beispiele präsentiert, auf die die Implementation angewendet wurde.

Lebenslauf

Name:	Simon Konzett
Geburtsdatum, -ort:	2.11.1984 in Feldkirch, Vorarlberg
Staatsangehörigkeit:	Österreich
Familienstand:	ledig

Schulische Ausbildung:

Sept. 1991 - Juli 1995	Volksschule Satteins
Sept. 1995 - Juni 2003	Bundesrealgymnasium Feldkirch - Rebberggasse, mit Abschluss Matura

Studium:

Okt. 2005 - Nov. 2011	Diplomstudium Mathematik an der Universität Wien, "Angewandte Mathematik und Scientific Computing"

Zivildienst:

Okt. 2003 - Sept. 2004	Sozialzentrum Satteins

Praktikum:

Sept. 2010	Polytechnisches Institut Kiew (KPI)

Sprachen:

Deutsch (Muttersprache)
Englisch (Gut)
Französisch (Grundkenntnisse)
Russisch (Grundkenntnisse)