



TECHNISCHE
UNIVERSITÄT
WIEN

N U M E R I K P R O J E K T

Titel
ggf. mehrzeilig

ausgeführt am

Institut für
Analysis und Scientific Computing
TU Wien

unter der Anleitung von

Prof. Dr. Lothar Nannen

durch

Lukas Moser

Matrikelnummer: 1607333

Stefan Schrott

Matrikelnummer: 1607388

Wien, am 24. Januar 2018

Inhaltsverzeichnis

1 Grundlagen	1
2 Aufgabe a	2
2.1 Implementierung	2
2.2 Tests	3
3 Aufgabe b	4
3.1 Implementierung Version 1	4
3.2 Implementierung Version 2	4
3.3 Tests	6
4 Aufgabe c	7
4.1 Kreuzung der Kurve	7
5 Implementierung von adaptiver Schrittweite	7
5.1 Mögliche Strategien für Aufgabe a	9
5.1.1 Abschätzung der Schrittweite anhand der Krümmung von f	9
5.1.2 Schrittweitensteuerung durch Länge der Korrektorschrittes	10
5.2 Implementierungen	11
5.3 Tests	11
5.4 L_1 -Norm des Fehlers	12
6 Implementierung von Niveaulinien	14
6.1 Problemstellung und Idee der Implementierung	14
6.2 Details der Implementierung	16
6.3 Tests	17
7 Anhang: Code-Listings	18

1 Grundlagen

Die Grundlage für die folgenden Überlegung ist der Hauptsatz über implizite Funktionen im Spezialfall von Funktionen $F : A \times B \rightarrow \mathbb{R}$, wobei A und B der Einfachheit halber offene Intervalle seien.

Satz 1: Seien $a < b$ sowie $c < d \in \mathbb{R}$ und $F : (a, b) \times (c, d) \rightarrow \mathbb{R}$ stetig differenzierbar. Seien $x_0 \in (a, b)$ und $y_0 \in (c, d)$, sodass $F(x_0, y_0) = 0$ und $\frac{\partial F}{\partial y}(x_0, y_0) \neq 0$.

Dann existieren $a_0, b_0 \in \mathbb{R}$ mit $a < a_0 < x_0 < b_0 < b$ und eine stetig differenzierbare Funktion $f : (a_0, b_0) \rightarrow \mathbb{R}$ mit $f(x_0) = y_0$, sodass

$$\forall x \in (a_0, b_0) : F(x, f(x)) = 0$$

und

$$\forall x \in (a_0, b_0) : f'(x) = -\frac{\frac{\partial F}{\partial x}(x, f(x))}{\frac{\partial F}{\partial y}(x, f(x))}. \quad (1)$$

Beweis: Unter den gegebenen Voraussetzungen ist der Hauptsatz über implizite Funktionen anwendbar und liefert Umgebungen U von x_0 und V von y_0 und eine Funktion $f : U \rightarrow V$ mit den geforderten Eigenschaften. Da x_0 ein innere Punkt von U ist, enthält U ein Intervall (a_0, b_0) mit den geforderten Eigenschaften.

Die Umgebung $V \subseteq \mathbb{R}$ in der Zielmenge von f kann durch ganz \mathbb{R} ersetzt werden, da wir nur behauptet haben, dass $y = f(x)$ eine Lösung von $F(x, \cdot) = 0$ ist, allerdings nicht dass diese eindeutig ist. ■

Satz 2: Sei unter den Voraussetzungen des vorherigen Satz F zwei mal stetig differenzierbar. Dann ist $f \in C^2((a_0, b_0))$ mit $f''(x) =$

$$\frac{-\frac{\partial^2 F}{\partial^2 x}(x, f(x)) \left(\frac{\partial F}{\partial y}(x, f(x))\right)^2 + 2\frac{\partial^2 F}{\partial x \partial y}(x, f(x)) \frac{\partial F}{\partial x}(x, f(x)) \frac{\partial F}{\partial y}(x, f(x)) - \frac{\partial^2 F}{\partial^2 y}(x, f(x)) \left(\frac{\partial F}{\partial x}(x, f(x))\right)^2}{\left(\frac{\partial F}{\partial y}(x, f(x))\right)^3}.$$

Außerdem gilt:

$$\forall x \in (a_0, b_0) \exists \xi \in (x_0, x) \cup (x, x_0) : f(x) = y_0 + \frac{\frac{\partial F}{\partial x}(x_0, y_0)}{\frac{\partial F}{\partial y}(x_0, y_0)}(x - x_0) + \frac{f''(\xi)}{2}(x - x_0)^2.$$

Beweis: Aus $F \in C^2$ folgt mit der Kettenregel und Einsetzen der Darstellung (1) für f' :

$$\begin{aligned} \frac{d}{dx} \left(\frac{\partial F}{\partial x}(x, f(x)) \right) &= \left(\frac{\partial^2 F}{\partial^2 x}(x, f(x)), \frac{\partial^2 F}{\partial x \partial y}(x, f(x)) \right) \cdot \begin{pmatrix} 1 \\ f'(x) \end{pmatrix} \\ &= \frac{\partial^2 F}{\partial^2 x}(x, f(x)) - \frac{\partial^2 F}{\partial x \partial y}(x, f(x)) \frac{\frac{\partial F}{\partial x}(x, f(x))}{\frac{\partial F}{\partial y}(x, f(x))}. \end{aligned}$$

Für $\frac{d}{dx} \left(\frac{\partial F}{\partial y}(x, f(x)) \right)$ erhält man analog eine ähnliche Darstellung. Damit kann man den Ausdruck (1) mithilfe der Quotientenregel differenzieren und erhält durch Erweitern mit $\frac{\partial F}{\partial y}(x, f(x))$ obige Darstellung für f'' .

Die zweite Aussage folgt aus dem Satz von Taylor und der Tatsache, dass f'' als Komposition stetiger Funktionen stetig ist. ■

2 Aufgabe a

Das Ziel dieser Aufgabenstellung ist es den Graphen, der durch die Nullstellenmenge einer Funktion, numerisch anzunähern. Im ersten Schritt wollen wir dazu einem Gitter $x_j = x_0 + j * h, j \in \{1 \dots n\}$ entlang x folgen. Dazu soll die Funktion F auf der gesamten betrachteten Menge die Bedingungen des Hauptsatzes über implizite Funktionen erfüllen und $F(x_0, y_0) = 0$ sein. Damit wissen wir, dass $\forall j \in \{1 \dots n\} : F(x_j, f(x_j)) = 0$ gilt. Um $y_n = f(x_n)$ zu berechnen betrachte man mittels Mittelwertsatz

$$f(x_{n+1}) = f(x_n) + f'(x_n) * h + r_n, |r_n| \leq \sup_{a, b \in [x_n, x_{n+1}]} |f'(a) - f'(b)| * h.$$

Demnach ist für hinreichend kleine Schrittweite das Restglied r_n klein genug, so dass die Nullstelle für $F(x_{n+1}, \cdot)$ und somit $f(x_{n+1})$ mittels Newtonverfahren gefunden werden kann.

2.1 Implementierung

Algorithm 1: Kurve A

Input : F ...Funktion

(x_0, y_0) ...Startpunkt

h ...Schrittweite

n ...Schrittzahl

Output: $(x_j, y_j)_{j \in [1, n]}$

1 **for** $i = 1$ **to** n **do**

2 $x_i = x_{i-1} + h$

3 $df = -\frac{\frac{\partial F}{\partial x}(x_{i-1}, y_{i-1})}{\frac{\partial F}{\partial y}(x_{i-1}, y_{i-1})}$

4 $\tilde{y}_i = y_{i-1} + h * df$

5 $G(a) = F(x_i, a)$

6 $y_i = \text{Newton}(G, \tilde{y}_i)$

7 **end**

Die Implementierung ist eine einfache Umsetzung von Algorithmus 1. Fehlschlägen können dabei folgende Punkte. $\frac{\partial F}{\partial y}(x_{i-1}, y_{i-1})$ kann 0 sein. Das wurde im Vorhinein ausgeschlossen. Das Newton verfahren kann keine Nullstelle finden bzw findet eine falsche Nullstelle. Das kann behoben werden indem man die Schrittweite reduziert.

2.2 Tests

Wie zu erwarten war, ist die Darstellung an Stellen in denen $f'(x)$ groß wird sehr ungenau, vor allem bei großen Schrittweiten. Siehe Abbildung 1.

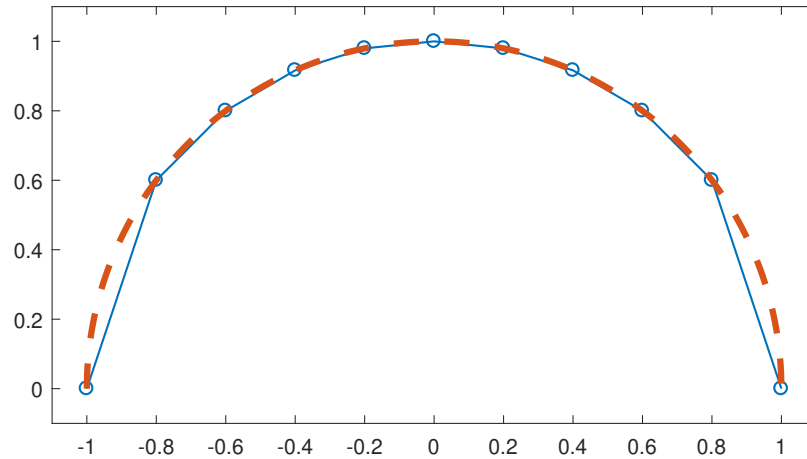


Abbildung 1: Plot zweier Halbkreise mit Schrittweiten 10^{-3} und 0.2

Wenn die darzustellende Funktion für ein x mehr als ein y hat so dass $F(x, y) = 0$ so kann es passieren, dass das Newtonverfahren gegen die falsche Nullstelle konvergiert. Ob die gefundene Nullstelle die Korrekte ist oder nicht, ist nur unter großem Aufwand feststellbar. Als Beispiel für so ein Funktion und zur Untersuchung des Verhaltens des Algorithmus verwenden wir

$$G(x, y) = \sin(10 * \pi * (\sin(x) - y))$$

mit Nullstellen

$$\{(x, y) \in \mathbb{R} : G(x, y) = 0\} = \{(z, \sin(z) + n * 0.1) : z \in \mathbb{R}, n \in \mathbb{N}\}$$

Wie Abbildung 3 zu entnehmen ist beginnt der Algorithmus ab einer Schrittweite $\geq \frac{\pi}{8}$ falsche Nullstellen zu finden.

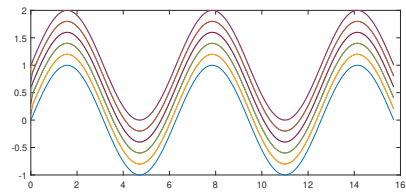


Abbildung 2: Illustration der Nullstellenmenge von G .

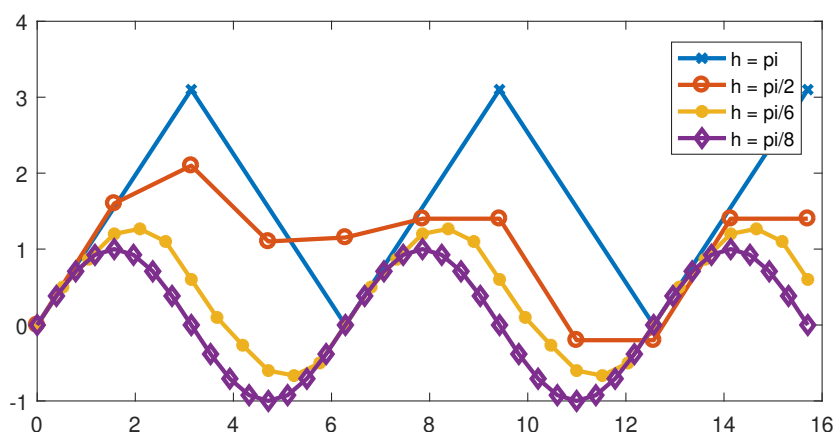


Abbildung 3: G , $x_0 = y_0 = 0$ dargestellt mit verschiedenen Schrittweiten.

3 Aufgabe b

Die Voraussetzung des Satzes über implizite Funktionen, dass die Ableitung nach y nicht 0 werden darf ist sehr restriktiv, da so nicht einmal ein Kreis gezeichnet werden kann. Es ist möglich diese Voraussetzung abzuschwächen indem man fordert, dass in den Nullstellen entweder die Ableitung nach x oder y ungleich 0 sein müssen. Dadurch kann die Nullstellenmenge der Funktion durch offene Mengen überdeckt werden für die jeweils $F(x, f(x)) = 0$ oder $F(g(y), y) = 0$ gilt.

3.1 Implementierung Version 1

Die einfachste Variante auf Aufgabe a basierend Aufgabe b zu implementieren ist rekursiv den Algorithmus aufzurufen, wenn die Betragsmäßige Ableitung in eine Richtung größer wird als die andere. Wichtig dabei ist auf folgendes zu achten, dass der Wechsel Richtung früh genug erfolgt, damit der Bereich mit Nullstellen nicht in eine Richtung verlassen wird und dass der Richtungswechsel mit dem korrekten Vorzeichen passiert. Nachteile dieser Implementierung liegen darin, dass es aufwändig ist vorherzusagen wie lang die Kurve wird. In Algorithmus 1 läuft die Kurve von $\{x_0\} \times \mathbb{R}$ bis $\{x_0 + h * n\} \times \mathbb{R}$. Das bedeutet für $m \in \mathbb{N}$, dass man einen Viertelkreis mit $n = m$ und $h = \frac{1}{m}$ zeichnen kann. Mit Algorithmus 2 benötigt man dafür $n = \lceil m * \sqrt{2} \rceil$ und $h = \frac{1}{m}$, da der Richtungswechsel bei $x_i \approx \sin(\frac{\pi}{4})$ stattfindet.

3.2 Implementierung Version 2

Um die Probleme mit Version 1 zu beheben, vor allem, wenn beide Ableitungen sehr klein sind ist es möglich nicht nur entlang der Richtungen der kanonischen Basisvektoren zu arbeiten. Stattdessen ist es möglich eine Orthonormalbasis mit Basistransformation t_ϕ zu finden, so dass

Algorithm 2: Kurve B1

```

Input  :  $F$ ...Funktion
           $(x_0, y_0)$ ...Startpunkt
           $h$ ...Schrittweite
           $n$ ...Schrittzahl

Output:  $(x_j, y_j)_{j \in [1, n]}$ 

1 for  $i = 1$  to  $n$  do
2    $dx = \frac{\partial F}{\partial x}(x_{i-1}, y_{i-1})$ 
3    $dy = \frac{\partial F}{\partial y}(x_{i-1}, y_{i-1})$ 
4   if  $abs(dy) < abs(dx)$  then
5      $G(a, b) = F(b, a)$ 
6      $dir = 1$  if  $abs(dy) > TOL$  then
7        $dir = \text{sgn}(dy)$ 
8     else if  $i > 1$  then
9        $dir = \text{sgn}(y_{i-1} - y_{i-2})$ 
10    end
11     $(x_k, y_k)_{i..n} = \text{KurveB1}(G, (y_{i-1}, x_{i-1}), abs(h) * dir, n-i)$ 
12    return
13  else
14     $x_i = x_{i-1} + h$ 
15     $\tilde{y}_i = y_{i-1} + h * df$ 
16     $G(a) = F(x_i, a)$ 
17     $y_i = \text{Newton}(G, \tilde{y}_i)$ 
18  end
19 end

```

für $G(x, y) = F(t_\phi(x, y))$ die Ableitung der impliziten Funktion $g'(x) = 0$ ist. Dies ist einfach zu verifizieren, da

$$t_\phi(x, y) = (x * \cos(\phi) - y * \sin(\phi), x * \sin(\phi) + y * \cos(\phi)) \quad (2)$$

$$g'(x) = -\frac{\frac{\partial G}{\partial x}(x, y)}{\frac{\partial G}{\partial y}(x, y)} \quad (3)$$

$$\frac{\partial G}{\partial x}(x, y) = \frac{\partial F}{\partial x}(t_\phi(x, y)) * \cos(\phi) + \frac{\partial F}{\partial y}(t_\phi(x, y)) * \sin(\phi) = c(x, y) * \sin(\phi + \psi(x, y)) \quad (4)$$

$$\frac{\partial G}{\partial y}(x, y) = c(x, y) * \cos(\phi + \psi(x, y)) \quad (5)$$

Dadurch ist nun $h * n$ annähernd die Kurvenlänge und die Genauigkeit des Prediktorschritts ist unabhängig von der ersten Ableitung von F . Da die $\frac{\partial G}{\partial x}(x, y)$ für ϕ und $\phi + \pi$ gleich ist und damit keine eindeutige Richtung vorgegeben ist, müssen andere Punkte dafür herangezogen werden. Mögliche Ansätze sind u.a. positives Skalarprodukt zwischen aktuellem und letzten Richtungsvektor oder minimale Differenz zwischen aktuellem und letzten ϕ .

Algorithm 3: Kurve B2

Input : F ...Funktion
 (x_0, y_0) ...Startpunkt
 h ...Schrittweite
 n ...Schrittzahl

Output: $(x_j, y_j)_{j \in [1, n]}$

```

1 for  $i = 1$  to  $n$  do
2    $dx = \frac{\partial F}{\partial x}(x_{i-1}, y_{i-1})$ 
3    $dy = \frac{\partial F}{\partial y}(x_{i-1}, y_{i-1})$ 
4   if  $dx == 0$  then
5      $x_i = x_{i-1} + h$ 
6      $\tilde{y}_i = y_{i-1}$ 
7      $G(a) = F(x_i, a)$ 
8      $y_i = \text{Newton}(G, \tilde{y}_i)$ 
9   else
10     $\psi = \text{atan}(dy/dx)$  //Richtung maximaler Ableitung
11     $\phi = \psi - \frac{\pi}{2}$  //Richtung minimaler Ableitung
12     $\phi = \text{correctPhi}(\phi)$ 
13     $\psi = \phi + \frac{\pi}{2}$  //psi mitkorrigieren
14     $\tilde{x}_i = x_{i-1} + h * \cos(\phi)$ 
15     $\tilde{y}_i = y_{i-1} + h * \sin(\phi)$ 
16     $G(a) = F(\tilde{x}_i + a * \cos(\psi), \tilde{y}_i + a * \sin(\psi))$ 
17     $s = \text{Newton}(G, 0)$ 
18     $x_i = \tilde{x}_i + s * \cos(\psi)$ 
19     $y_i = \tilde{y}_i + s * \sin(\psi)$ 
20  end
21 end

```

3.3 Tests

Abbildungen 5 und 4 stellt die notwendigen Korrekturen nach dem Prediktor für beiden Algorithmen 2 und 3. Man erkennt dabei, dass der Fehler an Stellen mit großer erster und zweiter Ableitung groß ist. Die Laufzeiten beider Algorithmen wurden nicht gemessen, sind aber in beiden Fällen für ≤ 10000 Iterationen nicht bemerkbar. Ein Fehlerbild, das bei zu großen Schrittweiten und vor allem bei Algorithmus 2 aufgetreten ist das verlassen des Bereichs in dem die Nullstellen liegen. zB dass für $F(x, y) = x^2 + y^2 - 1$ ein $x_n > 1$ wurde. Das Problem kann durch eine Sinnvolle Wahl der Schrittweite verhindert werden.

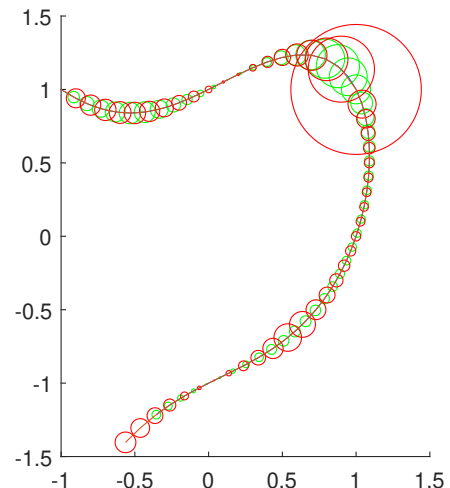


Abbildung 4: Wie Abb. 5 mit $x^3 - xy + x^2 - 1$

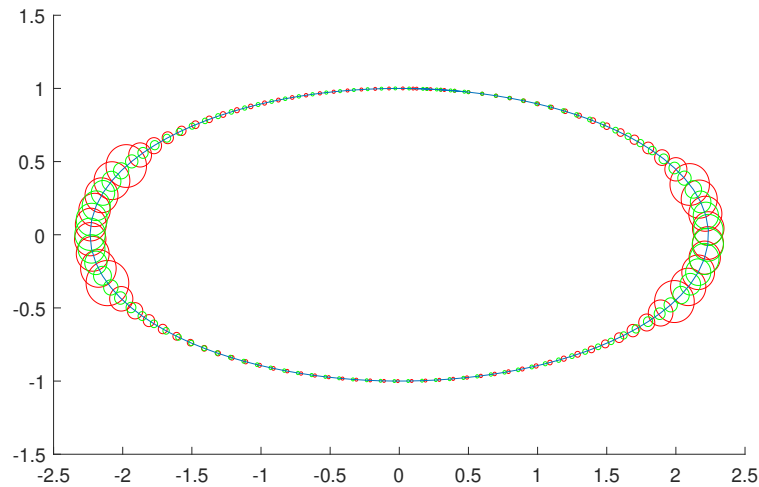


Abbildung 5: Vergleich der Korrekturschrittgrößen zwischen V1 und V2 als 10-facher Radius der roten bzw grünen Kreise für die Funktion $F(x, y) = 0.2x^2 + y^2 - 1$

4 Aufgabe c

Wenn alle Richtungsableitungen $\frac{\partial F}{\partial v} = 0$ sind gibt es keine eindeutige Richtung in die die Kurve fortgesetzt werden soll. Das kann u.a. an Stellen passieren an denen sich die Kurve kreuzt oder die Kurve abrupte Richtungswechsel vollzieht. Algorithmen um an solchen Stellen fortzufahren sind heuristische begründet und der Erfolg hängt jeweils von der zu plottenden Funktion ab.

4.1 Kreuzung der Kurve

Während der Entwicklung waren solche Stellen kein Problem, da üblicherweise nur eine sehr kleine Umgebung des Punktes betroffen ist, die auch ohne gezielte Wahl der Schrittweite nicht getroffen wird und damit einfach übersprungen wird. Siehe dazu Abbildung 6.3. Um die Kreuzung von Kurven gezielt zu behandeln wäre eine Strategie vom letzten Punkt aus die Schrittweite (mehr zu der Umsetzung in Sektion 5) zu variieren. Eine mögliche Implementierung wäre Algorithmus 4 der zuerst versucht mit reduzierter Schrittweite näher an die Nullstelle zu kommen um sie dann mit normaler oder erhöhter Schrittweite zu überspringen.

5 Implementierung von adaptiver Schrittweite

Das Ziel von adaptiver Schrittweite ist es im Idealfall, dass der Abstand des Polygonzugs durch die Punkte (x_i, y_i) zur Nullstellenmenge kleiner ist als eine vorgegebene Konstante und dafür möglichst wenige Punkte bzw. möglichst wenig Rechenzeit benötigt werden.

Für den die adaptive Schrittweite ergeben sich zwei Problemstellungen:

Algorithm 4: Kurve Kreuzung

Input : F ...Funktion
 $(x, y)_n$...bisher berechnet Punkte
 h ...Schrittweite
 i ...Schrittzahl aktuell

Output: (x_i, y_i)

```

1  $(a, b) = (x_i, y_i)$  for  $k = 1$  to  $m$  do
2    $(\tilde{x}, \tilde{y}) = \text{Prediktor}(F, (x_{i-1}, y_{i-1}), k \cdot h/m)$ 
3    $(\tilde{x}, \tilde{y}) = \text{Korrektor}(F, (\tilde{x}, \tilde{y}))$ 
4    $dx = \frac{\partial F}{\partial x}(\tilde{x}, \tilde{y})$ 
5    $dy = \frac{\partial F}{\partial y}(\tilde{x}, \tilde{y})$ 
6   if  $\text{abs}(dx) > TOL$  and  $\text{abs}(dy) > TOL$  then
7      $(a, b) = (\tilde{x}, \tilde{y})$ 
8   else
9     break
10  end
11 end
12 for  $k = 1$  to  $m$  do
13    $(\tilde{x}, \tilde{y}) = \text{Prediktor}(F, (a, b), h + k \cdot h/m)$ 
14    $(\tilde{x}, \tilde{y}) = \text{Korrektor}(F, (\tilde{x}, \tilde{y}))$ 
15    $dx = \frac{\partial F}{\partial x}(\tilde{x}, \tilde{y})$ 
16    $dy = \frac{\partial F}{\partial y}(\tilde{x}, \tilde{y})$ 
17   if  $\text{abs}(dx) > TOL$  and  $\text{abs}(dy) > TOL$  then
18      $(x_i, y_i) = (\tilde{x}, \tilde{y})$  return
19   end
20 end
21 fail
```

1. Die Datenpunkte (x_i, y_i) sollen möglichst sein.
2. Der Polygonzug soll die Nullstellenmenge möglichst gut approximieren.

Die erste Problemstellung ist recht einfach zu lösen: Durch das Newton-Verfahren im Korrektor-Schritt ist (x_i, y_i) extrem an einer Nullstelle – die dabei bleibende geringe Abweichung wird auch nicht Schrittweite des Prediktors beeinflusst. Das einzige was hier zu tun ist, ist also die Schrittweite zu verkleinern, falls das Newton-Verfahren nicht konvergiert.

Die zweite Problemstellung ist die wesentlich interessanter, die im folgenden zu lösen versucht wird. Der Einfachheit halber werden mögliche Strategien für adaptive Schrittweite zuerst an der Implementierung aus Aufgabe a getestet, da das Koordinatensystem dort noch fest ist. Dann werden sie, falls möglich und sinnvoll, auf den allgemeinen Fall ausgeweitet.

Wir gehen bei den Überlegungen dazu überdies davon aus, dass die Datenpunkte selbst korrekt sind bzw. dass wir deren Fehler vernachlässigen können.

5.1 Mögliche Strategien für Aufgabe a

Wir nehmen nun an, dass es auf (a, b) eine Funktion f gibt, sodass $F(x, f(x)) = 0$ für alle $x \in (a, b)$.

Es ergeben sich folgende mögliche Strategien:

1. Die Krümmung von f im letzten Punkt explizit berechnen
2. Versuchen, die Krümmung von f aus den letzten Punkten zu schätzen
3. Die Differenz von Prediktor und Korrektor betrachten

5.1.1 Abschätzung der Schrittweite anhand der Krümmung von f

Zusätzlich dazu nehmen wir an, dass $F \in C^2((a, b))$, was nach Satz ... eine hinreichende Bedingung für $f \in C^2((a, b))$ ist. Außerdem liefert der Satz dann eine explizite Darstellung von f'' .

Sei $x_0 \in (a, b)$ ein Startwert und $c > 0$ eine vorgegebene Toleranz. Das Ziel ist es nun, ein (möglichst großes) $h \in (0, h_{max})$ zu finden, sodass die Gerade zwischen $(x_0, f(x_0))$ und $(x_0 + h, f(x_0 + h))$ maximal um c von f abweicht. Hat man so ein h gefunden, kann man $x_1 := x_0 + h$ setzen und von dort aus fortfahren, um insgesamt einen Polygonzug zu erhalten, für den diese Abschätzung gilt.

Bei einem einzelnen Iterationsschritt handelt es sich dabei um eine lineare Interpolationsaufgabe, da wir nach dem Korrektorschritt davon ausgehen können, dass $f(x_0)$ und $f(x_0 + h)$ nahezu korrekt ist. Es geht hier also um die Frage, wie sich f zwischen diesen Punkten verhält, insbesondere wie sehr es von einer Geraden durch die Punkte $(x_0, f(x_0))$ und $(x_0 + h, f(x_0 + h))$ abweicht.

Die Gleichung der Interpolationsgeraden g_h ist gegeben durch (vgl Skriptum Bsp 3.17):

$$g_h(x) = \frac{f(x_0)(x_0 + h - x) + f(x_0 + h)(x - x_0)}{h},$$

wobei dann folgende Abschätzung gilt:

$$\sup_{x \in [x_0, x_0 + h]} |f(x) - g_h(x)| \leq \frac{h^2}{8} \sup_{x \in [x_0, x_0 + h]} |f''(x)| \leq \frac{h^2}{8} \sup_{x \in [x_0, x_0 + h_{max}]} |f''(x)|.$$

Es gilt:

$$c = \frac{h^2}{8} \sup_{x \in [x_0, x_0 + h_0]} |f''(x)| \iff h = \sqrt{\frac{8c}{\sup_{x \in [x_0, x_0 + h_{max}]} |f''(x)|}}.$$

Daher gilt für $h \in (0, h_{max})$:

$$h \leq \sqrt{\frac{8c}{\sup_{x \in [x_0, x_0 + h_{max}]} |f''(x)|}} \implies \sup_{x \in [x_0, x_0 + h]} |f(x) - g_h(x)| \leq c. \quad (6)$$

Da das numerische Berechnen des Supremums nicht möglich ist, kann man an dieser Stelle nicht mehr so weiterarbeiten, dass obige Abschätzungen garantiert werden. In einem ersten Schritt kann man heuristisch

$$h := \sqrt{\frac{8c}{|f''(x_0)| + \frac{8c}{h_{max}^2}}} \in (0, h_{max})$$

wählen. Dann sind zwar keine der obigen Abschätzungen garantiert, man hat einen ersten Ansatz für adaptive Schrittweite. Zusätzlich dazu wird bei der Implementierung auch eine Mindestschrittweite vorgegeben, die nicht unterschritten werden darf.

5.1.2 Schrittweitensteuerung durch Länge der Korrektorschrittes

Der Prediktor tut nichts anderes als eine Tangente $t_i(x)$ entlang von f durch den Punkt $(x_i, f(x_i))$ zu legen und anhand dessen einen Punkt $(x_{i+1}, \tilde{y}_{i+1})$ zu berechnen, der dann durch einen Korrektorschritt zum Punkt $(x_{i+1}, f(x_{i+1}))$ korrigiert wird. Bezeichne $d := |f(x_{i+1}) - \tilde{y}_{i+1}|$ die Länge des Korrektorschrittes.

Ein Vorteil dieses Verfahrens ist, dass es sehr wenig Aufwand hat und weder $F \in C^2$ gefordert werden muss noch Ableitungen zweiter Ordnung benötigt werden.

Ein Nachteil ist, dass man erst nach dem Prediktor- und Korrektorschritt d erhält und dann "falls d zu groß war", Prediktor- und Korrektorschritt wiederholen muss. Das zweite Problem, ist dass vorerst nicht klar, ist aber welcher Schranke für d man den Iterationsschritt wiederholen soll und um wie viel kleiner die Schrittweite dann sein soll.

Um die intuitiv vernünftige Tatsache, dass ein großer Korrektorschritt ein Anzeichen dafür ist, dass f nicht gut approximiert wird, auch formal plausibel zu machen und ein Gefühl dafür zu bekommen, welche d "zu groß" sind, kann man sich folgendes überlegen:

Wir wissen, dass $(f - t_i)(x_i) = 0$, $(f - t_i)'(x_i) = 0$ und $(f - t_i)'' = f''$. Daraus erhält man mit dem Satz von Taylor im Entwicklungspunkt x_i :

$$\exists \xi \in (x_i, x_{i+1}) : (f - t_i)(x_{i+1}) = \frac{f''(\xi)}{2} (x_{i+1} - x_i)^2 = \frac{f''(\xi)}{2} h^2.$$

Wegen $\frac{f''(\xi)}{2} (x_{i+1} - x_i)^2$ kann man daraus folgern, dass

$$\exists \xi \in (x_i, x_{i+1}) : h = \sqrt{\frac{2d}{|f''(\xi)|}}.$$

War nun $d > 4c$ folgt daraus sofort, dass die Ungleichung auf der linken Seite der Implikation in Formel (6) verletzt ist. Man kann daraus natürlich nicht schließen, dass daher die Abschätzung auf der rechten Seite dieser Implikation verletzt sein muss, aber es macht dennoch plausibel, dass h ungünstig groß gewesen ist.

Man sieht daraus auch, dass es die Toleranz für d in der Größenordnung von c wählen sollte und der Faktor, um dem man die Schrittweite gegebenenfalls verringert, proportional zu $\sqrt{d/c}$ sein sollte. Bei der Implementierung wird man zusätzlich fordern, dass der Faktor jedenfalls größer 2 sein muss, um nicht mehrmals hintereinander mit sehr ähnlichen Schrittweiten zu arbeiten. Außerdem wird man auch hier eine Mindestschrittweite festlegen.

5.2 Implementierungen

Da bei adaptiver Schrittweite für den Nutzer im Vorhinein nicht mehr absehbar ist, wie weit die Nullstellenmenge bei gegebener Schrittzahl gezeichnet ist, wird nun die Länge des Polygonzuges als Abbruchkriterium verwendet.

Die Funktionen haben also jetzt folgende Form:

```
[ x, y, z, steps ] =  
    implicitCurveAdapt( F, dFx, dFy, d2Fxx, d2Fxy, d2Fyy, x0, y0, length,  
        maxStepWidth, minStepWidth, c )
```

Dabei sind die Eingabe-Parameter:

- Die ersten sechs sind F bzw deren Ableitungen (werden die Ableitungen 2. Ordnung nicht gebraucht, entfallen sie)
- x_0 und y_0 sind die Startwerte
- `length` ist die Ziel-Länge des Polygonzuges
- `maxStepWidth` und `minStepWidth` geben der Bereich für die Schrittweite an
- c ist die Konstante c aus Kapitel 5.1.

Die Ausgabe-Daten sind:

- x und y sind die Vektoren der Eckpunkte des Polygonzuges.
- z dient zur Ausgabe eines der Parameter für die adaptive Schrittweite (zu Testzwecken)
- `steps` ist die Anzahl der benötigten Schritte.

Es sind folgende Funktionen implementiert worden:

1. Adaptive Schrittweite nach Kapitel 5.1.1 sowie Halbierung der Schrittweite, falls das Newton-Verfahren nicht konvergiert,
2. Adaptive Schrittweite nach Kapitel 5.1.2 mit Reduktion der Schrittweite um den Faktor $\sqrt{d/c}$, falls $d > c$ sowie Halbierung der Schrittweite, falls das Newton-Verfahren nicht konvergiert,
3. Eine Kombination der Kriterien aus den beiden vorherigen Kriterien, wobei das zweite erst bei $d > 5c$ greift.

Ein Listing des Codes der 4. Version befindet sich im Anhang.

5.3 Tests

Die Strategien wurden für die Funktion $F(x, y) := \sin(x^2) - y$ getestet. Die Nullstellenmenge von F ist also genau der Graph von $f(x) = \sin(x^2)$. Dabei war Parameter `length` jeweils 200, die Schrittweite durfte zwischen 1 und 10^{-10} variieren. Der Parameter c wurde so gewählt, dass die verschiedenen Strategie jeweils möglichst gleich viele Schritte benötigten, um die Ergebnisse sinnvoll vergleichen zu können.

In Abbildung 6 sieht man, dass die gemischte Strategie besser ist als ein alleiniges Betrachten der Krümmung im vorherigen Punkt. Beim ersten Schritt ist startet man im Punkt $(0, f(0))$, wo $f''(x) = (\sin(x^2))'' = 2 \cdot \cos(x^2) - 4 \cdot x^2 \cdot \sin(x^2)$ den Wert 2 hat, also im Vergleich zu den Werten von f'' klein ist. Daher wird eine zu große Schrittweite verwendet. Bei $x \approx 2.521$ hat f'' ungefähr den Wert 0.160, was zu einer so großen Schrittweite führt, dass ein Teil der Graphen übersprungen wird. Bei der gemischten Strategie war der Korrektorschritt dort aber dann so groß, dass die Schrittweite verkleinert wurde und es zu keinen Problemen kam. Später wird das Verhalten dieses Verfahren besser, weil die Funktionswerte von f'' tendenziell größer werden.

In Abbildung 7 sieht man, dass die gemischte Strategie dem alleinigen Betrachten der Länge des Korrektorschrittes deutlich überlegen ist. Insbesondere wenn die Schwingung von $\sin(x^2)$ schon sehr groß ist, kann es passieren, dass auch bei sehr großer Schrittweite der Korrektorschritt nur sehr klein ist, weil "zufällig" fast eine Nullstelle von F getroffen wurde. Dabei können wie man beispielsweise bei $x \approx 11$ und $x \approx 13$ sieht, sogar mehrere Schwingungen von f einfach "übersehen" werden.

Auch bei deutlich kleinerem c (zB um dem Faktor 10 kleiner) liegen die Schrittzahlen in der Größenordnung von 400. Dort liefern die Abschätzung der Krümmung sowie die gemischte Strategie noch relativ brauchbare Ergebnisse. Die Schrittweitensteuerung anhand des Korrektorschrittes liefert dann hingegen unbrauchbare Ergebnisse.

Die Bereich für die mögliche Schrittweite war mit $[1, 10^{-10}]$ für diesen Test natürlich sehr groß gewählt. Tatsächlich wurden nie Schrittweiten kleiner $5 \cdot 10^{-3}$ verwendet. Eine kleinere maximale Schrittweite hätte auch die Probleme der Strategie mit der Länge der Korrektorschrittes verringert. Völlig beseitigt hätte es aber die Probleme nie, da die Schwingung von $\sin(x^2)$ immer schneller werden.

5.4 L_1 -Norm des Fehlers

Im Vorherigen wurde nur qualitativ darauf geachtet, ob der Algorithmus grobe Fehler macht, aber in keiner Form quantitativ verglichen, welcher Polygonzug näher an f ist, also zB wie groß die Fläche zwischen f und dem Polygonzug ist.

Da wir uns bisher darauf einschränken, dass die Nullstellenmenge von F im betrachten Bereich durch ein $f(x)$ auflösbar ist, folgt dass sowohl f als auch der Polygonzug p wohldefinierte Funktionen von x sind und die Fläche genau die L_1 -Norm der Funktion $f - p$ ist. Diese ist aber im Allgemeinen nicht einfach zu berechnen, weil man nicht weiß, wo $f - p$ Vorzeichenwechsel hat.

Schränkt man sich auf den Fall ein, dass $f(x)$ konkav ist, folgt sofort $p(x) \leq f(x)$, also $|f(x) - p(x)| = f(x) - p(x)$.

Es seien also $x_1 < \dots < x_n$ die x -Werte der bekannten Datenpunkte und p der Polygonzug durch die Punkte $(x_i, f(x_i))_{i=1, \dots, n}$. Dann gilt für die L_1 -Norm von $f - p$ auf $[x_1, x_n]$:

$$\begin{aligned}\|f - p\|_1 &= \int_{x_1}^{x_n} f(x) - p(x) dx = \int_{x_1}^{x_n} f(x) dx - \sum_{i=1}^{n-1} \int_{x_i}^{x_{i+1}} p(x) dx \\ &= \int_{x_1}^{x_n} f(x) dx - \sum_{i=1}^{n-1} \frac{f(x_i) + f(x_{i+1})}{2} (x_{i+1} - x_i)\end{aligned}$$

Die letzte Gleichheit folgt daraus, dass p auf den Intervallen $[x_i, x_{i+1}]$ ein Polynom ersten Grades ist und daher von der Trapezregel exakt integriert wird.

Sei $F(x, y) = x^{10} + y^{10} - 1$. Dann kann $F(x, y) = 0$ auf $(-1, 1)$ für $y > 0$ durch die Funktion $f(x) = \sqrt[10]{1 - x^{10}}$ aufgelöst werden. Wegen $f''(x) < 0$ sieht man leicht, dass f konkav ist.

Das Integral $\int_{x_1}^{x_n} f(x) dx$ wurde mithilfe symbolischer Integration durch Matlab und anschließender Auswertung der Stammfunktion berechnet. Die symbolische Integration mit Matlab war notwendig, da die Stammfunktion nicht durch elementare Funktionen darstellbar ist (man benötigt die hypergeometrische Funktion).

Der Grund, warum hier F so gewählt wurde (und nicht zB als $x^2 + y^2 - 1$, bei der die Stammfunktion von f deutlich einfacher wäre) ist, dass es ein sehr einfaches Beispiel ist, wo f konkav ist und eine sehr unterschiedliche Krümmung hat.

Es wurden dabei folgende vier Methoden verglichen¹:

1. konstante Schrittweite
2. adaptive Schrittweite anhand der Krümmung im Punkt x_i
3. adaptive Schrittweite anhand der Länge des Korrektorschrittes
4. adaptive Schrittweite anhand des gemischten Kriteriums

Die Eingabeparameter wurden dabei jeweils so gewählt, dass die Schrittzahl jeweils (bis auf eine Abweichung $\leq 2\%$) gleich ist.

Man sieht, dass die Fehler bei allen Methoden mit der Größenordnung $\mathcal{O}(h^2)$ gegen Null gehen. Für äquidistante Stützstellen ist das nicht überraschend, da dies genau der Konvergenzaussage der summierten Trapezregel entspricht (siehe Numerik-Skriptum Bsp 4.9). Die Methoden mit adaptiver Schrittweite haben also alle die selbe Konvergenzgeschwindigkeit, sind allerdings alle um einen Faktor in der Größenordnung 10 besser.

Zwischen den einzelnen Strategien ist kein sehr großer Unterschied, wobei die gemischte Strategie etwas besser ist gefolgt von der Betrachtung der Länge des Korrektorschrittes.

¹Das sind genau die selben Algorithmen wie vorher, sie werden hier nur noch einmal aufgezählt, um klarzustellen, welche Spalte der Tabelle sich auf welche Methode bezieht.

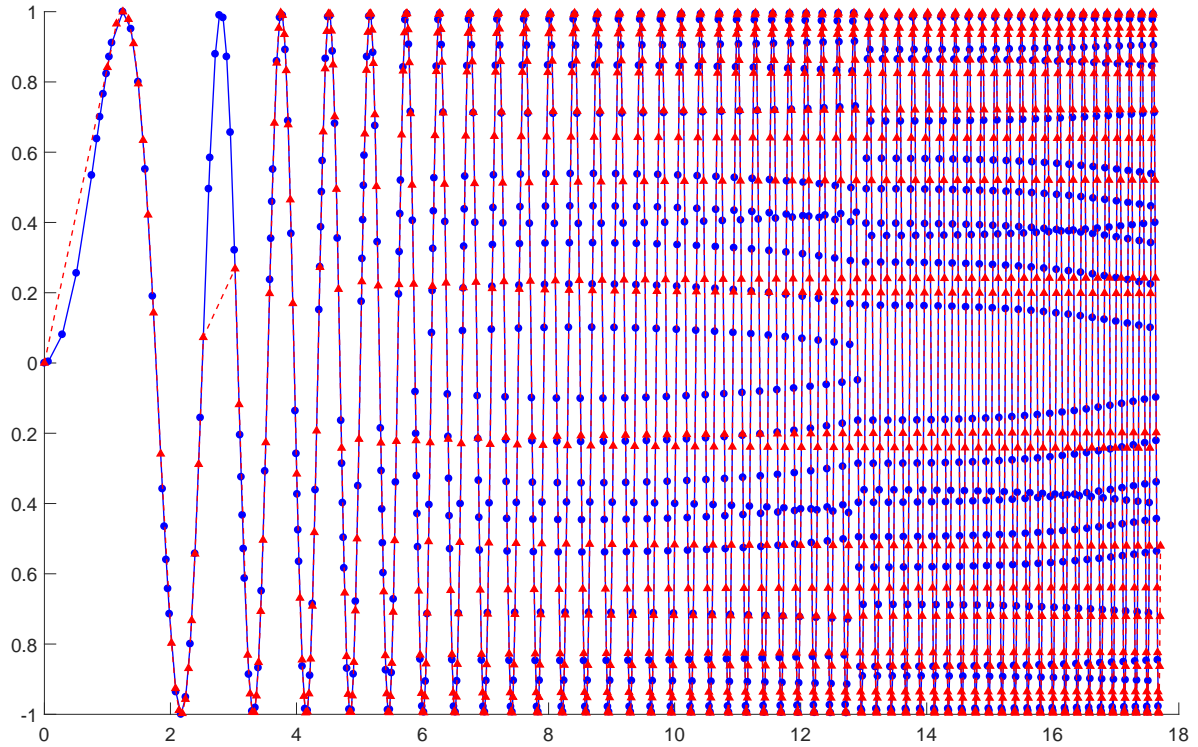


Abbildung 6: Vergleich der Strategie aus Kapitel 5.1.1 mit $c = 6 \cdot 10^{-3}$ (rot, 1087 Schritte) mit der gemischten Strategie mit $c = 1.3 \cdot 10^{-2}$ (blau, 1097 Schritte)

6 Implementierung von Niveaulinien

6.1 Problemstellung und Idee der Implementierung

Die bisherigen Algorithmen finden Paare $(x_i, y_i)_{i=1, \dots, N}$, sodass für $F(x_i, y_i) = 0$ für $i = 1, \dots, N$ und stellen damit die Nullstellenmenge von F (oder nur einen Teil davon) näherungsweise graphisch dar.

Im Folgenden sind $c_1, \dots, c_k \in \mathbb{R}$ gegeben und es sollen für $j = 1, \dots, k$ die Teilmengen von $\{(x, y) \in \mathbb{R}^2 : F(x, y) = c_j\}$ graphisch dargestellt werden.

Grundsätzlich ist dieses Problem einfach auf die vorherigen Algorithmen zurückzuführen, indem man die Nullstellenmengen der Funktionen $F_j(x, y) := F(x, y) - c_j$ graphisch darstellt.

Bei den vorherigen Algorithmen musste ein Startwert $(x_0, y_0) \in \mathbb{R}^2$ übergeben werden, für den gilt $F(x_0, y_0) = 0$, also müsste in diesem Fall k Startwerte $(x_j, y_j) \in \mathbb{R}^2$ übergeben werden, sodass

$$F(x_j, y_j) = c_j \quad j = 1, \dots, k.$$

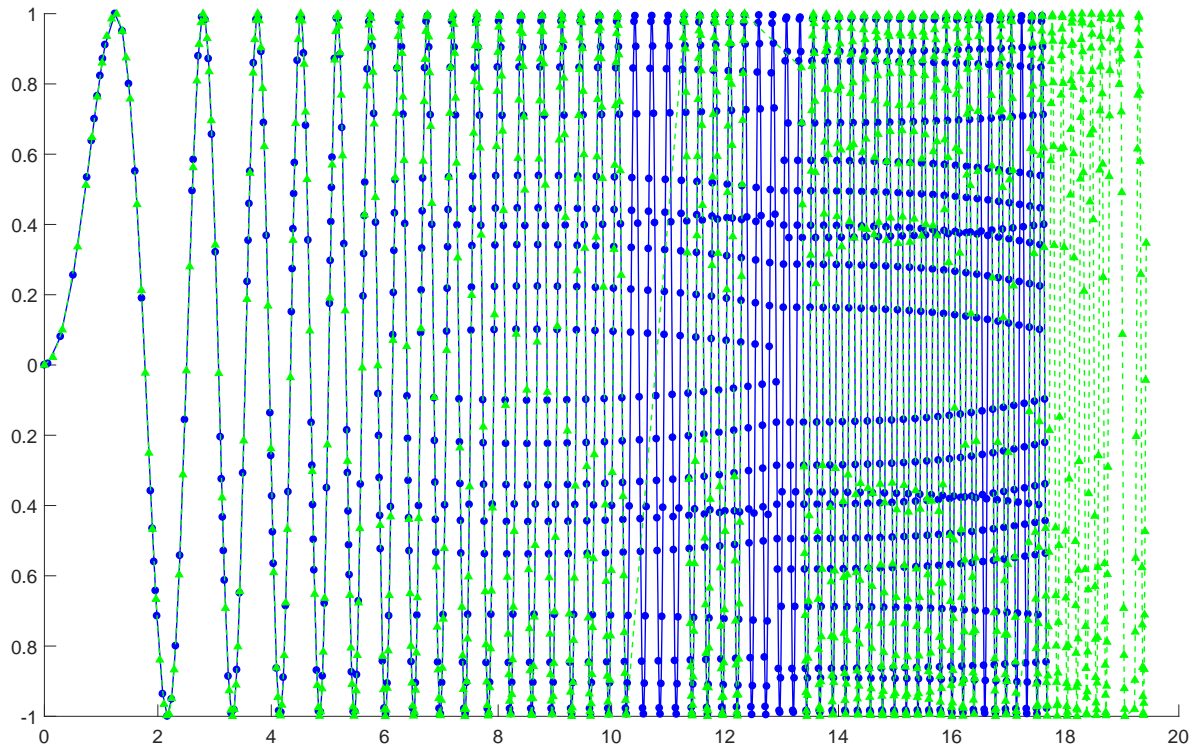


Abbildung 7: Vergleich der Strategie aus Kapitel 5.1.2 mit $c = 7.4 \cdot 10^{-2}$ (grün, 1074 Schritte) mit der gemischten Strategie mit $c = 1.3 \cdot 10^{-2}$ (blau, 1097 Schritte)

Dies stellt sich in der Praxis als sehr benutzerunfreundlich heraus, da die Gleichungen $F(x_j, y_j) = c_j$ im Allgemeinen nicht einfach zu lösen sind.

Aus diesem Grund wurde ein Algorithmus implementiert, der in einem gegebenen Intervall $[a, b] \times [c, d] \subseteq \mathbb{R}^2$ entsprechende (x_j, y_j) sucht und anschließend für $j = 1, \dots, k$ einen der vorherigen Algorithmen mit der Funktion F_j und den Startwerten (x_j, y_j) aufruft.

Der wesentliche Schritt ist also, nach Möglichkeit Nullstellen von F_j in $[a, b] \times [c, d]$ zu finden. Das Newton-Verfahren im \mathbb{R}^n steht hier nicht zur Verfügung, da nur es für Funktionen $G : \mathbb{R}^n \rightarrow \mathbb{R}^n$ anwendbar ist. Wegen der Regularitätsforderung an die Jacobi-Matrix von G , ist es auch nicht möglich etwa $G(x, y) := \begin{pmatrix} F(x, y) \\ 0 \end{pmatrix}$ oder $G(x, y) := \begin{pmatrix} F(x, y) \\ F(x, y) \end{pmatrix}$ zu setzen und damit das Newton-Verfahren zu verwenden.

Es wird daher folgende Strategie verwendet:

- Sei $m := \begin{pmatrix} m_x \\ m_y \end{pmatrix} := \begin{pmatrix} (a+b)/2 \\ (c+d)/2 \end{pmatrix}$. Berechne $F(m)$. Falls $F(m) = 0$ sind wir fertig, falls $F(m) < 0$ betrachte $-F$. Wir können also im folgenden annehmen $F(m) > 0$.
- Werte mithilfe geeigneter Schleifen F an verschiedenen $(x, y) \in [a, b] \times [c, d]$ aus, bis

(x, y) mit $F(x, y) \leq 0$ gefunden wird. Tritt dies nicht ein, bricht der Algorithmus an der Stelle ohne Ergebnis ab. Ist $F(x, y) = 0$ sind wir fertig. Wir können also im Folgenden annehmen, dass $F(x, y) < 0$ ist.

- Sei nun $\Psi : [0, 1] \rightarrow \mathbb{R}^2 : t \mapsto \begin{pmatrix} m_x \\ m_y \end{pmatrix} + t \begin{pmatrix} x - m_x \\ y - m_y \end{pmatrix}$. Dann ist $G := \Psi \circ F : [0, 1] \rightarrow \mathbb{R}$ stetig mit $G(0) > 0$ und $G(1) < 0$. Mithilfe des Bisektionsverfahrens kann man eine Nullstelle t_0 von G finden.
- Dann ist $\Psi(t_0) \in [a, b] \times [c, d]$ eine Nullstelle von F .

Diese Strategie hat in unseren Tests immer die Nullstellen gefunden. Nullstellen die gleichzeitig Extremstellen der Funktion F sind, können damit nur durch großen Zufall gefunden werden, da die Funktion bei ihnen keinen Vorzeichenwechsel macht. Das ist kein großer Mangel, da diese Nullstellen aber uninteressant sind, denn dort ist $\frac{\partial F}{\partial x} = 0$ und $\frac{\partial F}{\partial y} = 0$, was sie als Startwerte eher unbrauchbar macht.

6.2 Details der Implementierung

Es wurde also eine Funktionen der Art

`nivlines (F, dFx, dFy, Z, A, B, C, D, Steps, StepWidth)`

implementiert. Dabei sind:

- Z ein Vektor ist, der die Funktionswerte enthält, zu denen Niveaulinien geplottet werden sollen. Bezeichne k im Folgenden die Länge von Z).
- A, B, C, D jeweils Vektoren der Länge k , sodass ein Startwert für die Niveaulinie zu $Z(j)$ im Intervall $[A(j), B(j)] \times [C(j), D(j)]$ gesucht wird. Alternativ können auch Skalare übergeben werden, die wie Vektoren mit konstanten Einträgen behandelt werden.
- **Steps** und **StepWidth** sind ebenfalls Vektoren der Länge k oder Skalare, die die Schrittzahl bzw. Schrittweite übergeben.

Die Implementierung der Funktion sieht dann im Wesentlichen (Assertions etc. wurden im Listing weggelassen) so aus:

```

1 function [ X ,Y ] =nivlines4 (F, dFx, dFy, Z, A, B, C, D, Steps,
   StepWidth)
2
3 X = cell(k,1);
4 Y = cell(k,1);
5
6 X0=zeros(1,k);
7 Y0=zeros(1,k);
8
9 for j = 1:k
10     X{j}=zeros(Steps(j)+1,1);
11     Y{j}=zeros(Steps(j)+1,1);

```

```

12     Fj = @(x,y)F(x,y) - Z(j);
13     [X0(j),Y0(j),err]=findZero(Fj,A(j),B(j),C(j),D(j));
14
15     if err ~= 0 % kein Startwert gefunden
16         X{j}=zeros(0); %leerer Vektor, damit nichts geplottet
            wird
17         Y{j}=zeros(0);
18     else
19         [X{j},Y{j}] = implicitCurveXXX( Fj, dFx, dFy, X0(j), Y0(
            j), Steps(j), StepWidth(j) );
20     end
21 end
22 end

```

Listing 1: Ich bin ein Beispiel-Lisitng

Da die Anzahl der Datenpunkte für unterschiedliche Niveaulinien sehr unterschiedlich sein kann (zB wegen unterschiedlicher Länge oder Krümmung) und man die jeweiligen Anzahl je nach Algorithmus im Vorhinein auch nicht weiß, ist es nicht sinnvoll, alle Vektoren der x - bzw. y -Werte der Punkte für die einzelnen Niveau-Linien in gemeinsame Matrizen der Art $X, Y \in \mathbb{R}^{k \times \maxSteps}$ zu schreiben.

der Punkte für die einzelnen Niveaulinien in Matrix $X \in \mathbb{R}^{k \times \maxSteps}$ zu schreiben. Stattdessen bietet sich ein cell-Arrays an, der k Vektoren der Länge $Steps$ enthält. Der Zugriff auf die einzelnen Vektoren erfolgt durch $X\{j\}$.

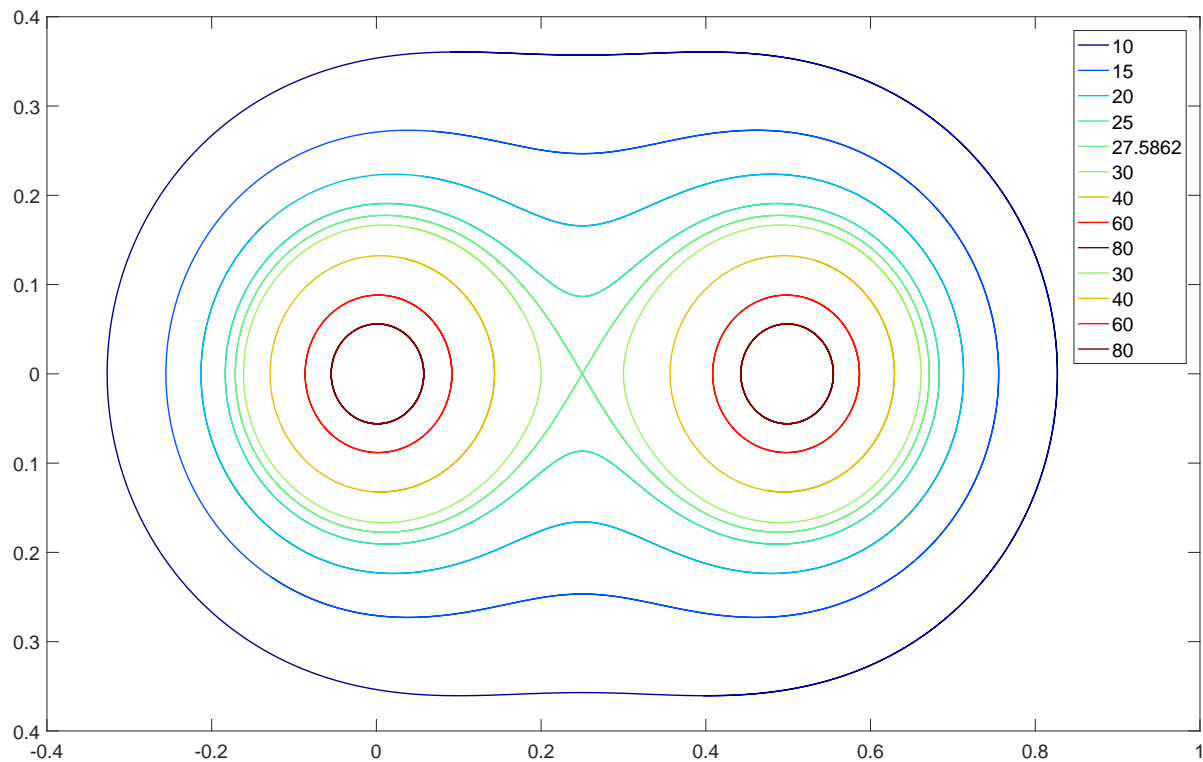
6.3 Tests

Sei

$$F(x, y) := \frac{1}{x^2 + y^2 + 10^{-2}} + \frac{1}{(x - 0.5)^2 + y^2 + 10^{-2}}$$

Sei $Z := (10, 15, 20, 25, 800/29, 30, 40, 60, 80, 30, 40, 60, 80)$ der Vektor der Funktionswerte, für die Niveau-Linien geplottet werden sollen. Für alle Werte wurden Startpunkte im Intervall $[0, 1/4] \times [0, 1]$ gesucht, für jene Werte, die im Vektor Z doppelt vorkommen, wurde zusätzlich im Intervall $[1/2, 3/4] \times [0, 1]$ nach einem Startwert gesucht. Die Motivation für die Auswahl des Wertes $800/29$ ist, dass $F(1/4, 0) = 800/29$ und $DF(1/4, 0) = (0, 0)$.

Die Schrittweite betrug $2 \cdot 10^{-3}$ die Schrittzahl 2000 für die ersten fünf Niveaulinien bzw. 500 für die Restlichen.



7 Anhang: Code-Listings

Der gesamte Code und diese Ausarbeitung ist unter https://github.com/xicht/NumProj17-18_2 auffindbar.

```

1 function [x0,y0,err] = findZero (F, a, b, c, d)
2 % finde (x0,y0) in [a,b]x[c,d] mit F(x0,y0)=0
3
4 mx = (a+b)/2;
5 my = (c+d)/2;
6
7 if isZero(F(mx,my))
8     x0y0 = [mx,my];
9 else
10     if F(mx,my) > 0
11         x0y0 = findZero2 (F,a,b,c,d);
12     else
13         x0y0 = findZero2 (@(x,y)-F(x,y),a,b,c,d);

```

```

14     end
15 end
16 x0=x0y0(1);
17 y0=x0y0(2);
18
19 if isZero(F(x0,y0))
20     err=0;
21 else
22     err=1;
23 end
24 end
25
26 function [X0Y0,err] = findZero2 (F, a, b, c, d)
27 % finde (x0,y0) in [a,b]x[c,d] mit F(x0,y0)=0
28 % fuer den Spezialfall F(mx,my) > 0
29
30 mx = (a+b)/2;
31 my = (c+d)/2;
32
33
34 %finde Funktionswert kleiner null
35 [x0y0,err]=findNegVal(F,a,b,c,d,4,20);
36
37 if err==1
38     [x0y0,err]=findNegVal(F,a,b,c,d,4,99); %99 statt 100 um
39     andere Funktionswerte zu treffen
40 end
41 if err==1
42     X0Y0=[0,0];
43     return; %kein vorzeichen welchsel, also wird es nix
44 end
45
46
47 %transformiere auf Funktion F(Psi)=G: [0,1]-> R
48 Psi1= @(t) mx + t*(x0y0(1)-mx);
49 Psi2= @(t) my + t*(x0y0(2)-my);
50 G = @(t) F(Psi1(t),Psi2(t));
51
52
53 %finde Nullstelle von G in [0,1]
54 t0 = bisection(G,0,1);
55
56 %transfomiere Nullstelle in [0,1] zurueck auf NSt in R^2
57 X0Y0 = [mx+t0*(x0y0(1)-mx),my+t0*(x0y0(2)-my)];

```

```
58
59 end
60
61
62 function [x0y0, err] = findNegVal(F,a,b,c,d,k,n)
63 % n anzahl der einzelnen zerteilung
64 % k anzahl der intervallverkleinerungen
65
66 mx = (a+b)/2;
67 my = (c+d)/2;
68 err = 0;
69
70
71 for j=k:-1:1
72     [x0y0, err2] = findNegVal2(F,mx-(b-a)/2^j,mx+(b-a)/2^j,my-(d
73     -c)/2^j,my+(d-c)/2^j,n);
74     %suche_in = [[mx-(b-a)/2^j,mx+(b-a)/2^j],[my-(d-c)/2^j,my+(d
75     -c)/2^j]]
76     if err2==0
77         return;
78     end
79 end
80 %wenn man bis daher kommt wurde nix gefunden
81 warning('gar keine NSt gefunden');
82 err=1;
83 x0y0=[0,0];
84
85 end
86
87 function [x0y0,err]=findNegVal2(F,a,b,c,d,n)
88 % n gibt die Feinheit der Suche an: [a,b] resp [c,d] wird in ca
89 2n
90 %intervalle zerlegt
91
92 err=0;
93
94 mx = (a+b)/2;
95 my = (c+d)/2;
96
97 dx = (b-a)/(2*n);
98 dy = (d-c)/(2*n);
99
100 for j=-n:n
101     for k=-n:n
102         %[mx+j*dx,my+k*dy]
```

```
100         if F(mx+j*dx,my+k*dy) < 0
101             x0y0 = [mx+j*dx,my+k*dy];
102             return;
103         end
104     end
105 end
106
107 %wenn wir bis daher kommen waren wir erfolglos
108 x0y0=[0,0];
109 err=1;
110 %warning('jetzt keine NSt gefunden');
111
112 end
```

Listing 2: Implementierung der Nullstellensuche im \mathbb{R}^2