NUMERIKPROJEKT

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 23. Januar 2018

Inhaltsverzeichnis

1 Grundlagen

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

Satz 1: Seien $a < b$ sowie $c < d \in \mathbb{R}$ und $F : (a, b) \times (c, d) \to \mathbb{R}$ 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 $f : (a_0, b_0) \to \mathbb{R}$ mit $f(x_0) = y_0$, sodass

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

und

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

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 $f : U \to V$ 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 Vorraussetzungen 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 $x_j = x_0 + j * h, j \in \{1...n\}$ 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 $\forall j \in \{1...n\} : F(x_j, f(x_j) = 0)$ gilt. Um $y_n = f(x_n)$ zu berechnen betrachte man mittels Mittelwertsatz

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

Demnach ist für hinreichend kleine Schrittweite das Restglied r_n klein genug, so dass die Nullstelle für $F(x_{n+1}, .)$ 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: $(x_j, y_j)_{j \in [1,n]}$

for $i = 1$ *to* n **do**
  $x_i = x_{i-1} + h$
  $df = -\frac{\frac{\partial F}{\partial x}(x_{i-1}, y_{i-1})}{\frac{\partial F}{\partial y}(x_{i-1}, y_{i-1})}$
  $\tilde{y}_i = x_{i-1} + h * df$
  $G(a) = F(x_i, a)$
  $y_i = \text{Newton}(G, \tilde{y}_i)$
end

Die Implementierung ist eine einfache Umsetzung von Algorithmus 1. Fehlschlagen können dabei folgende 3 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. Wenn die darzustellende Funktion für ein

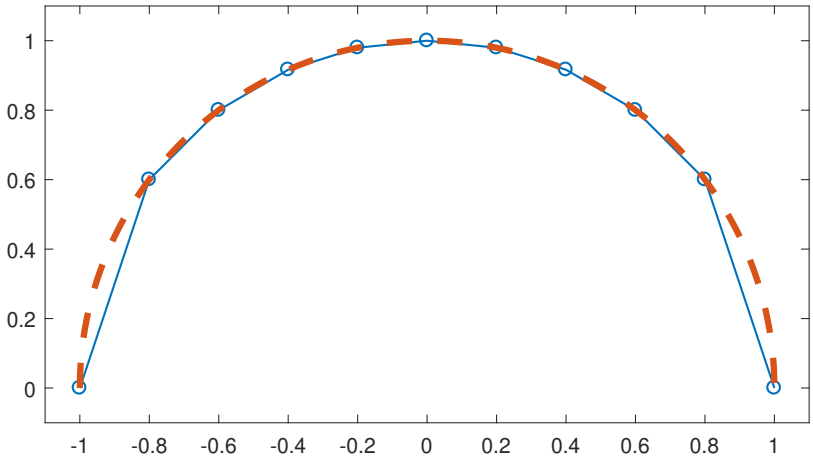


Abbildung 1: Plot zweier Halbkreise mit Schrittweiten 10^{-3} und 0.2

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.

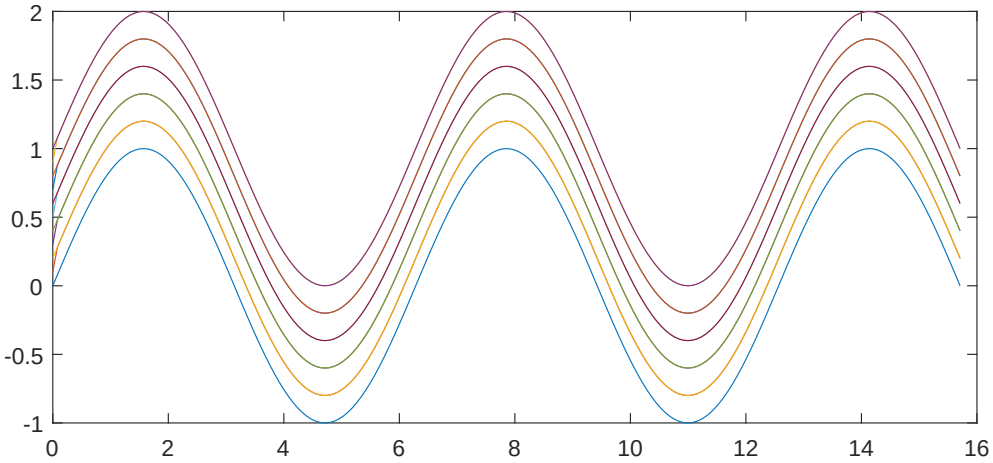


Abbildung 2: Illustration der Nullstellenmenge von G.

3 Implementierung von Aufgabe b Version 1

3.1 Tests

4 Implementierung von Aufgabe b Version 2

Um $y_n = f(x_n)$ zu berechnen betrachte man mittels Taylorformel.

$$f(x_{n+1}) = f(x_n) + f'(x_n) * h + r, |r| \leq \left| \sup_{x \in [x_n, x_{n+1}]} \frac{f''(x)}{2} \right| * h^2$$

4.1 Tests

5 Implementierung von adaptiver Schrittweite

Das Ziel von adaptiver Schrittweite ist es im Idealfall, dass der Abstand des Polygonzuges 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:

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

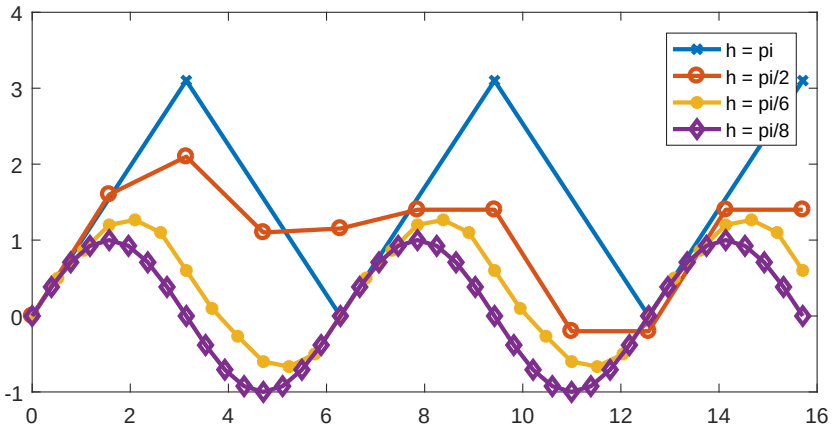


Abbildung 3: G, $x_0 = y_0 = 0$ dargestellt mit verschiedenen Schrittweiten.

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 einen 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:

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

Daher gilt für $h \in (0, h_{max})$:

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

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 Korrektschritt d erhält und dann "falls d zu groß war", Prediktor- und Korrektschritt 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 keiner 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 Entwickungspunkt 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 (2) 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 Schrittanzahl 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)
- `x0` und `y0` 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.

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

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

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 $j = 1, \ldots, k$ 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 $G : \mathbb{R}^n \to \mathbb{R}^n$ anwendbar ist. Wegen der Regularitätsforderung an die Jacobi-Matrix von G, ist es auch nicht möglich etwa $G(x,y) := \binom{F(x,y)}{0}$ oder $G(x,y) := \binom{F(x,y)}{F(x,y)}$ zu setzen und damit das Newton-Verfahren zu verwenden.

Es wird daher folgende Strategie verwendet:

- Sei $m := \binom{m_x}{m_y} := \binom{(a+b)/2}{(c+d)/2}$. 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 $\Psi : [0,1] \to \mathbb{R}^2 : t \mapsto \binom{m_x}{m_y} + t\binom{x-m_x}{y-m_y}$. Dann ist $G := \Psi \circ F : [0,1] \to \mathbb{R}$ 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 unterinteressant 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)]×[ 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 Schrittanzahl bzw. Schrittweite übergeben.

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

```
function [ X ,Y ] =nivlines4 (F, dFx, dFy, Z, A, B, C, D, Steps,
    StepWidth)

X = cell(k,1);
Y = cell(k,1);

X0=zeros(1,k);
Y0=zeros(1,k);

for j = 1:k
    X{j}=zeros(Steps(j)+1,1);
    Y{j}=zeros(Steps(j)+1,1);
    Fj = @(x,y)F(x,y) - Z(j);
    [X0(j),Y0(j),err]=findZero(Fj,A(j),B(j),C(j),D(j));

    if err ~= 0 % kein Startwert gefunden
        X{j}=zeros(0); %leerer Vektor, damit nichts geplotet
   wird
        Y{j}=zeros(0);
    else
        [X{j},Y{j}] = implicitCurveXXX( Fj, dFx, dFy, X0(j), Y0(
   j), Steps(j), StepWidth(j) );
    end
end
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-Arays an, der k Vektoren der Länge *Steps* enthält. Der Zugriff auf die einzelnen Vektoren erfolgt durch `X{j}`.

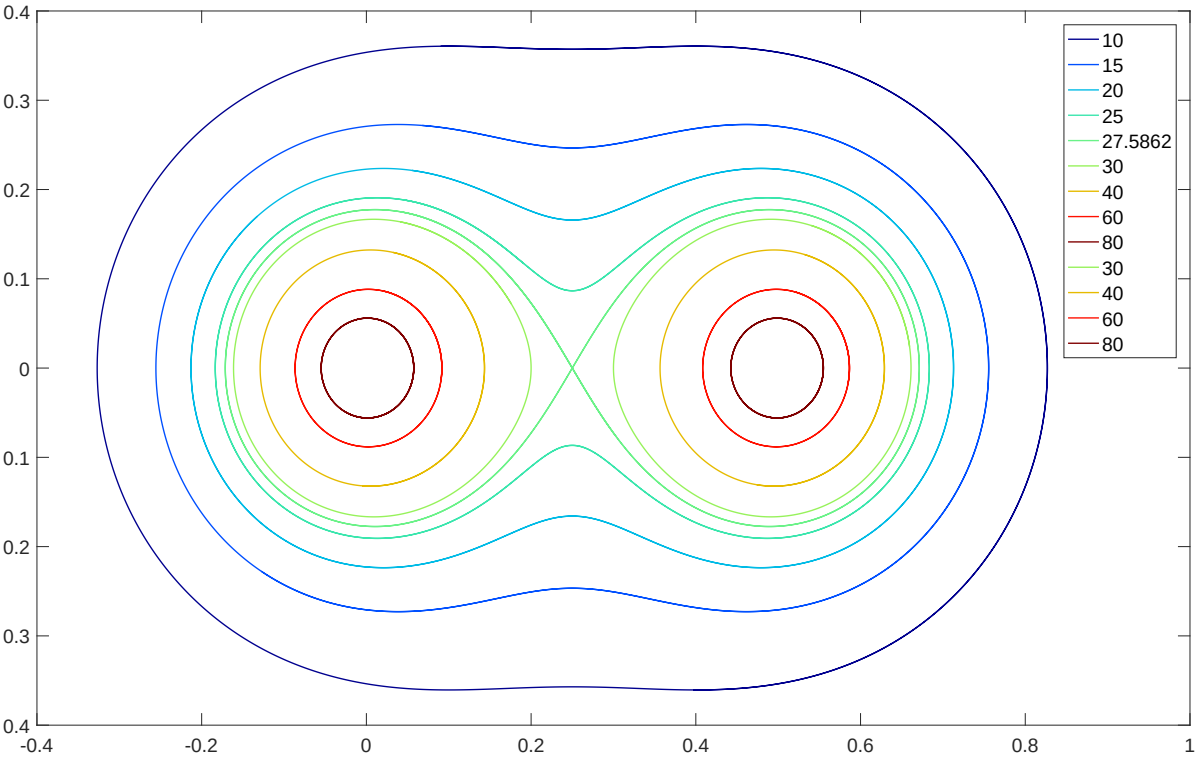
6.3 Tests

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 Schrittanzahl 2000 für die ersten fünf Niveaulinien bzw. 500 für die Restlichen.



7 Anhang: Code-Listings

```
function [x0,y0,err] = findZero (F, a, b, c, d)
% finde (x0,y0) in [a,b]x[c,d] mit F(x0,y0)=0

mx = (a+b)/2;
my = (c+d)/2;

if isZero(F(mx,my))
    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
      andere Funktionswerte zu treffen
39 end
40 if err==1
41     X0Y0=[0,0];
42     return; %kein vorzeichen welchsel, also wird es nix
43 end
44
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
   -c)/2^j,my+(d-c)/2^j,n);
73     %suche_in = [[mx-(b-a)/2^j,mx+(b-a)/2^j],[my-(d-c)/2^j,my+(d
   -c)/2^j]]
74     if err2==0
75         return;
76     end
77 end
78 %wenn man bis daher kommt wurde nix gefunden
79 warning('gar keine NSt gefunden');
80 err=1;
81 x0y0=[0,0];
82
83 end
84
85 function [x0y0,err]=findNegVal2(F,a,b,c,d,n)
86 % n gibt die Feinheit der Suche an: [a,b] resp [c,d] wird in ca
   2n
87 %intervalle zerlegt
88
89 err=0;
90
91 mx = (a+b)/2;
92 my = (c+d)/2;
93
94 dx = (b-a)/(2*n);
```

```
95  dy = (d-c)/(2*n);
96
97  for j=-n:n
98      for k=-n:n
99          %[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 $\mathbb{R}^2$