paper
a4 [t=2cm][b=2cm][f=1cm][h=1cm][r=2.5cm][l=2.5cm]

Iterative Lösungsverfahren

Kantonsschule am Burggraben St.Gallen

17. September 2015

Vorgelegt durch: Silvan Mosberger
Vorgelegt bei: Angelika Rupflin Signer
Externer Professor: Norbert Hungerb¸ühler

Inhaltsverzeichnis

1. Einleitung

1.1. Problemstellung

Die Schulmathematik handelt hauptsächlich von Gleichungen mit relativ kleinem Lösungsaufwand, wie zum Beispiel $x^2 = x + 1$, welche einfach nach x aufgelöst werden kann und man erhält eine exakte Lösung wie in diesem Fall $x_{1,2} = \frac{1 \pm \sqrt{5}}{2}$. Doch diese Art ist in der Praxis nur selten anzutreffen, viel öfters kommen um einiges kompliziertere Gleichungen auf, für die man keine exakte Lösung finden kann. Als Beispiel die Gleichung $2^x + 3^x = 10$, welche auch mit aller Mühe nicht nach x aufgelöst werden kann. Die Lösung kann also nicht mit den herkömmlichen Lösungsmethoden für Gleichungen gefunden werden. Dies war einer der Gründe für die Entwicklung von iterativen Lösungsverfahren. Mit ihnen kann die Lösung einer beliebig komplizierten Gleichung relativ einfach auf beliebig viele Nachkommastellen angenähert werden. Es wird zwar keine exakte Lösung gefunden, aber immerhin eine Annäherung. Als Vereinfachung wird das Problem auf die Nullstellensuche von eindimensionalen, stetig differenzierbaren Gleichungen begrenzt. Die Erweiterung auf mehrdimensionale Gleichungssysteme ist nicht mehr weit, wenn die eindimensionalen Verfahren bekannt sind.

1.2. Ziel

Ziel dieser Arbeit ist es, die drei wichtigsten iterativen Lösungsverfahren zu besprechen und spezifisch zu testen. Es sollen die Schwächen und Stärken der jeweiligen Verfahren herausgefunden werden, mit welchen Geschwindigkeiten sie konvergieren bzw. divergieren. Es soll erklärt werden wieso und wann diese Verfahren funktionieren und bei welchen Bedingungen mit Sicherheit eine Konvergenz vorhergesagt werden kann. Ausserdem sollen die Verfahren für die Erweiterung der Verständnis gut und einfach veranschaulicht werden.

1.3. Vorgehen

Um einen allgemeinen Überblick über dieses Thema zu erhalten werden zuerst alle Verfahren grundsätzlich beschrieben. Mithilfe eines für diesen Zweck geschriebenen SciLab-Programmes wird jeweils ein Beispiel der Anwendung des Verfahrens gezeigt, um eine Konvergenz zu bestätigen. Im nächsten Schritt werden die Verfahren genauer untersucht. Durch Umformungen und Abschätzungen der Rechenschritte werden dessen Konvergenzkriterien und Konvergenzgeschwindigkeiten bestimmt. Schlussendlich werden diese Verfahren genauer getestet mithilfe eines selbst entwickelten Java-Programms. Es liefert alle nötigen Werkzeuge um das Konvergenzverhalten zu untersuchen.

1.4. Aufbau

Nach der Einleitung werden im Kapitel 2 wichtige Begriffe wie das Lösungsverfahren, die Konvergenzgeschwindigkeit und das Abbruchkriterium erläutert, damit einen Einstieg ins Thema

einfach möglich ist.

Im Kapitel 3 werden die drei wichtigsten und bekanntesten Verfahren beschrieben und dessen Umsetzung erklärt. Darunter befinden sich das Bisektions-, Fixpunkt- und Newtonverfahren. Ausserdem wird jeweils ein kleines Beispiel dazu gezeigt, um zu illustrieren, wie es konkret funktioniert.

Weiter im Kapitel 4 werden grundsätzliche Eigenschaften der Verfahren hergeleitet, darunter das Konvergenzkriterium und die Konvergenzgeschwindigkeit. Damit kann unerwartetes Verhalten der Lösungsverfahren begründet werden.

Im Kapitel 5 werden die vorgestellten Verfahren auf ihre Anwendung in der Praxis getestet. Es werden drei verschiedene Funktionen auf drei verschiedene Genauigkeiten getestet. Dies wird mit einem von mir für diesen Zweck entwickelten Java-Programmes durchgeführt.

Das letzte Kapitel 6 enthält eine Zusammenfassung mit den Vor- und Nachteilen der besprochenen Lösungsverfahren.

1.5. Motivation

Heutzutage braucht man oft einen Taschenrechner, selbstverständlich erwartet man von ihm, dass er Gleichungen lösen kann, auch wenn sie recht kompliziert sind. Der Taschenrechner verwendet ebenfalls iterative Lösungsmethoden, meistens das Newton-Verfahren. Warum genau das Newtonverfahren, wenn es doch zahlreiche andere Verfahren gibt, was ist so speziell an dieser Methode?

1.6. Wieso iterative Verfahren

Schon in der Sekundarstufe, als man den Logarithmus behandelte, kam ich mit der Gleichung $2^x + 3^x = 10$ auf, indem ich schon bekanntes verkettete. Für mich klang das wie eine Herausforderung, dessen Lösung nach ein paar Minuten gefunden werden kann. Ich versuchte es, doch die Lösung blieb mir unbekannt und ich vermutete, dass ich einfach noch nicht dazu fähig war. Ein paar Jahre später stolperte ich im Internet über algebraische Gleichungen, also Gleichungen, welche mittels der Algebra gelöst werden können. Mir wurde bewusst, dass es auch nicht-algebraische Gleichungen gibt und vermutete schon, dass jene eine sein könnte. Erst ein Weilchen später entdeckte ich die numerischen Lösungsverfahren mit denen beliebige Gleichungen gelöst werden können.

2. Definitionen

2.1. Lösungsverfahren

Iterative Lösungsverfahren sind Methoden bzw. Algorithmen, welche mithilfe von wiederholten Aufrufen derselben Rechenschritte, der Lösung einer Gleichung immer näher kommen. Jedes dieser Verfahren benötigt einen oder mehrere Startwerte. Dabei kann es vorkommen, dass bei bestimmten Startwerten und Funktionen keine Konvergenz vorliegt und somit keine Annäherung an eine Nullstelle gefunden wird.

2.2. Konvergenzgeschwindigkeit

Die Konvergenzgeschwindigkeit gibt an, mit welcher Geschwindigkeit eine konvergente Folge sich seinem Grenzwert nähert. Dabei wird der Faktor angegeben, mit dem die Differenzen zur exakten Lösung kleiner werden pro Schritt.

2.2.1. Lineare Konvergenzgeschwindigkeit

Lineare Konvergenzgeschwindigkeit liegt vor, wenn sich eine Folge $\{x_n\}(n \in \mathbb{N})$ in jedem Schritt um mindestens einen konstanten Faktor c bzw. eine Folge $\{c_n\}$ mit $0 < c < 1$ dem Grenzwert $\tilde{x}$ nähert.

$$|x_{n+1} - \tilde{x}| \leq c|x_n - \tilde{x}| \quad \text{für } 0 < c < 1 \tag{2.1}$$

Eine äquivalente Definition mit dem Limes ist

$$\lim_{n\to\infty} \frac{|x_{n+1} - \tilde{x}|}{|x_n - \tilde{x}|} = c$$

Beispiel Die Folge $\frac{1}{2}^k(k \in \mathbb{N})$ konvergiert mit linearer Geschwindigkeit und $c = \frac{1}{2}$ zum Grenzwert Null, da die Glieder pro Schritt um den Faktor $\frac{1}{2}$ der Lösung näher kommen. Allgemein konvergiert die Folge $\frac{1}{a}^k(k \in \mathbb{N}, |a| > 1)$ mit $c = \frac{1}{|a|}$. Ein Beweis dazu befindet sich im Anhang (A.1).

2.2.2. Konvergenzgeschwindigkeit der Ordnung p

Konvergenzgeschwindigkeit der Ordnung p tritt auf, wenn sich die Differenz mit der Potenz p verkleinert:

$$|x_{n+1} - x| \leq c|x_n - x|^p$$

Ebenso kann dies auch durch den Limes formuliert werden:

$$\lim_{n\to\infty} \frac{|x_{n+1} - \tilde{x}|}{|x_n - \tilde{x}|^p} = c$$

Konkret heisst das, dass sich mit jedem Iterationsschritt die Anzahl korrekter Stellen etwa um den Faktor p vergrössert. Somit ist ein grosses p sehr vorteilhaft.

Beispiel Das Newtonverfahren konvergiert mit $p = 2$, was in Kapitel 4.2.1 hergeleitet wird.

Das Halley-Verfahren hat eine kubische Konvergenzordnung ($p = 3$). Nachteil dieses Verfahrens ist, dass es die erste sowie die zweite Ableitung für einen Iterationsschritt benötigt. Auch muss sehr oft die Funktion ausgewertet werden, was sich wiederum auf die Rechenzeit auswirkt.

Das Sekantenverfahren, welches im Anhang (A.2) kurz behandelt wird, konvergiert mit der Ordnung $p = \phi = \frac{1+\sqrt{5}}{2} \approx 1.610803$ also der Goldenen Zahl.

2.3. Abbruchkriterium

Idealerweise werden die Iterationsschritte abgebrochen, sobald die Abweichung zur exakten Nullstelle die gewünschte Genauigkeit ϵ unterschritten hat:

$$|x_n - \tilde{x}| < \epsilon \tag{2.2}$$

Da aber diese Nullstelle meist nicht bekannt ist, muss man andere Abbruchkriterien in Betracht ziehen. Die eine Möglichkeit ist abzubrechen, wenn die Differenz zweier nachfolgenden Näherungen ein bestimmtes ϵ unterschreitet. Damit gibt es keine Probleme mehr, da die beiden Näherungen immer bekannt sind. Allerdings gibt es keine Garantie, dass nicht zu einem unglücklichen Zeitpunkt abgebrochen wird.

$$|x_{n+1} - x_n| < \epsilon \tag{2.3}$$

Es gibt auch noch die andere Möglichkeit abzubrechen, wenn der Funktionswert der Näherung ein bestimmtes ϵ unterschreitet. Der Nachteil dabei ist, dass die Funktion ein weiteres Mal ausgewertet werden muss, was zeitintensiver wird.

$$|f(x_n)| < \epsilon \tag{2.4}$$

Obwohl die drei hier verwendeten ϵ nicht ein und dasselbe sind, können sie grundsätzlich als gleiches verwendet werden. Wenn deshalb in den folgenden Kapiteln von einer Genauigkeit von zehn Nachkommastellen bzw. $\epsilon = 0.5 \cdot 10^{-10}$ gesprochen wird, bezeichnet dies jeweils das ϵ für das verwendete Abbruchkriterium und es kann sein, dass aufgrund dieses Kriteriums nicht die gewünschten zehn Stellen erreicht werden.

3. Vorstellung der Verfahren

Es gibt dutzende von numerischen Lösungsverfahren. Eine Selektion von jenen werden in diesem Kapitel vorgestellt, deren Funktionsweisen beschrieben und ein konkretes Beispiel gerechnet. Dazu gehören das Bisektionsverfahren, das Fixpunktverfahren und das Newton-Verfahren.

3.1. Bisektionsverfahren

3.1.1. Beschreibung

Dieses Verfahren ist das wohl Bekannteste. Das Prinzip ist einfach: Es wird ein Bereich der Funktion gefunden, in dem garantiert eine Nullstelle vorkommt. Dieser wird nun in zwei Abschnitte geteilt und anschliessend ermittelt, in welchem sich eine Nullstelle befindet. Nun wiederholt man diese Schritte mit dem neuen Intervall bis die gewünschte Genauigkeit erreicht ist.

3.1.2. Umsetzung

Ausgegangen wird vom Startintervall $[a_0, b_0]$, welches beim Start vorgegeben werden muss. Um sicherzustellen, dass das Startintervall mindestens eine Nullstelle enthält, müssen die beiden Funktionswerte der Randstellen unterschiedliche Vorzeichen haben bzw. ihr Produkt muss kleiner als null sein.

Als nächstes wird die Mitte des Startintervalls berechnet und als q_0 gespeichert.

Nun muss ermittelt werden, in welchem Teilintervall, $[a_0, q_0]$ oder $[q_0, b_0]$, sich eine Nullstelle befindet, indem man wieder die Funktionswerte der Randstellen miteinander multipliziert, eine Nullstelle ist vorhanden, falls dieser Wert negativ ist.

Wenn sie sich im linken Teilintervall befindet, wird die rechte Intervallgrenze b_0 durch die berechnete Mitte q_0 ersetzt; die linke Intervallgrenze wird beibehalten. Andernfalls wird die linke Intervallgrenze a_0 durch q_0 ersetzt[1].

Dieses Vorgehen wird ab der Berechnung der neuen Intervallmitte solange wiederholt, bis die Intervallbreite die gewünschte Genauigkeit unterschreitet.

Algorithmus Wähle das Startintervall $[a_0, b_0]$ mit $\operatorname{sgn}(f(a_0)) \neq \operatorname{sgn}(f(b_0))$, welches garantiert eine Nullstelle enthält

Berechne die Mitte $q_0 = \frac{a_0+b_0}{2}$ zwischen den beiden Randstellen.

Da der Abstand der Näherung zur Nullstelle immer kleiner als das Intervall sein muss, kann das Abbruchkriterium (2.2) zu $|b_n - a_n| < \epsilon$ umgeformt werden. Bis das Abbruchkriterium eintritt,

[1] Mithilfe des Bisektionsverfahren können auch mehrere Nullstellen im Startintervall gefunden werden, wenn auf jedes Intervall, welches eine Nullstelle enthält, das Verfahren angewendet wird. Es ist aber nicht möglich zu sagen, ob alle Nullstellen gefunden wurden.

wiederhole die folgenden Schritte mit steigenden n:

$$
\begin{aligned}
[a_{n+1}, b_{n+1}] &= \begin{cases} [q_n, b_n] & \text{falls } \operatorname{sgn}(f(q_n)) = \operatorname{sgn}(f(a_n)) \\ [a_n, q_n] & \text{falls } \operatorname{sgn}(f(q_n)) = \operatorname{sgn}(f(b_n)) \end{cases} \\
q_{n+1} &= \frac{a_{n+1} + b_{n+1}}{2}
\end{aligned}
$$

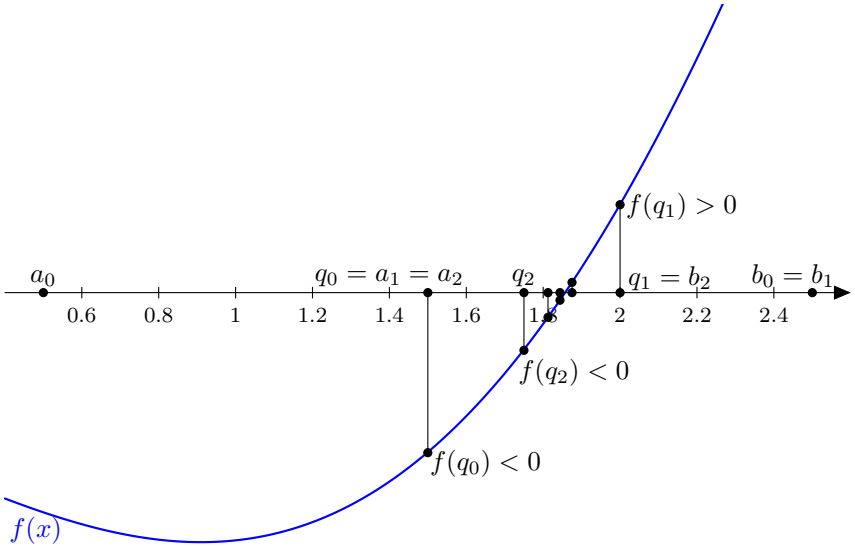

Abbildung 1: Illustration des Bisektionsverfahrens. Die beiden Startwerte a_0 und b_0 haben unterschiedliche Vorzeichen.

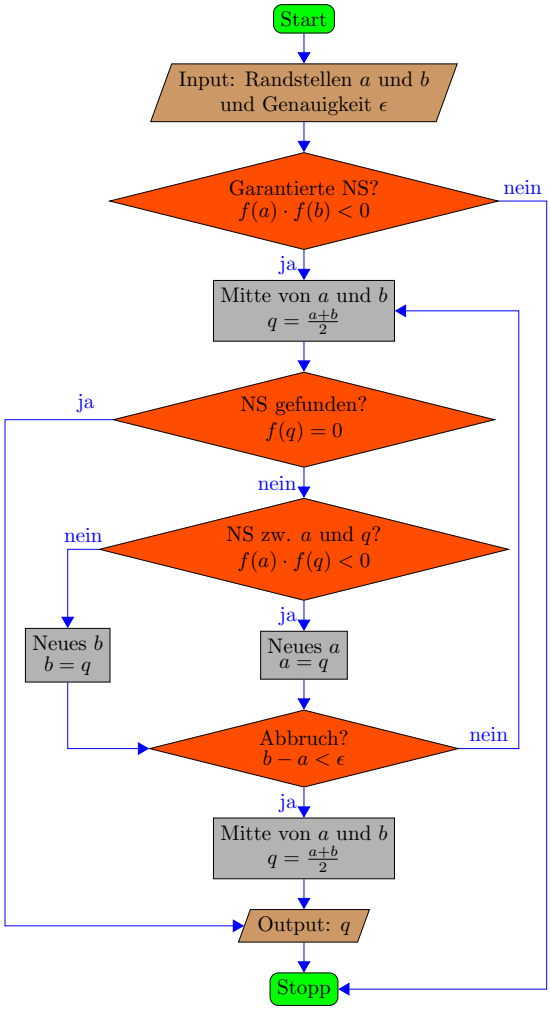

Abbildung 2: Detaillierter Programmablaufplan des Bisektionsverfahrens. Es wird jeweils geprüft, ob garantiert eine Nullstelle (NS) im Startintervall vorhanden ist und ob die Nullstelle mit einem Iterationsschritt genau erreicht wurde.

Mithilfe dieses Algorithmus kann das Bisektionsverfahren in SciLab implementiert werden:

```
1  // a, b Grenzen des Intervalls; e Genauigkeit Epsilon
2  function [q] Bisektion(a, b, e)
3  while b - a >= e,              // Abbruchkriterium
4      q = (a + b) / 2
5      if f(a) * f(q) < 0 then // Ersetzung der
6          a = q                  // unteren
7      else                       // bzw.
8          b = q                  // oberen
9      end                        // Intervallsgrenze mit q
10 end
```

3.1.3. Beispiel

Als Beispiel wird die Funktion $f(x) = x^3 - 2x^2 + 1$ genommen, welche in Abbildung 3 dargestellt ist.

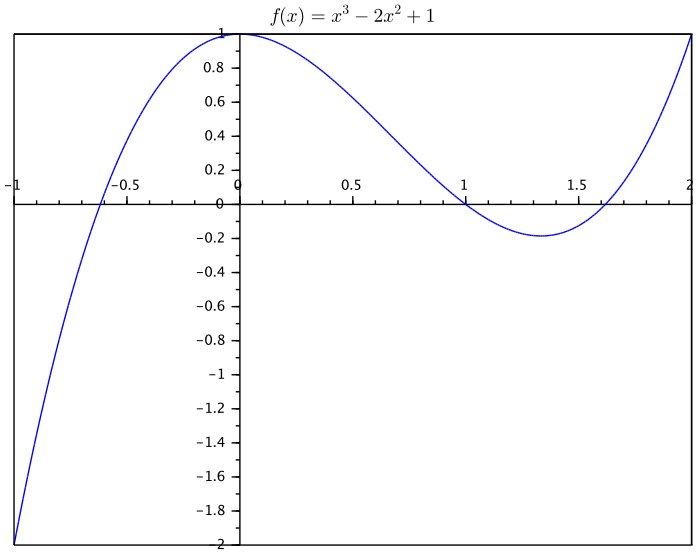

Abbildung 3: Beispielfunktion für das Bisektionsverfahren $f(x) = x^3 - 2x^2 + 1$

An der Grafik kann abgelesen werden, dass die Funktion eine Nullstelle im Intervall $[1.5, 2]$ besitzt also $a_0 = 1.5$ und $b_0 = 2$. q_0 ist somit $\frac{1.5+2}{2} = 1.75$. Die Genauigkeit soll $\epsilon = 0.005 = 0.5 \cdot 10^{-2}$ bzw. zwei Nachkommastellen betragen. Mithilfe des SciLab-Programmes kann nun Tabelle 1 erstellt werden:

n	a_n	b_n	$b_n - a_n$	q_n	$f(q_n)$
0	1.5000	2.0000	0.5000	1.7500	0.2344
1	1.5000	1.7500	0.2500	1.6250	0.0098
2	1.5000	1.6250	0.1250	1.5625	−0.0681
3	1.5625	1.6250	0.0625	1.5938	−0.0319
4	1.5938	1.6250	0.0312	1.6094	−0.0118
5	1.6094	1.6250	0.0156	1.6172	−0.0012
6	1.6172	1.6250	0.0078	1.6211	0.0043
7	1.6172	1.6211	0.0039	1.6191	0.0015

Tabelle 1: Output eines Programmes zur Berechnung von Nullstellen mithilfe des Bisektionsverfahrens

Schlussendlich hat man eine Nullstelle gefunden bei $q = 1.62 \pm 0.005$. Der Rechenaufwand war mit sieben Schritten relativ hoch, wenn man bedenkt, dass lediglich zwei Nachkommastellen bestimmt werden konnten.

3.2. Fixpunktverfahren

3.2.1. Beschreibung

Das Ziel bei diesem Verfahren ist es, einen Fixpunkt zu finden. Ein Fixpunkt ist ein Punkt, in dem der Funktionswert dem Funktionsargument entspricht. Fixpunkte erfüllen somit eine Gleichung der Form $f(x) = x$, auch Fixpunktgleichung genannt.

3.2.2. Umsetzung

Der erste Schritt ist eine Fixpunktgleichung $\varphi(x)$ zu finden. Die namengebende Eigenschaft einer solchen ist, dass die Gleichung für jeden Fixpunkt wahr ist. In den Beispielen wird gezeigt wie solche zu finden sind.

Wenn erstmal eine solche Gleichung gefunden ist, muss festgestellt werden, in welchen Bereichen diese überhaupt konvergiert. Dies wird im Kapitel 4.3.1 hergeleitet.

Als nächstes muss ein Startwert gefunden werden. Man kann diesen frei im Konvergenzbereich wählen. Um die folgende Näherung zu berechnen, setzt man den Anfangswert in die Fixpunktgleichung ein.

$$q_{n+1} = \varphi(q_n)$$

So fährt man weiter, solange das Abbruchkriterium nicht erfüllt ist. Abbildung 4 zeigt diesen Verlauf im Falle der Konvergenz. Abbildung 5 zeigt den Programmablaufplan des Fixpunktverfahrens.

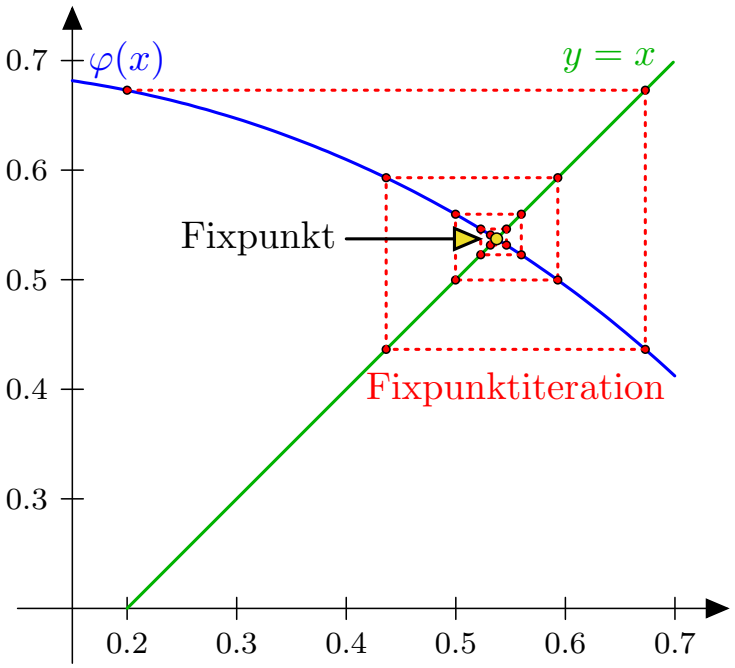

Abbildung 4: Illustration des Fixpunktverfahrens mit der Fixpunktgleichung $\varphi(x) = \ln(2 - x^2)$ und dem Startwert $q_0 = 0.2$. Es wird jeweils der Funktionswert $y = f(x)$ der Näherungen berechnet und durch eine waagerechte Projektion auf $y = x$ entsteht die Folge $x = f(x)$

Mit den folgenden beiden Beispielen wird illustriert, wie eine Fixpunktgleichung und somit evtl. eine Schnittstelle gefunden werden kann. Im Kapitel 4.3.1 wird genauer auf das Konvergenzkriterium eingegangen.

3.2.3. Beispiele

Beispiel 1: $2^x + 3^x = 10$ mit der einzigen Lösung $\tilde{x} = 1.729255558981\ldots$. Es wird eine Fixpunktgleichung gefunden, indem man zur Nullgleichung x addiert:

$$\begin{aligned} 2^x + 3^x &= 10 && || - 10 + x \\ 2^x + 3^x - 10 + x &= x \\ \varphi(x) &= 2^x + 3^x - 10 + x \end{aligned}$$

Wenn wir den Startwert $q_0 = 2$ einsetzen ergibt sich folgendes:

$$q_1 = \varphi(2) = 2^2 + 3^2 - 10 + 2 = 5 \qquad q_2 = \varphi(5) = 2^5 + 3^5 - 10 + 5 = 270$$
$$q_3 = \varphi(270) \approx 6.65 \cdot 10^{128} \qquad q_4 \approx \varphi(6.65 \cdot 10^{128}) \approx 2.98 \cdot 10^{10^{128}}$$

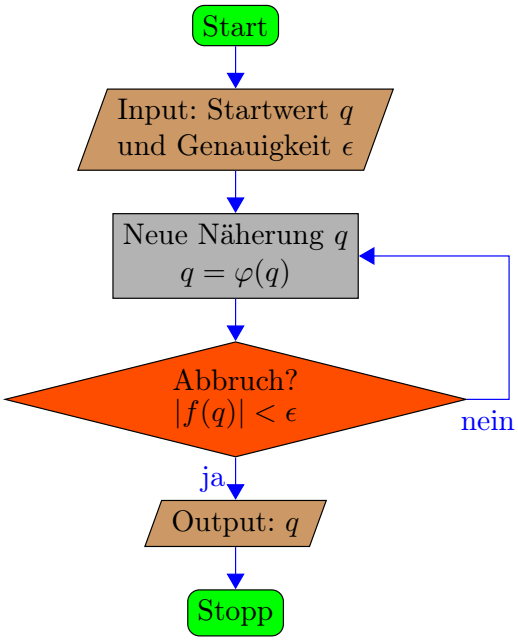

Abbildung 5: Programmablaufplan des Fixpunktverfahrens. Es wird sichtbar, dass dieses Verfahren sehr einfach durchzuführen ist. Hier wurde das Abbruchkriterium (2.4) verwendet.

Wie man sieht, divergiert diese Folge sehr schnell ins Unendliche. Mit dem Startwert $q_0 = 1.5$ entstehen folgende Werte:

$$q_1 = \varphi(1.5) \approx -0.48 \qquad q_2 \approx \varphi(-0.48) \approx -9.16$$
$$q_3 \approx \varphi(-9.16) \approx -19.16 \qquad q_4 \approx \varphi(-19.16) \approx -29.16$$

Die Folge divergiert ebenfalls, somit ist die Wahrscheinlichkeit gross, dass diese Fixpunktgleichung für alle Werte (ausser der Lösung) divergiert und somit nicht für die Suche von Lösungen gebraucht werden kann.

Beispiel 2: $x^3 - 2x^2 + 1 = 0$ mit den drei reellen Lösungen $x_1 = \frac{1-\sqrt{5}}{2}$, $x_2 = 1$ und $x_3 = \frac{1+\sqrt{5}}{2}$.
In diesem Beispiel wird eine Alternative, das Auflösen nach x, gezeigt.

$$\begin{aligned} x^3 - 2x^2 + 1 = 0 \quad &\Leftrightarrow \quad x = \sqrt[3]{2x^2 - 1} \\ \Rightarrow \varphi_1(x) \quad &= \quad \sqrt[3]{2x^2 - 1} \end{aligned}$$

Somit ist eine Fixpunktgleichung gefunden. Ebenfalls können weitere gefunden werden, indem man auf andere Weise umformt:

$$x^3 - 2x^2 + 1 = 0 \quad \Leftrightarrow \quad x^2 = \frac{x^3 + 1}{2}$$
$$\Rightarrow \varphi_2(x) = \sqrt{\frac{x^3 + 1}{2}} \quad \wedge \quad \varphi_3(x) = -\sqrt{\frac{x^3 + 1}{2}}$$

Mit diesen drei Fixpunktgleichungen $\varphi_{1,2,3}$ wird die Abbildung 6 erstellt. Gut zu erkennen sind die Schnittstellen der Fixpunktgleichungen mit der Winkelhalbierenden. Diese Eigenschaft lässt sich einfach erklären, da $y = \varphi(x) = x$.

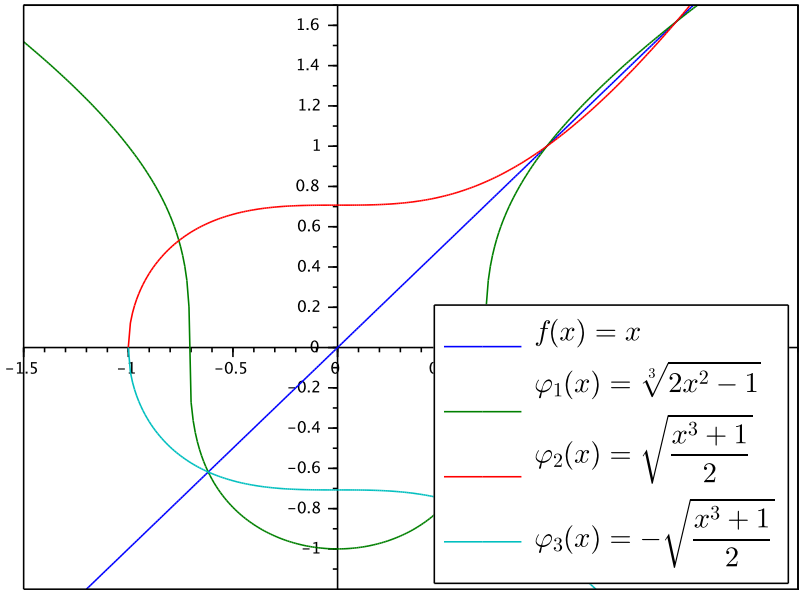

Abbildung 6: Drei verschiedene Fixpunktgleichungen für die Funktion $x^3 - 2x^2 + 1 = 0$

Mit der Fixpunktgleichung $\varphi_1(x)$ und dem Startwert $q_0 = 1.5$ ergibt sich nach 6 Schritten Tabelle 2.

n	q_n	$\lvert q_n - q_{n-1}\rvert$
1	1.5182945	0.0182945
2	1.5340984	0.0158040
3	1.5476435	0.0135450
4	1.5591756	0.0115322
5	1.5689396	0.0097640
6	1.5771683	0.0082287

Tabelle 2: Output eines Programmes zur Berechnung von Nullstellen mithilfe des Fixpunktverfahrens.

Wie man sieht, konvergiert die Folge q_n mit immer kleiner werdendem $|q_n - q_{n-1}|$ zur Lösung $x_3 = \frac{1+\sqrt{5}}{2}$ hin. Im Kapitel 4.3.1 wird weiter darauf eingegangen.

3.3. Newton-Verfahren

3.3.1. Beschreibung

Das Newton-Verfahren (nach ISAAC NEWTON, 1669) basiert auf der Idee, eine Tangente an den Funktionspunkt der Näherung zu legen, deren Schnittpunkt mit der Abszissenachse zu finden und diesen als neue Näherung zu verwenden. Da die Funktion für die Tangente abgeleitet werden muss, hat diese eine C^1 Funktion zu sein.

3.3.2. Umsetzung

Angefangen wird mit einem ersten Näherungswert q_0, welcher frei im Definitionsbereich der Funktion gewählt werden kann. Dieser Wert beeinflusst das Konvergenzverhalten des Verfahrens.

Nun wird die Steigung des Funktionsgraphen im Punkt $P_0(q_0|f(q_0))$ mithilfe der Ableitung ermittelt und eine Gleichung der Tangente $t_0(x)$ durch den Punkt P_0 gefunden.

Der Graph dieser linearen Funktion wird mit der x-Achse geschnitten. Somit findet man die neue Näherung q_1, mit welcher diese Schritte wiederholt werden können. Abbildung 7 zeigt dieses Vorgehen.

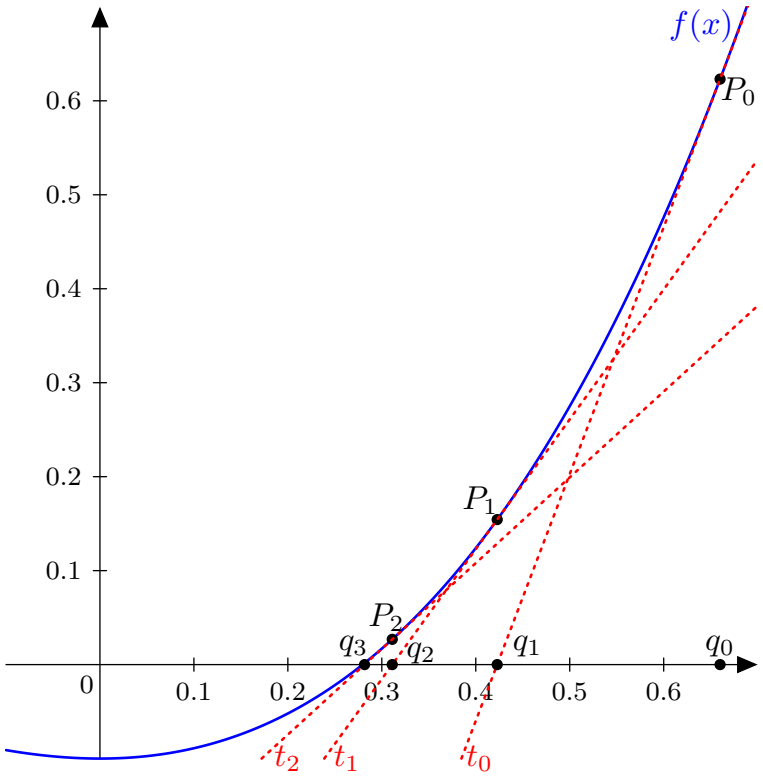

Abbildung 7: Illustration des Newton-Verfahrens

Diese Schritte werden wiederholt, bis das Abbruchkriterium eintritt. Dafür kann gewählt werden zwischen den Abbruchkriterien 2.3 und 2.4.

Mithilfe des nun beschriebenen Vorgehens kann ein einfacher Algorithmus gefunden werden:

Wähle ein Abszissenstartwert $q_0 \in \mathrm{D}$. Ermittle die Ableitung der Funktion und somit die Steigung m_0 bei $x = q_0$.

$$m_0 = f'(q_0)$$

Damit kann ein Ansatz der Gleichung für die Tangente $t_0(x)$ durch den Punkt $P_0(q_0|f(q_0))$ gefunden werden

$$t_0(x) = m_0 x + a_0 = f'(q_0)x + a_0$$

und durch Einsetzen des Punktes P_0 kann das Absolutglied a_0 berechnet werden.

$$f(q_0) = f'(q_0)q_0 + a_0$$
$$a_0 = f(q_0) - f'(q_0)q_0$$

Somit ist die Gleichung der Tangente durch den Punkt P_0

$$t_0(x) = f'(q_0)x + f(q_0) - f'(q_0)q_0$$

Durch Nullsetzen und Auflösen nach x wird die Schnittstelle mit der Abszissenachse gefunden.

$$\begin{aligned} 0 &= f'(q_0)x + f(q_0) - f'(q_0)q_0 && || - f(q_0) + f'(q_0)q_0 \\ f'(q_0)x &= f'(q_0)q_0 - f(q_0) && || \cdot \frac{1}{f'(q_0)} \text{ mit } f'(q_0) \neq 0 \\ x &= \frac{f'(q_0)q_0 - f(q_0)}{f'(q_0)} && || \text{ vereinfachen} \\ x &= q_0 - \frac{f(q_0)}{f'(q_0)} \end{aligned}$$

Für den Sonderfall $f'(q_0) = 0$ muss ein anderer Startwert gewählt werden. x wird nun als neuen Näherungswert q_1 definiert

$$q_1 = q_0 - \frac{f(q_0)}{f'(q_0)}$$

Algorithmus Wähle einen Startwert $q_0 \in \mathrm{D}$.

Wiederhole folgenden Schritt:

$$q_{n+1} = q_n - \frac{f(q_n)}{f'(q_n)}$$

Wobei $f'(q_n)$ nie Null sein darf. Als Abbruchkriterium gibt es die beiden Möglichkeiten 2.3 und 2.4

$$|f(q_n)| < \epsilon \quad \text{oder} \quad |q_n - q_{n-1}| < \epsilon$$

Bei beiden Abbruchkriterien kann jedoch nicht ausgeschlossen werden, dass nicht bei einem ungünstigen Zeitpunkt abgebrochen wird.

Die Implementation mit dem Abbruchkriterium $|f(q_n)| < \epsilon$ sieht in SciLab wie folgt aus:

```
1 while abs(f(q)) >= e,          // Abbruchkriterium
2     q=q-f(q)/f_Abl(q)          // Neuer Wert für q berechnen
3 end
```

In Abbildung 8 ist der Programmablaufplan des Newton-Verfahrens dargestellt.

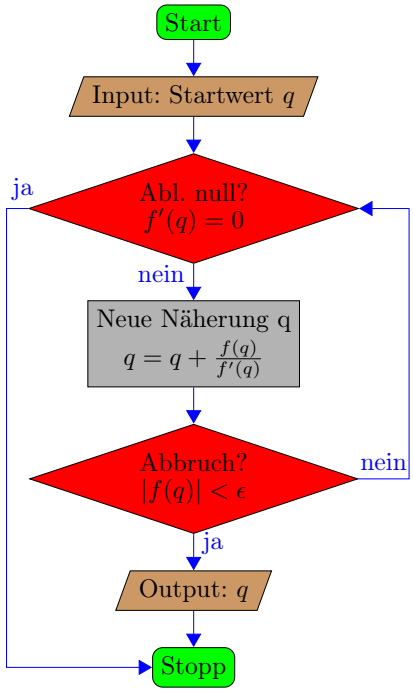

Abbildung 8: Programmablaufplan des Newton-Verfahrens

3.3.3. Beispiel

Es wird wieder die gleiche Funktion verwendet: $f(x) = x^3 - 2x^2 + 1$. Diesmal mit einer Genauigkeit von $\epsilon = 0.5 \cdot 10^{-14}$ bzw. 14 Nachkommastellen für den Funktionswert von q_n und einem Startwert $q_0 = 2$. Das Abbruchkriterium ist $|f(q_n)| < \epsilon$.

n	q_n	$f(q_n)$
0	2.0000000000000000	1.0000000000000000
1	1.7500000000000000	0.2343750000000000
2	1.6428571428571428	0.0360787172011658
3	1.6192068800764454	0.0016248238757495
4	1.6180368184751353	0.0000039106069574
5	1.6180339887664317	0.0000000000228537
6	1.6180339887498945	−0.0000000000000009

Tabelle 3: Output eines Programmes zur Berechnung von Nullstellen mithilfe des Newton-Verfahrens

Damit kommt man auf das Resultat $q_4 \approx 1.61803398874989$ in nur sechs Schritten, was im

Vergleich zum Bisektionsverfahren eine grosse Verbesserung ist, denn jenes kam mit sieben Schritten gerade einmal auf eine Genauigkeit von zwei Nachkommastellen.

4. Analyse

4.1. Bisektion

4.1.1. Konvergenzkriterium

Das Bisektionsverfahren konvergiert genau dann, wenn $[a_0, b_0]$ mindestens eine Nullstelle enthält. Dies ist ein grosser Vorteil dieses Verfahrens, welchen die anderen Verfahren nicht besitzen. Die Schwierigkeit liegt somit nicht in der Konvergenz des Verfahrens, sondern in der richtigen Wahl des Intervalls.

4.1.2. Konvergenzgeschwindigkeit

Da beim Bisektionsverfahren die Intervalllänge pro Iteration sich um den Faktor zwei verkürzt

$$b_{n+1} - a_{n+1} = \frac{1}{2} |b_n - a_n|$$

kann die Intervalllänge bei der n-ten Iteration als

$$b_n - a_n = \frac{1}{2^{n-1}} |b_0 - a_0| \tag{4.1}$$

angegeben werden. Da für die Nullstelle $\tilde{q} \in [a_n, q_n]$ oder $\tilde{q} \in [q_n, b_n]$ gilt, kann man aussagen:

$$|q_n - \tilde{q}| \leq q_n - a_n = b_n - q_n = \frac{1}{2} |b_n - a_n|$$

Wenn wir nun dies mit der Gleichung (4.1) kombinieren, ergibt sich die Ungleichung

$$|q_n - \tilde{q}| \leq \frac{1}{2^n} |b_0 - a_0|$$

und somit konvergiert das Bisektionsverfahren mit $c = \frac{1}{2}$ und Ordnung $p = 1$. Das heisst durchschnittlich verringert sich die maximale Abweichung zur Nullstelle um den Faktor $\frac{1}{2}$. Somit benötigt man etwa $\log_2 10 \approx 3.32$ Schritte um eine weitere Nachkommastelle der Nullstelle zu erhalten.

4.1.3. Anzahl benötigter Iterationen

Um zu berechnen wie viele Schritte notwendig sind, um den maximalen Fehler ϵ zu unterschreiten, wird die Abweichung $|q_n - \tilde{q}|$ mit $\frac{1}{2^n}(b_0 - a_0)$ ersetzt:

$$\frac{1}{2^n}(b_0 - a_0) \leq \epsilon$$

Aufgelöst nach n ergibt sich:

$$n \geq \log_2 \left(\frac{b_0 - a_0}{\epsilon} \right)$$

Also kann die Anzahl benötigter Schritte sogar ohne Kenntnis der Funktion berechnet werden. Beim Beispiel in Kapitel 3.1.3 mit $a_0 = 1.5$, $b_0 = 2$ und $\epsilon = 0.005$ kann die Anzahl Schritte schon berechnet werden mit

$$n \geq \log_2 \left(\frac{2 - 1.5}{0.005} \right) = 6.643 \ldots$$

also $n = 7$, was korrekt ist.

4.2. Newton-Verfahren

4.2.1. Konvergenzgeschwindigkeit und Konvergenzkriterium

Ein Iterationsschritt vom Newtonverfahren kann umgeformt werden zu:

$$x_{n+1} - x_n = -\frac{f(x_n)}{f'(x_n)}$$

Indem man beide Seiten mit $-f'(x_n)$ multipliziert erhält man:

$$f(x_n) = -f'(x_n) \cdot (x_{n+1} - x_n) \tag{4.2}$$

Die Funktion $f(x)$ als Taylor-Polynom erster Ordnung mit Restglied ist

$$f(x) = f(x_n) + f'(x_n) \cdot (x - x_n) + \frac{f''(\xi)}{2!} \cdot (x - x_n)^2$$

Mit einem ξ zwischen x und x_n. Nun setzt man (4.2) für $f(x_n)$ ein:

$$\begin{aligned} f(x) &= -f'(x_n) \cdot (x_{n+1} - x_n) + f'(x_n) \cdot (x - x_n) + \frac{f''(\xi)}{2!} \cdot (x - x_n)^2 \\ f(x) &= f'(x_n) \cdot (x - x_{n+1}) + \frac{f''(\xi)}{2!} \cdot (x - x_n)^2 \end{aligned}$$

Wenn wie beim Newtonverfahren eine Nullstelle gesucht ist, kann man $f(x)$ gleich null setzen, und x mit der gesuchten Nullstelle $\tilde{x}$ ersetzen. Durch Einsetzen und Auflösen:

$$\begin{aligned} 0 &= f'(x_n) \cdot (\tilde{x} - x_{n+1}) + \frac{f''(\xi)}{2!} \cdot (\tilde{x} - x_n)^2 \\ |x_{n+1} - \tilde{x}| &= \left| \frac{f''(\xi)}{2f'(x_n)} \right| \cdot |x_n - \tilde{x}|^2 \end{aligned}$$

Ersetzt man nun $f''(\xi)$ mit $\max |f''(x)|$ und $f'(x_n)$ mit $\min |f'(x)| \neq 0$ mit einem x aus einem Intervall I in der Umgebung der Nullstelle, ergibt sich die Ungleichheit

$$|x_{n+1} - \tilde{x}| \leq c \cdot |x_n - \tilde{x}|^2 \quad \text{mit } c = \frac{\max |f''(x)|}{2 \min |f'(x)|}, x \in I \tag{4.3}$$

$$\min_{x \in I} |f'(x)| \neq 0 \tag{4.4}$$

Also konvergiert das Newtonverfahren im Intervall I mit quadratischer Konvergenzordnung[2]. Mit (4.3) folgt:

$$c|x_n - \tilde{x}| \leq c|x_{n-1} - \tilde{x}|^2 \leq c^3|x_{n-2} - \tilde{x}|^4 \leq \cdots \leq c^{2^n-1}|x_0 - \tilde{x}|^{2^n}$$

Damit kann weiter abgeleitet werden:

$$c^{2^n-1}|x_0 - \tilde{x}|^{2^n} \quad = \quad c^{2^n-1}|x_0 - \tilde{x}|^{2^n-1}|x_0 - \tilde{x}|^1 = (c|x_0 - \tilde{x}|)^{2^n-1}\,|x_0 - \tilde{x}|$$

Mit dieser Schreibweise wird klar, dass das Newtonverfahren dann konvergiert, wenn $c|x_0 - \tilde{x}| < 1$ und somit $|x_0 - \tilde{x}| < \frac{1}{c}$.

4.3. Fixpunktverfahren

4.3.1. Konvergenzkriterium

Das Fixpunktverfahren konvergiert nach dem Banachschen Fixpunktsatz in Anhang A.3 im abgeschlossenen Intervall $\mathrm{D} = [a, b]$ genau dann zum Fixpunkt x^*, wenn die Fixpunktgleichung $f(x)$ selbstabbildend und kontrahierend ist.

Damit die Funktion selbstabbildend im Intervall D ist, muss der Wertebereich der Funktion in D sein.

Damit die Funktion kontrahierend ist, muss nach dem Mittelwertsatz der Differentialrechnung (Anhang A.4) und dem Fixpunktsatz von Banach

$$\max\{|\varphi'(x)| : x \in \mathrm{D}\} < 1$$

gelten. Das heisst in der Umgebung um die Nullstelle muss der Betrag der Ableitung kleiner als eins sein, dann ist die Konvergenz sicher. Als Abschätzung kann nur die Steigung beim Fixpunkt alleine berechnet werden.

$$|\varphi'(\tilde{x})| < 1 \tag{4.5}$$

4.3.2. Beispiel

Damit kann auch erklärt werden, warum das erste Beispiel bei der Einführung vom Fixpunktverfahren in Kapitel 3.2.3 nicht konvergierte. Die Fixpunktgleichung war

$$\varphi(x) = 2^x + 3^x - 10 + x$$

und somit ist deren Ableitung

$$\varphi'(x) = \ln 2 \cdot 2^x + \ln 3 \cdot 3^x + 1$$

Weil 2^x bzw. 3^x für alle x grösser als null ist, muss

$$\varphi'(x) > 1$$

was heisst, dass das obig genannte Kriterium (4.5) nie eintreffen kann! Also divergiert diese Fixpunktgleichung für beliebige Startwerte.

[2] mehrfache ns

5. Testen der Verfahren

5.1. Disziplinen

Alle Verfahren werden in verschiedenen Disziplinen auf mögliche Stärken, Schwächen oder Anomalien getestet.

5.1.1. Funktionen

Es werden 4 Funktionen mit unterschiedlichen Eigenschaften getestet. Natürlich wäre das Newton-Verfahren nicht anwendbar auf eine nicht-differenzierbare Funktion und ebenso das Fixpunktverfahren nicht ohne Fixpunktgleichung mit $|\varphi'(\tilde{x})| < 1$, wie auch das Bisektionsverfahren nicht ohne Nullstelle zwischen dem Startintervall. Deshalb erfüllen hier alle drei Funktionen diese Kriterien. Die Startwerte wurden sinnvoll gewählt mit jeweils einem Abstand von weniger als einer halben Einheit von der Lösung.

1. Eine Polynomfunktion dritten Grades $f(x) = x^3 - 2x^2 - \frac{2}{3}x + 1$ mit drei reellen Nullstellen. Als Fixpunktgleichung wird $\varphi_f(x) = \sqrt{\frac{1}{2}x^3 - \frac{1}{3}x + \frac{1}{2}}$ verwendet.
2. Eine monoton steigende, nicht-lineare Exponentialfunktion $2^x + 3^x = 10 \Leftrightarrow g(x) = 2^x + 3^x - 10$ mit einer einfachen Nullstelle. Als Fixpunktgleichung wird $\varphi_g(x) = \log_3(10 - 2^x)$ verwendet.
3. Eine speziell zusammengestellte Funktion $h(x) = x^2 - \cos\left(|x|^x \sqrt[e]{e} - 1\right)$. Als Fixpunktgleichung wird $\varphi_h(x) = \sqrt{\cos\left(|x|^x \sqrt[e]{e} - 1\right)}$ verwendet

Diese drei Gleichungen sind in Abbildung 9 mit ihren Ableitungen dargestellt.

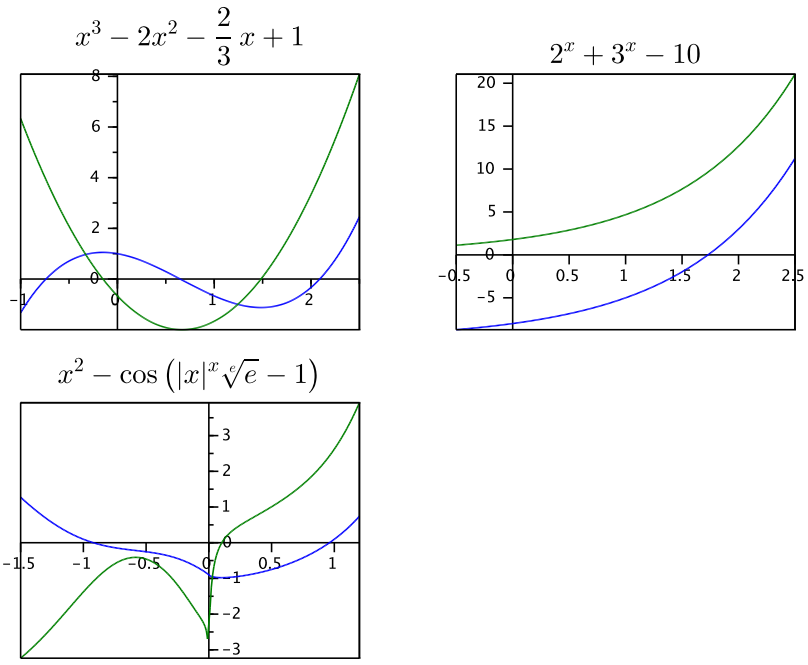

Abbildung 9: Testfunktionen: Die Funktionen sind in blau und deren Ableitungen in grün dargestellt.

5.1.2. Genauigkeit

Alle Funktionen werden mit verschiedenen gewünschten Genauigkeiten getestet und ermittelt, wie viele Iterationen benötigt werden um jene zu erreichen

- Kleine Genauigkeit von 4 Stellen $\epsilon = 0.5 \cdot 10^{-2^2}$
- Mittlere Genauigkeit von 16 Stellen $\epsilon = 0.5 \cdot 10^{-2^4}$
- Grosse Genauigkeit von 64 Stellen $\epsilon = 0.5 \cdot 10^{-2^6}$

5.2. Resultate

Es wurden die Anzahl Iterationen ermittelt, bis die gewünschte Genauigkeit erreicht ist. Mithilfe der Definition für die Konvergenzgeschwindigkeit

$$c = \lim_{n \to \infty} \frac{x_{n+1} - \tilde{x}}{|x_n - \tilde{x}|^p}$$

kann das zu einem konstanten Wert konvergierende c gefunden werden. p ist dabei jeweils die schon bekannte Ordnung des Verfahrens. Ein kleines Problem ist jedoch die exakte Lösung $\tilde{x}$, denn wir kennen diese eigentlich erst am Schluss. Indem man schon vor Beginn mithilfe des

schnell konvergierenden Newton-Verfahren eine sehr genaue Näherung an die exakte Lösung ermittelt hat, überwindet man diese Hürde. Die Resultate sind in Tabelle 4 aufgestellt.

Verfahren	Funktion	Stellen	Iterationen	c
Bisektionsverfahren	$f(x)$, $g(x)$, $h(x)$	4	15	≈ 0.5
		16	55	
		64	214	
Fixpunktverfahren	$f(x)$	4	8	$0.22896461\ldots$
		16	27	
		64	102	
	$g(x)$	4	10	$0.31294947\ldots$
		16	33	
		64	129	
	$h(x)$	4	7	$0.26497389\ldots$
		16	28	
		64	111	
Newtonverfahren	$f(x)$	4	2	$0.01801639\ldots$
		16	4	
		64	6	
	$g(x)$	4	3	$0.86632869\ldots$
		16	5	
		64	7	
	$h(x)$	4	4	$0.92739374\ldots$
		16	6	
		64	8	

Tabelle 4: Testresultate: Die Anzahl Iterationen hängt beim Bisektionsverfahren lediglich von der Startintervalllänge und der Genauigkeit ab. Weil alle Startintervalle die gleiche Länge haben, gibt es jeweils die gleiche Anzahl Iterationen bei jeder Funktion.

6. Zusammenfassung

6.1. Eigenschaften der Verfahren

Hier sind alle wichtigen Eigenschaften der Verfahren in einer Tabelle zusammengefasst.

Verfahren	Iterationsschritt	Konvergenzkriterium	p
Bisektion	$[a_{n+1}, b_{n+1}] = \begin{cases} [a_n, q_n] \\ [q_n, b_n] \end{cases}$	$f(a_0) \cdot f(b_0) < 0$	1
Fixpunkt	$q_{n+1} = \varphi(q_n)$	$\lvert\varphi'(\tilde{q})\rvert < 1$	1
Newton	$q_{n+1} = q_n - \frac{f(q_n)}{f'(q_n)}$	$\lvert q_0 - \tilde{q}\rvert < \frac{1}{c}$	2

6.2. Vor- und Nachteile

Wir haben gesehen, dass es verschiedenste Iterationsverfahren gibt mit unterschiedlichen Eigenschaften. Jedes hat seine eigenen Vor- und Nachteile, welche hier zusammengefasst sind:

- Bisektionsverfahren

  Vorteile:

  - Falls die Funktionswerte der Startwerte a_0 und b_0 unterschiedliche Vorzeichen habe, konvergiert dieses Verfahren mit Sicherheit zu einer Nullstelle zwischen diesen Startwerten.
  - Mit dem Abbruchkriterium $|b_n - a_n| < \epsilon$ ist die Abweichung zur wahren Nullstelle eingeschränkt, somit kann eine genau bestimmte Anzahl Nachkommastellen berechnet werden.

  Nachteile:

  - Da die Intervalllänge in jedem Iterationsschritt lediglich halbiert wird, kommt es zu einer sehr langsamen Konvergenzgeschwindigkeit.

- Fixpunktverfahren

  Vorteile:

  - Weil die nächste Näherung nur durch $q_{n+1} = \varphi(q_n)$ berechnet wird, hat das Fixpunktverfahren den kleinsten Rechenaufwand von allen hier vorgestellten Verfahren.

  Nachteile:

  - Bevor mit den Berechnungen begonnen werden kann, muss eine Fixpunktgleichung, dessen Betrag der Ableitung in der Umgebung der Nullstelle kleiner als eins ist, gefunden werden.
  - Die Konvergenzgeschwindigkeit ist nur linear und somit auch langsam. Allerdings kann eine schnellere Konvergenz als beim Bisektionsverfahren erreicht werden.

- Die Abweichung zur exakten Nullstelle kann nicht genau bestimmt werden und es muss auf ein anderes Abbruchkriterium zurückgegriffen werden. Dies führt dazu, dass nicht ausgesagt werden kann, auf wie viele Nachkommastellen die Näherung korrekt ist.

- Newton-Verfahren

  Vorteile:

  - Es ist das einzige hier vorgestellte Verfahren mit quadratischer Konvergenzgeschwindigkeit. Somit benötigt es viel weniger Iterationsschritte, um die gleiche Genauigkeit zu erreichen.

  Nachteile:

  - Da mit $q_{n+1} = q_n - \frac{f(q_n)}{f'(q_n)}$ die Ableitung der Funktion benötigt wird, muss sie einfach differenzierbar sein und es muss deren Ableitung bestimmt werden.
  - Die Abweichung zur exakten Lösung kann wie beim Fixpunktverfahren nicht genau bestimmt werden und es muss auf ein anderes Abbruchkriterium zurückgegriffen werden. Obwohl auch nicht ausgesagt werden kann, wie viele Nachkommastellen korrekt sind, ist dies meist irrelevant, da pro Iterationsschritt die Anzahl korrekter Stellen etwa verdoppelt wird.

6.3. Fazit

Aufgrund der sehr verschiedenen Eigenschaften der Verfahren, ist je nach Situation das eine oder das andere geeignet. Für schnelle Konvergenz mit Kenntnis über den ungefähren Aufenthaltsort einer Nullstelle und Differenzierbarkeit der Funktion ist das Newton-Verfahren geeignet, denn mit jeder Iteration werden die Anzahl korrekter Nachkommastellen etwa verdoppelt.

Ist der Verlauf der Funktion sehr unklar und nicht bekannt ist, wo sich eine Nullstelle aufhalten könnte, oder die Ableitung nicht bekannt ist, eignet sich das Bisektionsverfahren. Wenn man eine obere und eine untere Grenze angeben kann, konvergiert dieses Verfahren bestimmt gegen eine Nullstelle, was man von anderen Verfahren nicht erwarten kann. Nachteil ist, dass es sehr langsam konvergiert und mehr als 3 Schritte benötigt für eine weitere Nachkommastelle an Präzision. Um den Prozess zu beschleunigen, kann auch, sobald der ungefähre Aufenthaltsort einer Nullstelle gefunden wurde, das Newton-Verfahren angewendet werden.

Das Fixpunktverfahren eignet sich gut, wenn eine kontrahierende Fixpunktgleichung einfach gefunden werden kann. Es konvergiert dann meist schneller als das Bisektionsverfahren, aber langsamer als das Newton-Verfahren. Ausserdem ist es ein gut darzustellendes Verfahren.

Wegen der schnellen Konvergenzgeschwindigkeit des Newton-Verfahrens, wird dieses in den meisten Fällen verwendet, inklusive beim Taschenrechner.

7. Schlussbemerkungen

7.1. Eigenaufwand

7.1.1. LaTeX

Damit diese Arbeit gutaussehend gestaltet werden konnte, brachte ich mir LaTeX bei, welches sehr gut geeignet ist für wissenschaftliche Arbeiten mit einer grösseren Anzahl Formeln. Mithilfe der anfänglichen Hilfe von Frau Rupflin erlangte ich eine gute Startbasis, sie half mir den grundsätzlichen Aufbaus eines LaTeX-Dokumentes zu verstehen. Während des Schreibens dieser Arbeit kamen viele Fehler auf, welche jedoch alle mit relativ kleinem Aufwand gefunden und korrigiert werden konnten. Darstellungen zur Illustration der Verfahren und die Programmablaufpläne wurden alle selbst mit dem Package TikZ erstellt.

7.1.2. SciLab

Ein eindeutig schwierigerer Teil dieser Arbeit war das Einlernen in SciLab. Frau Rupflin leistete ebenfalls Hilfe dabei mit zwei guten Tutorials. Allerdings war mir SciLab vor dieser Arbeit unbekannt, aus diesem Grund ist der beiliegende Code nicht sehr professionell geschrieben und könnte besser implementiert werden. Der Code wurde so optimiert, dass durch Aufruf aller Programme, alle Grafiken und Tabellen von SciLab automatisch aktualisiert und in dem Verzeichnis, welches mit LaTeX verlinkt ist, gesichert werden.

7.1.3. Java

Die grösste Schwierigkeit war das Testen der Verfahren. Eine Genauigkeit von 64 Stellen soll erreicht werden, was mit normaler Rechnerarithmetik nicht möglich ist, denn deren präzisester Datentyp hat lediglich eine Genauigkeit von 15 Stellen. Meine erste Idee war es, die Klasse BigDecimal zu verwenden. Dies ist ein Datentyp, welcher Zahlen nicht binär absichert sondern dezimal. Ausserdem gibt es keine Begrenzung der Präzision. Die Klasse stellt primitive Operationen bereit wie Addition, Multiplikation, Modulo, etc. Als ich jedoch versuchte, die Terme $|x|^x$ und $\cos(x)$ zu berechnen, stiess ich an die Grenzen dieser Klasse.

Eine Lösung lieferte Apfloat (http://www.apfloat.org), eine Wrapperklasse für BigDecimal. Sie lieferte weitere grundsätzliche Operationen wie Potenzen, Logarithmen, trigonometrische Funktionen, etc. und ist einfacher zu benutzen als BigDecimal. Ausserdem verfügt Apfloat über eine Implementation der komplexen Zahlen, ich kann diese Klasse jedem empfehlen, der schnelle, hochpräzise Berechnungen durchführen muss.

Das Design der Struktur des Programms ist recht einfach: Drei Klassen für die drei zu testenden Funktionen mit jeweils der Ableitung und einer Fixpunktgleichung; weitere drei Klassen für die verschiedenen Verfahren. Sehr speziell ist die Klasse des Polynoms, denn sie berechnet jeweils ohne Eingabe deren eigene Ableitung. Ausserdem kann vorgegeben werden, nach welchem x im Polynom aufgelöst werden soll für die Fixpunktgleichung.

Jedes Verfahren hat jeweils Methoden für die Berechnung der nächsten Iteration, Methoden für die Berechnung der Variablen (wie z.B. q beim Bisektionsverfahren), Methoden zum Testen des Abbruchkriteriums, Methoden für die Ausgabe und noch einige weitere.

7.2. Ausblick

In dieser Arbeit war das zu besprechende Gebiet sehr eingeschränkt. Es wurde auf eine Gleichung mit nur einer Unbekannten vereinfacht. Iterative Lösungsverfahren lassen sich auch relativ einfach auf mehrdimensionale Gleichungssysteme erweitern. Das Newton-Verfahren kann noch erweitert werden auf mehrfache Nullstellen und, um zu vermeiden, dass immer die Gleiche Nullstelle gefunden wird, Nullstellenabspaltung[3].

Eine visuell sehr attraktive Weiterführung wäre die Arbeit mit komplexen Zahlen, denn numerische Verfahren sind auch auf diese anwendbar. Damit käme man schnell zu dem Newton-Fraktal.

Auch sind hier nur ein relativ kleiner Teil aller Verfahren vorgestellt, sehr interessant wäre die Auseinandersetzung mit komplizierteren Verfahren wie dem Halley-Verfahren. Oder sogar andere Arten von numerischen Verfahren wie solche für die Integration, Interpolation, normale oder partielle Differentialgleichungen.

Eine weitere Möglichkeit der Weiterführung, im Sinne der Programmierung mit Java, wäre die Entwicklung einer noch geeigneteren Struktur für die Verfahren und Funktionen. Durch Approximation der Ableitung an einem bestimmten Punkt könnte das Programm ohne die Ableitungsfunktion das Newtonverfahren durchführen. Polynome vom Grad 5 oder weniger könnten ihre eigenen exakten Lösungen finden (evt. sogar komplexe) und durch Nullstellenschranken geeignete Startwerte für die Verfahren bereitstellen. Sehr weit ausgeholt könnte die ganze Algebra programmiert werden, um eine Fixpunktgleichung durch Auflösen nach x zu finden.

[3] Eine grossartige Herleitung zur Abspaltung von Nullstellen beim Newton-Verfahren findet sich bei Kerscher M. (2013, 22. Apr). Nichtlineare Gleichungen, Nullstellensuche (S. 12). *Uni München*. Zugriff am 19. Jan 2014 unter http://www.mathematik.uni-muenchen.de/~kerscher/vorlesungen/numeriksose13/skript/nichtlinear.pdf

Literaturverzeichnis

Ackermann S. (ohne Datum). Übersicht zu Näherungsverfahren zur Nullstellenbestimmung. *Universität Leipzig*. Zugriff am 19. Jan 2014 unter http://www.math.uni-leipzig.de/MI/ackermann/pdf/Iter.pdf

Brand C. (2005, 23. Feb). Fixpunkt-Iteration. *Montanuniversität Leoben*. Zugriff am 19. Jan 2014 unter http://institute.unileoben.ac.at/amat/lehrbetrieb/num/vl-skript/skripts05/node30.html

Faires J. D. & Burden R. L. (1994). *Numerische Methoden* (S. 30-48). Heidelberg: Spektrum Akademischer Verlag.

Hundley D. R. (2006, 13. Sept). Notes: Rate of Convergence (S. 4). *Whitman College*. Zugriff am 19. Jan 2014 unter http://people.whitman.edu/~hundledr/courses/M467F06/ConvAndError.pdf

Kerscher M. (2013, 22. Apr). Nichtlineare Gleichungen, Nullstellensuche (S. 3). *Uni München*. Zugriff am 19. Jan 2014 unter http://www.mathematik.uni-muenchen.de/~kerscher/vorlesungen/numeriksose13/skript/nichtlinear.pdf

Kiani H. P. (2010). Banachscher Fixpunktsatz, Konditionszahlen (S. 2-4). *Mathematikwelt*. Zugriff am 19. Jan 2014 unter http://mathematikwelt.npage.de/get_file.php?id=19448411&vnr=841603

Oberle H. J. (2012). Das Newton-Verfahren (S. 47f). *Universität Hamburg*. Zugriff am 19. Jan 2014 unter http://www.math.uni-hamburg.de/home/oberle/skripte/optimierung/optim08.pdf

Richter T. & Wick T. (2012, 30. Okt). Nullstellenbestimmung (S. 25ff). *Universität Heidelberg*. Zugriff am 19. Jan 2014 unter http://numerik.uni-hd.de/~lehre/SS12/numerik0/2-nullstellen.pdf

Rogge L. (2009). Mittelwertsätze der Differentialrechnung mit Anwendung (S. 2). *Uni Duisburg-Essen*. Zugriff am 19. Jan 2014 unter https://www.uni-due.de/~hn213me/sk/rogge/Ana19.pdf

Tischel G. (1980). *Angewandte Mathematik* (S. 2-37). Frankfurt am Main: Verlag Moritz Diesterweg.

Trenchea C. (2009, 11. Dez). Rootfinding for Nonlinear Equations (S. 8). *University of Pittsburgh*. Zugriff am 19. Jan 2014 unter http://www.math.pitt.edu/~trenchea/math1070/MATH1070_5_Rootfinding.pdf

Wikipedia (Iteration, Numerische Mathematik, Intervallschachtelung, Fixpunktiteration, Newton-Verfahren, Sekantenverfahren, Halley-Verfahren, Mittelwertsatz der Differentialrechnung, Fixpunktsatz von Banach, Kontraktion, Konvergenzgeschwindigkeit). Zugriff am 19. Jan 2014 unter http://de.wikipedia.org

Abbildungsverzeichnis

Tabellenverzeichnis

A. Zusatz

A.1. Ein Beweis zur Konvergenzgeschwindigkeit einer Folge

Zu beweisen ist, dass die Folge $\left(\frac{1}{a}\right)^k$ $(k \in \mathbb{N}, |a| > 1)$ linear mit $c = \frac{1}{|a|}$ gegen null konvergiert.

Eine Folge $\{x_k\}$ konvergiert genau dann mit dem Faktor c zu ihrem Grenzwert x, wenn

$$|x_{k+1} - x| = c|x_k - x| \quad \text{für } 0 < c < 1$$

Der Grenzwert der Folge $\left(\frac{1}{a}\right)^k$ ist Null. Einsetzten der Glieder liefert

$$\left|\left(\frac{1}{a}\right)^{k+1}\right| = c\left|\left(\frac{1}{a}\right)^{k}\right|$$

Für den Fall, dass $a > 1$ ist, gilt

$$\begin{aligned}\left(\frac{1}{a}\right)^k \cdot \frac{1}{a} &= c \cdot \left(\frac{1}{a}\right)^k \\ c &= \frac{1}{a} = \frac{1}{|a|}\end{aligned}$$

Für den Fall, dass $a < -1$ ist, gilt

$$\begin{aligned}
\left(\frac{1}{a}\right)^{k+1} \cdot (-1)^{k+1} &= c \cdot \left(\frac{1}{a}\right)^{k} \cdot (-1)^{k} \\
\left(\frac{1}{a}\right)^{k} \cdot \frac{1}{a} \cdot (-1)^{k} \cdot (-1)^{1} &= c \cdot \left(\frac{1}{a}\right)^{k} \cdot (-1)^{k} \\
\frac{1}{a} \cdot (-1) &= c \\
c &= \frac{1}{-a} = \frac{1}{|a|}
\end{aligned}$$

□

A.2. Sekantenverfahren

A.2.1. Beschreibung

Ganz ähnlich wie das Bisektionsverfahren funktioniert auch das Sekantenverfahren. Durch zwei Punkte auf der Funktion wird eine Sekante gelegt, deren Schnittpunkt mit der Abszissenachse ist die neue Näherung. Nun wird mit den zwei neuesten Punkten fortgefahren (Abbildung 10).

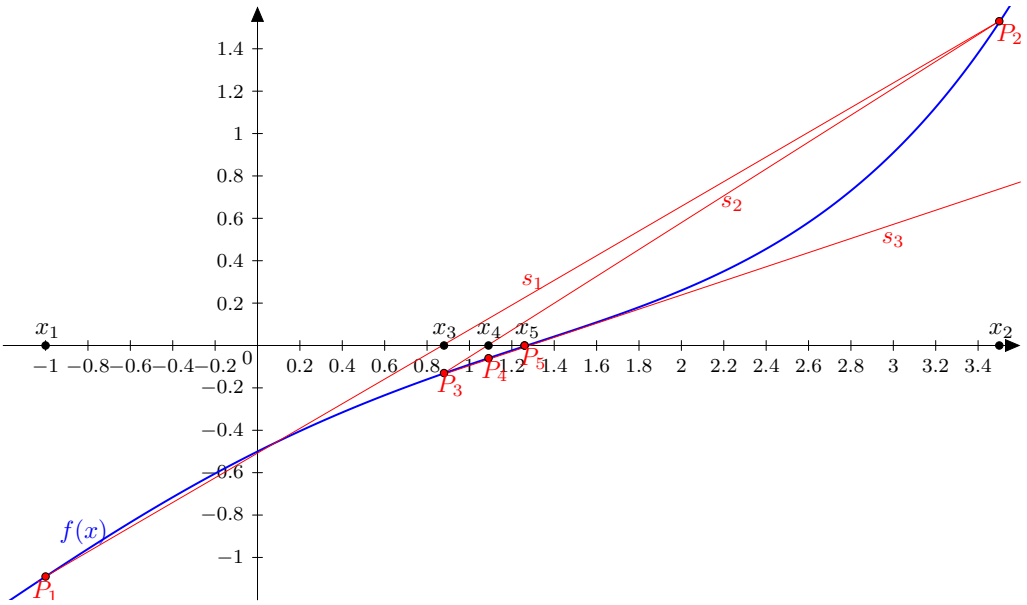

Abbildung 10: Illustration des Sekantenverfahrens mit der Funktion $f(x) = \frac{1}{100}x^4 - \frac{1}{10}x^2 + \frac{1}{2}x + 0.5$ mit den Startwerten $x_1 = -1$ und $x_2 = 3.5$. Zu beachten ist, dass die dritte Sekante von P_3 zu P_4 die Abszissenachse sehr Nahe beim Nullpunkt schneidet.

A.2.2. Umsetzung

Bekannt sind die beiden Startwerte x_1 und x_2. Die entsprechenden Punkte auf der Funktion sind $P_1(x_1, f(x_1))$ und $P_2(x_2, f(x_2))$. Eine Gerade $s_1(x)$ zwischen diesen zwei Punkten hat die Steigung

$$m = \frac{f(x_2) - f(x_1)}{x_2 - x_1}$$

Dann ist

$$s(x) = mx + q$$

Durch Einsetzen von P_1 erhält man

$$q = f(x_1) - mx_1$$

Dann hat die Gerade die Gleichung

$$s(x) = mx + f(x_1) - mx_1$$

Durch Nullsetzen und Auflösen erhält man

$$\begin{aligned} x_3 &= x_1 - \frac{f(x_1)}{m} \\ &= x_1 - \frac{x_2 - x_1}{f(x_2) - f(x_1)} \cdot f(x_1) \end{aligned}$$

Algorithmus Somit ist der allgemeine Algorithmus für das Sekantenverfahren:

$$x_{n+1} = x_{n-1} - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} \cdot f(x_{n-1})$$

A.2.3. Konvergenzgeschwindigkeit

Das Sekantenverfahren besitzt eine sehr aussergewöhnliche Konvergenzordnung $p = \phi = \frac{1+\sqrt{5}}{2}$ und damit eine superlineare Ordnung[4].

A.3. Banachscher Fixpunktsatz

Da hier nur eindimensionale Funktionen betrachtet werden, kann man auch den Banachschen Fixpunktsatz auf $\mathbb{R}^1$ vereinfachen.

Eine Funktion $\varphi : D \to \mathbb{R}$ heisst selbstabbildend, wenn jeder Wert des Definitionsbereich auf den Definitionsbereich abgebildet wird, also

$$\varphi(\mathrm{D}) \in \mathrm{D}$$

Eine Funktion $\varphi : [a, b] \to \mathbb{R}$ heisst kontrahierend, wenn

$$|\varphi(x) - \varphi(y)| \leq L \cdot |x - y| \quad \forall x, y \subset [a, b]$$

gilt, mit der Kontraktionszahl $\mathrm{L} < 1$. Das bedeutet, dass die Distanz zweier beliebiger Werte x, y im Intervall $[a, b]$ mit jedem Iterationsschritt kleiner wird und somit x und y sich von unten bzw. oben dem Grenzwert nähern. Falls φ eine einfach stetig differenzierbare Funktion ist, liefert der Mittelwertsatz der Differentialrechnung (Anhang A.4):

$$\begin{aligned} |\varphi(x) - \varphi(y)| &= |\varphi'(\alpha)||x - y| \quad \text{mit einem } \alpha \subset]x, y[\\ &\leq \max\{|\varphi'(x)| : x \subset [a, b]\} \cdot |x - y| \quad \forall x, y \subset [a, b] \end{aligned}$$

[4]Ein Beweis, dass für das Sekantenverfahren $p = \phi$ gilt findet man bei: Ackermann S. (ohne Datum). Übersicht zu Näherungsverfahren zur Nullstellenbestimmung (S. 1f). *Universität Leipzig*. Zugriff am 19. Jan 2014 unter http://www.math.uni-leipzig.de/MI/ackermann/pdf/Iter.pdf

Somit kann bei einer auf (a, b) einfach stetig differenzierbaren Funktion

$$L = \max \{|\varphi'(x)| : x \subset [a, b]\}$$

gewählt werden unter der Voraussetzung, dass $\max \{|\varphi'(x)| : x \subset [a, b]\} = L < 1$ gilt.

Satz 1 (Banachscher Fixpunktsatz)**.** *Sei* $\varphi : \mathrm{D} \to \mathrm{D}$ *kontrahierend auf* D. *Dann gibt es genau einen Fixpunkt* x^* *mit*

$$\varphi(x^*) = x^*$$

A.4. Mittelwertsatz der Differentialrechnung

Der Mittelwertsatz der Differentialrechnung sagt aus, dass es zu jedem Intervall $[a, b]$ der stetig differenzierbaren Funktion $f(x)$ ein α auf f gibt, bei dem die Steigung gleich der Steigung von Punkt $(a, f(a))$ zu Punkt $(b, f(b))$ ist

Satz 2 (Mittelwertsatz der Differentialrechnung)**.** *Sei* $f : [a, b] \to \mathbb{R}$ *stetig und in allen Punkten von* $]a, b[$ *differenzierbar. Dann gibt es ein* $\alpha \in]a, b[$ *mit*

$$f'(\alpha) = \frac{f(b) - f(a)}{b - a}$$

B. Kompletter SciLab-Code

B.1. Hinweise

Es wurde bei allen Programmen, die ein Unterprogramm aufrufen, der jeweilige Pfad mit „...“ ersetzt. Auch wurden die Befehle für die Generierungen der PDF-Dateien kommentiert. Jeweils steht der Name der Datei über dem Code. Programme, die nur andere Programme aufrufen, sind hier nicht vorhanden.

B.2. Code

Funktion1.sce

```
function [y]=f(x)
    y=x^3-2*x^2-2/3*x+1
endfunction

function [a, b, latex, undef]=F()
    a=-1
    b=2.5
    latex="$x^3-2x^2-\frac{2}{3}x+1$"
    undef=[]
endfunction

function [y]=f_Abl(x)
    y=3*x^2-4*x-2/3
endfunction

function [undef]=F_Abl()
    undef=[]
endfunction
```

Funktion2.sce

```
function [y]=f(x)
    y=2^x+3^x-10
endfunction

function [a, b, latex, undef]=F()
    a=-0.5
    b=2.5
    latex="$2^x+3^x-10$"
    undef=[]
endfunction

function [y]=f_Abl(x)
    y=log(2)*exp(x*log(2))+log(3)*exp(x*log(3))
endfunction

function [undef]=F_Abl()
```

```
17      undef=[]
18  endfunction
```

Funktion3.sce

```
1   function [y]=f(x)
2       y=x^2-(cos(exp(x.*log(abs(x))).*nthroot(%e, %e)-1))
3   endfunction
4
5   function [a, b, latex, undef]=F()
6       a=-1.5
7       b=1.2
8       latex="$x^2-cos(|x|^x\sqrt[e]{e}-1)$"
9       undef=[0]
10  endfunction
11
12  function [y]=f_Abl(x)
13      y=2*x+sin(exp(x.*log(abs(x))).*nthroot(%e, %e)-1).*nthroot(%e, %e).*exp
            (x.*log(abs(x))).*(log(abs(x))+1)
14  endfunction
15
16  function [undef]=F_Abl()
17      undef=0
18  endfunction
```

PlotFunktionUndAbleitung.sce

```
1   // F() gibt die Randstellen des Intervalls für die Bisektion, den LaTeX-
        Code und die undefinierten Werte der Funktion zurück
2   [a, b, latex, undef]=F();
3   x=a:0.01:b;
4
5   // Löscht alle undefinierten Werte aus x heraus
6   for i = 1:length(undef)
7       column=find(x==undef(i));
8       x(:, column)=[];
9   end
10
11  // f() und f_Abl() müssen schon vorher definiert sein
12  plot(x, f(x), x, f_Abl(x));
13  xtitle(latex);
14  a=get('current_axes');
15  a.x_location="origin";
16  a.y_location="origin";
17  a.title.font_size=4;
18  r=xstringl(0, 0, a.title.text, 6, 4);
19  b=a.data_bounds;
20  a.title.position=[(b(1)+b(2))/2-r(3)/2 b(4)];
```

TestfunktionenPlot.sce

```
1   // ... ist der Pfad zu diesem File
2   subplot(221);
```

```
exec('.../Funktion1.sce');
exec('.../PlotFunktionUndAbleitung.sce');

subplot(222);
exec('.../Funktion2.sce');
exec('.../PlotFunktionUndAbleitung.sce');

subplot(223);
exec('.../Funktion3.sce');
exec('.../PlotFunktionUndAbleitung.sce');
```

FpEx_1.sce

```
function [q]=FpEx_1(q, stellen, zSt)

// Standardwerte setzen, falls Argumente inexistent sind
if ~exists('q', 'local')
    q = 1.5 // Startwert q_0
end
if ~exists('stellen', 'local')
    stellen = 5 // Anzahl Nachkommastellen (Min: 0, Max: 15)
end
if ~exists('zSt', 'local')
    zSt=2 // Anzahl zusätzlich zu zeigenden Stellen
end


function [y]=fkt(x)
    y=nthroot(2*x^2-1, 3)
endfunction

// Deklarationen und Initialisierungen
eps=.5*10^(-stellen) // Abweichung
m=10 // Anzahl Schritte (Fixpunktgleichung ist divergent)
n=1
zStr=max(length(floor([q, q]))) + 1
qn=q

// Steuerung der Konsolen-Ausgabe
formatS=sprintf(' %%%is', stellen+zStr)
formatS=sprintf('%%%2s%s%s', formatS, formatS)
formatF=sprintf(' %%%i.%if', stellen+zStr, stellen+zSt)
formatF=sprintf('%%%2i%s%s', formatF, formatF)
mprintf(formatS + '\n', 'n', 'q_n', 'q_n-q_{n-1}}')

// Steuerung der .tex Datei-Ausgabe
formatFF=sprintf(' & %%.%if', stellen+zSt)
formatFF=sprintf('%%%2i%s%s', formatFF, formatFF)
fd=mopen('.../LatexInputs/Fixpunkt_Beispiel2_Tabelle.tex', 'w') // Pfad;
    nur schreiben (w) = write
mfprintf(fd, '\\begin{displaymath}\n')
mfprintf(fd, '\t\\begin{array}{|r|r|r|}\n')
mfprintf(fd, '\t\t\\hline\n')
```

```
mfprintf(fd, '\t\tn & q_n & |q_n-q_{n-1}|\\\\\n')
mfprintf(fd, '\t\t\\hline\n')


// Main Loop
while n <= m,
    qn=fkt(q)
    mprintf(formatF + '\n', n, qn, qn-q)
    mfprintf(fd, '\t\t' + formatFF + '\\\\\n', n, qn, qn-q)
    n=n+1
    q=qn
end


// Fertigstellung der .tex Datei-Ausgabe
mfprintf(fd, '\t\t\\hline\n')
mfprintf(fd, '\t\\end{array}\n')
mfprintf(fd, '\\end{displaymath}')
mclose(fd)

endfunction
```

Newton_Beispiel_1.sce

```
function [q]=NtEx(q, stellen, zSt)
if ~exists('q', 'local')
    q = 2 // Startwert q_0
end
if ~exists('stellen', 'local')
    stellen = 14 // Anzahl Nachkommastellen (Min: 0, Max: 15)
end
if ~exists('zSt', 'local')
    zSt=2 // Anzahl zusätzlich zu zeigenden Stellen
end


function [y]=fkt(x)
    y=x^3-2*x^2+1
endfunction

function [y]=fkt_Abl(x)
    y=3*x^2-4*x
endfunction


// Deklarationen und Initialisierungen
f=fkt(q) // Funktionswert
eps=.5*10^(-stellen) // Abweichung
n=0 // Anzahl benötigter Schritte

zStr=max(length(floor([q, f]))) + 1


```

```
30 // Steuerung der Konsolen-Ausgabe
31 formatS=sprintf(' %%%is', stellen+zStr)
32 formatS=sprintf('%%2s%s%s', formatS, formatS)
33 formatF=sprintf(' %%%i.%if', stellen+zStr, stellen+zSt)
34 formatF=sprintf('%%2i%s%s', formatF, formatF)
35 mprintf(formatS + '\n', 'n', 'q_n', 'f(q_n)')
36 mprintf(formatF + '\n', n, q, f)
37
38 // Steuerung der .tex Datei-Ausgabe
39 formatFF=sprintf(' & %%.%if', stellen+zSt)
40 formatFF=sprintf('%%2i%s%s', formatFF, formatFF)
41 fd=mopen('.../LatexInputs/Newton_Beispiel_Tabelle.tex', 'w') // Pfad; nur
     schreiben (w) = write
42 mfprintf(fd, '\\begin{displaymath}\n')
43 mfprintf(fd, '\t\\begin{array}{|r|r|r|}\n')
44 mfprintf(fd, '\t\t\\hline\n')
45 mfprintf(fd, '\t\tn & q_n & f(q_n)\\\\\n')
46 mfprintf(fd, '\t\t\\hline\n')
47 mfprintf(fd, '\t\t' + formatFF + '\\\\\n', n, q, f)
48
49
50 // Main Loop
51 while abs(f) >= eps,
52     q=q-f/fkt_Abl(q)
53     f=fkt(q)
54     n=n+1
55     mprintf(formatF + '\n', n, q, f)
56     mfprintf(fd, '\t\t' + formatFF + '\\\\\n', n, q, f)
57 end
58
59 // Fertigstellung der .tex Datei-Ausgabe
60 mfprintf(fd, '\t\t\\hline\n')
61 mfprintf(fd, '\t\\end{array}\n')
62 mfprintf(fd, '\\end{displaymath}')
63 mclose(fd)
64
65 endfunction
```

Newton_Beispiel_2.sce

```
1  function [q]=NtEx(q, stellen, zSt)
2  if ~exists('q', 'local')
3      q = 2 // Startwert q_0
4  end
5  if ~exists('stellen', 'local')
6      stellen = 5 // Anzahl Nachkommastellen (Min: 0, Max: 15)
7  end
8  if ~exists('zSt', 'local')
9      zSt=2 // Anzahl zusätzlich zu zeigenden Stellen
10 end
11
12
13 function [y]=fkt(x)
```

```
14      y=x^3-2*x^2+1
15  endfunction
16
17  function [y]=fkt_Abl(x)
18      y=3*x^2-4*x
19  endfunction
20
21
22  // Deklarationen und Initialisierungen
23  qn=q+1
24  f=fkt(q) // Funktionswert
25  eps=.5*10^(-stellen) // Abweichung
26  n=0 // Anzahl benötigter Schritte
27
28  zStr=max(length(floor([q, f]))) + 1
29
30  // Steuerung der Konsolen-Ausgabe
31  formatS=sprintf(' %%%is', stellen+zStr)
32  formatS=sprintf('%%2s%s%s', formatS, formatS)
33  formatF=sprintf(' %%%i.%if', stellen+zStr, stellen+zSt)
34  formatF=sprintf('%%2i%s%s', formatF, formatF)
35  mprintf(formatS + '\n', 'n', 'q_n', 'f(q_n)')
36  mprintf(formatF + '\n', n, q, f)
37
38  // Steuerung der .tex Datei-Ausgabe
39  formatFF=sprintf(' & %%.%if', stellen+zSt)
40  formatFF=sprintf('%%2i%s%s', formatFF, formatFF)
41  fd=mopen('.../LatexInputs/Newton_Beispiel_Tabelle.tex', 'w') // Pfad; nur
        schreiben (w) = write
42  mfprintf(fd, '\\begin{displaymath}\n')
43  mfprintf(fd, '\t\\begin{array}{|r|r|r|}\n')
44  mfprintf(fd, '\t\t\\hline\n')
45  mfprintf(fd, '\t\tn & q_n & f(q_n)\\\\\n')
46  mfprintf(fd, '\t\t\\hline\n')
47  mfprintf(fd, '\t\t' + formatFF + '\\\\\n', n, q, f)
48
49
50  // Main Loop
51  while abs(qn-q) >= eps,
52      q=qn
53      qn=q-f/fkt_Abl(q)
54      f=fkt(qn)
55      n=n+1
56      mprintf(formatF + '\n', n, qn, f)
57      mfprintf(fd, '\t\t' + formatFF + '\\\\\n', n, qn, f)
58  end
59
60
61  // Fertigstellung der .tex Datei-Ausgabe
62  mfprintf(fd, '\t\t\\hline\n')
63  mfprintf(fd, '\t\\end{array}\n')
64  mfprintf(fd, '\\end{displaymath}')
65  mclose(fd)
```

```
66
67 endfunction
```

Phi1.sce

```
1  // Zeichnet die Fixpunktgleichungen
2  s=-2:0.01:3;
3  t=-1:0.01:3;
4  x=nthroot(2*s^2-1, 3);
5  y=sqrt((t^3+1)/2);
6  z=-sqrt((t^3+1)/2);
7  plot(s, s, s, x, t, y, t, z);
8  a=get('current_axes');
9  a.x_location="origin";
10 a.y_location="origin";
11 a.data_bounds=[-1.5, -1.2; 2.2, 1.7];
12 lg=legend(['$f(x)=x$' '$\varphi_1(x)=\sqrt[3]{2x^2-1}$' '$\varphi_2(x)=\
       sqrt{\frac{x^3+1}{2}}$' '$\varphi_3(x)=-\sqrt{\frac{x^3+1}{2}}$'], 4);
13 lg.font_size=4;
14 // xs2pdf(0, '/LatexInputs/Phi');
```

PlotFunktion.sce

```
1  [a, b, latex, undef]=F();
2  x=a:0.01:b;
3  for i = 1:length(undef)
4      column=find(x==undef(i));
5      x(:, column)=[];
6  end
7  plot(x, f(x));
8  xtitle(latex);
9  a=get('current_axes');
10 a.x_location="origin";
11 a.y_location="origin";
12 a.title.font_size=3;
13 r=xstringl(0, 0, a.title.text, 6, 3);
14 b=a.data_bounds;
15 a.title.position=[(b(1)+b(2))/2-r(3)/2 b(4)];
```

Funktion0.sce

```
1  // Beispielfunktion
2  function [y]=f(x)
3      y=x^3-2*x^2+1
4  endfunction
5
6  function [a, b, latex, undef]=F()
7      a=-1
8      b=2
9      latex="$f(x)=x^3-2x^2+1$"
10     undef=[]
11 endfunction
12
```

```
13 function [y]=f_Abl(x)
14     y=3*x^2-4*x
15 endfunction
16
17 function [undef]=F_Abl()
18     undef=[]
19 endfunction
```

Bisektion_Beispiel_Grafik.sce

```
1 x=0:0.01:2
2 y=x^3-2*x^2+1
3 plot(x, y)
4 a=gca()
5 a.x_location="origin"
6 // xs2pdf(0, '/LatexInputs/BeispielFunktion')
```

Bisektion_Beispiel.sce

```
1  function [q]=BsEx(a, b, stellen, zSt)
2
3  if ~exists('a', 'local')
4      a = 1.5 // Startwert a_0
5  end
6  if ~exists('b', 'local')
7      b = 2 // Startwert b_0
8  end
9  if ~exists('stellen', 'local')
10     stellen = 2 // Anzahl Nachkommastellen (Min: 0, Max: 15)
11 end
12 if ~exists('zSt', 'local')
13     zSt=2 // Anzahl zusätzlich zu zeigenden Stellen
14 end
15
16
17 function [y]=fkt(x)
18     y=x^3-2*x^2+1
19 endfunction
20
21
22 // Deklarationen und Initialisierungen
23 d=b-a // Intervallgrösse
24 q=(a+b)/2 // Startwert q_0
25 f=fkt(q)
26 eps=.5*10^(-stellen) // Abweichung
27 n=0 // Anzahl benötigter Schritte
28
29 zStr=max(length(floor([a, b, d, q, f]))) + 1
30
31 // Steuerung der Konsolen-Ausgabe
32 formatS=sprintf(' %%%is', stellen+zStr)
33 formatS=sprintf('%%2s%s%s%s%s%s', formatS, formatS, formatS, formatS,
       formatS)
```

```
34 formatF=sprintf(' %%%i.%if', stellen+zStr, stellen+zSt)
35 formatF=sprintf('%%2i%s%s%s%s%s', formatF, formatF, formatF, formatF,
       formatF)
36 mprintf(formatS + '\n', 'n', 'a_n', 'b_n', 'd_n', 'q_n', 'f(q_n)')
37 mprintf(formatF + '\n', n, a, b, d, q, f)
38
39 // Steuerung der .tex Datei-Ausgabe
40 formatFF=sprintf(' & %%.%if', stellen+zSt)
41 formatFF=sprintf('%%2i%s%s%s%s%s', formatFF, formatFF, formatFF, formatFF,
       formatFF)
42 fd=mopen('.../LatexInputs/Bisektion_Beispiel_Tabelle.tex', 'w') // Pfad;
       nur schreiben (w) = write
43 mfprintf(fd, '\\begin{displaymath}\n')
44 mfprintf(fd, '\t\\begin{array}{|r|r|r|r|r|}\n')
45 mfprintf(fd, '\t\t\\hline\n')
46 mfprintf(fd, '\t\tn & a_n & b_n & b_n - a_n & q_n & f(q_n)\\\\\n')
47 mfprintf(fd, '\t\t\\hline\n')
48 mfprintf(fd, '\t\t' + formatFF + '\\\\\n', n, a, b, d, q, f)
49
50
51 // Main Loop
52 while d >= eps,
53     if sign(f) == sign(fkt(a)) then
54         a = q
55     else
56         b = q
57     end
58     d = b - a
59     q = (a + b) / 2
60     f=fkt(q)
61     n=n+1
62     mprintf(formatF + '\n', n, a, b, d, q, f) // Konsolen-Ausgabe
63     mfprintf(fd, '\t\t' + formatFF + '\\\\\n', n, a, b, d, q, f)
64 end
65
66
67 // Fertigstellung der .tex Datei-Ausgabe
68 mfprintf(fd, '\t\t\\hline\n')
69 mfprintf(fd, '\t\\end{array}\n')
70 mfprintf(fd, '\\end{displaymath}')
71 mclose(fd)
72
73 endfunction
```

C. Kompletter Java-Code

C.1. Hinweise

Namen der Dateien wurden nicht über dem Code hinterlegt, da sie jeweils dem Namen der Klasse entsprechen. Für die Benutzung der Klasse Apfloat muss sie von http://www.apfloat.org heruntergeladen werden und mit dem Kompilierer verknüpft werden. Das Interface Verfahren ist zwar vorhanden, jedoch wurde es nicht verwendet. Die Methode main aus der Klasse LatexInputs generiert den Output. Die Lösungen der Funktionen werden in dieser Methode auf 256 Stellen initialisiert, um jedoch Platz zu sparen wurde nach einigen Stellen „... “ eingesetzt. Die Klasse BigDecimal wurde teils für den Output verwendet, da sie mit dem String-Formatierer besser funktioniert. Da Objekte der Klasse Apfloat unveränderlich sind, werden bei der Initialisierung einer Funktion die benötigten Apfloat-Konstanten abgespeichert, sodass nicht bei jeder Funktionsauswertung neue, gleiche Objekte erstellt werden.

C.2. Code

```
1  package MainPackage;
2
3  import Funktionen.Exponentialfunktion;
4  import Funktionen.Funktion;
5  import Funktionen.Polynomfunktion;
6  import Funktionen.SpezielleFunktion;
7
8  import org.apfloat.Apfloat;
9
10 public class LatexInputs {
11  public static void main(String[] args) {
12   int Dig, MoreDig = 2, DispDig, CalcDig;
13
14   for (int n = 0; n < 3; n++) {
15    Dig = (int) Math.pow(4, n + 1);
16    DispDig = Dig + MoreDig;
17    CalcDig = DispDig + 4;
18
19    // ////////////////////////////////////
20
21    Funktion[] f = new Funktion[3];
22
23    f[0] = new Polynomfunktion(CalcDig, 2, new Apfloat("1", CalcDig),
24      new Apfloat("-2", CalcDig),
25      new Apfloat("-2", CalcDig)
26        .divide(new Apfloat("3", CalcDig)), new Apfloat(
27        "1", CalcDig));
28    f[0].setIntervall(0, 1);
29    f[0].setX_Newton(0.5);
30    f[0].setX_Fixpunkt(1.0);
31    f[0].setSolve(new Apfloat(
```

```
32        "0.648144971184663211841279937292311 4...") );
33
34      f[1] = new Exponentialfunktion(CalcDig);
35      f[1].setIntervall(1, 2);
36      f[1].setX_Newton(1.5);
37      f[1].setX_Fixpunkt(1.5);
38      f[1].setSolve(new Apfloat(
39        "1.729255589818595724738162714273986...") );
40
41      f[2] = new SpezielleFunktion(CalcDig);
42      f[2].setIntervall(0.1, 1.1);
43      f[2].setX_Newton(0.5);
44      f[2].setX_Fixpunkt(0.5);
45      f[2].setSolve(new Apfloat(
46        "0.961495574785851654191919587467357 8...") );
47
48      // ////////////////////////////////////////
49
50      for (int k = 0; k < 1; k++) {
51       System.out.println("Bisektion, Funktion f(x) = "
52         + f[k].toString() + ", Stellen = " + Dig);
53       Bisektionverfahren Bs = new Bisektionverfahren(false, MoreDig);
54       Bs.run(f[k], Dig);
55
56       System.out.println("Newtonverfahren, Funktion f(x) = " +
57       f[k].toString()
58       + ", Stellen = " + Dig);
59       Newtonverfahren Nt = new Newtonverfahren(true, MoreDig);
60       Nt.run(f[k], Dig);
61
62       System.out.println("Fixpunktverfahren, Funktion f(x) = " +
63       f[k].toString()
64       + ", Stellen = " + Dig);
65       Fixpunktverfahren Fp = new Fixpunktverfahren(true, MoreDig);
66       Fp.run(f[k], Dig);
67      }
68    }
69  }
70 }
```

```
1  package MainPackage;
2
3  public interface Verfahren {
4   int n = 0;
5
6   public void next();
7   public void print();
8   public boolean abbruch();
9  }
```

```
1  package MainPackage;
2
3  import java.math.BigDecimal;
```

```
4  import java.text.DecimalFormat;
5
6  import Funktionen.Funktion;
7
8  import org.apfloat.Apfloat;
9  import org.apfloat.ApfloatMath;
10
11 public class Bisektionverfahren {
12  int N = 0, Dig, MoreDigs, DispDig;
13  Apfloat A, B, D, X, F, E;
14  Apfloat Fehler, FehlerAlt, C;
15  Apfloat[] CA = new Apfloat[10000];
16  Funktion f;
17
18  boolean showSteps;
19  Apfloat[] NUM;
20
21  DecimalFormat Format;
22  String OUT;
23
24  public Bisektionverfahren(boolean showSteps, int MoreDigs) {
25   this.showSteps = showSteps;
26   this.MoreDigs = MoreDigs;
27   initKonst();
28  }
29
30  public void initKonst() {
31   NUM = new Apfloat[11];
32   for (Integer k = 0; k < NUM.length; k++)
33    NUM[k] = new Apfloat(k.toString(), Apfloat.INFINITE);
34  }
35
36  public Apfloat f(Apfloat x) {
37   return f.f(x);
38  }
39
40  public void next() {
41   if (f(A).signum() * F.signum() == -1)
42    B = X;
43   else
44    A = X;
45   calc();
46   N++;
47  }
48
49  public void initOutputFormat() {
50   StringBuilder str = new StringBuilder("0.");
51   for (int k = 0; k < DispDig; k++)
52    str.append("0");
53   Format = new DecimalFormat(str.toString());
54
55   int ABXLength = Math.max(
56     Math.max(Format.format(A).length(), Format.format(B).length()),
```

```
57      Format.format(X).length());
58    int DFLength = Math.max(Format.format(D).length(), Format.format(F)
59      .length()) + 4;
60
61    OUT = String
62      .format("N = %%4d, X = %%%ds, C = %%%ds, A = %%%ds, D = %%%ds, B = %%%
            ds, F = %%%ds%%n",
63          ABXLength, ABXLength, ABXLength, DFLength, ABXLength,
64          DFLength);
65  }
66
67  public String toFormat(Apfloat XX) {
68    return Format.format(new BigDecimal(XX.toString(true)));
69  }
70
71  public void print() {
72    System.out.printf(OUT, N, toFormat(X), toFormat(C), toFormat(A),
73      toFormat(D), toFormat(B), toFormat(F));
74    return;
75  }
76
77  public boolean abbruch() {
78    return D.compareTo(E) == -1 || F.signum() == 0;
79  }
80
81  public void calc() {
82    X = A.add(B).divide(NUM[2]);
83    D = B.subtract(A);
84    F = f(X);
85    FehlerAlt = Fehler;
86    Fehler = ApfloatMath.abs(X.subtract(f.solve())).divide(f.solve());
87    if (null == FehlerAlt)
88      C = Apfloat.ZERO;
89    else {
90      if (FehlerAlt.compareTo(Apfloat.ZERO) == 0)
91        C = new Apfloat("-1");
92      else
93        C = Fehler.divide(ApfloatMath.pow(FehlerAlt, 1));
94      CA[N] = C;
95    }
96  }
97
98  public void run(Funktion f, int Dig) {
99    this.Dig = Dig;
100   DispDig = this.Dig + MoreDigs;
101   this.E = ApfloatMath.scale(new Apfloat("0.5", Apfloat.INFINITE),
102     (long) -1 * this.Dig);
103   this.f = f;
104   this.A = f.getA();
105   this.B = f.getB();
106
107   calc();
108
```

```
109    initOutputFormat();
110
111    iterate();
112
113    Apfloat avg = Apfloat.ZERO;
114    Integer n = 1;
115    while (CA[n] != null) {
116     avg = avg.add(CA[n]);
117     n++;
118    }
119    avg = avg.divide(new Apfloat(n.toString(), Apfloat.INFINITE));
120    System.out.println(avg);
121   }
122
123   public void iterate() {
124    if (f(A).signum() * f(B).signum() == 1)
125     System.out
126       .println("Keine garantierte Nullstelle in dem Intervall!");
127    else if (f(A).signum() == 0)
128     System.out.println("Nullstelle bei A = " + A);
129    else if (f(B).signum() == 0)
130     System.out.println("Nullstelle bei B = " + B);
131    else {
132     if (showSteps)
133      while (!abbruch()) {
134       print();
135       next();
136      }
137     else
138      while (!abbruch())
139       next();
140     print();
141    }
142
143    System.out.println();
144   }
145  }
```

```
1  package MainPackage;
2  import java.math.BigDecimal;
3  import java.text.DecimalFormat;
4
5  import Funktionen.Funktion;
6
7  import org.apfloat.Apfloat;
8  import org.apfloat.ApfloatMath;
9
10
11
12 public class Fixpunktverfahren implements Verfahren {
13  int N = 0, Dig, MoreDigs, DispDig, CalcDig;
14  Apfloat X, F, E;
15  Funktion f;
```

```
16  Apfloat Fehler, FehlerAlt, C;
17  Apfloat[] CA = new Apfloat[10000];
18
19  boolean showSteps;
20  Apfloat[] NUM;
21
22  DecimalFormat Format;
23  String OUT;
24
25  public Fixpunktverfahren(boolean showSteps, int MoreDigs) {
26   this.showSteps = showSteps;
27   this.MoreDigs = MoreDigs;
28   initKonst();
29  }
30
31  public void initKonst() {
32   NUM = new Apfloat[11];
33   for (Integer k = 0; k < NUM.length; k++)
34    NUM[k] = new Apfloat(k.toString(), Apfloat.INFINITE);
35  }
36
37  public Apfloat f(Apfloat x) {
38   return f.f(x);
39  }
40
41  public Apfloat phi(Apfloat x) {
42   return f.f_Fixpunkt(x);
43  }
44
45  @Override
46  public void next() {
47   X = phi(X);
48   calc();
49   N++;
50  }
51
52  public void initOutputFormat() {
53   StringBuilder str = new StringBuilder("0.");
54   for (int k = 0; k < DispDig; k++)
55    str.append("0");
56   Format = new DecimalFormat(str.toString());
57
58   int XLength = Format.format(X).length();
59   int FLength = Format.format(F).length() + 4;
60
61   OUT = String
62     .format("N = %%4d, X = %%%ds, C = %%%ds, F = %%%ds%%n",
63       XLength, XLength, FLength, FLength);
64  }
65
66  public String toFormat(Apfloat XX) {
67   return Format.format(new BigDecimal(XX.toString(true)));
68  }
```

```
69
70  public void print() {
71   System.out.printf(OUT, N, toFormat(X), toFormat(C), toFormat(F));
72   return;
73  }
74
75  @Override
76  public boolean abbruch() {
77   if (F.signum() == 0 || ApfloatMath.abs(F).compareTo(E) == -1)
78    return true;
79   return false;
80  }
81
82  public void calc() {
83   F = f(X);
84   FehlerAlt = Fehler;
85   Fehler = ApfloatMath.abs(X.subtract(f.solve())).divide(f.solve()).
         precision(CalcDig);
86   if (null == FehlerAlt)
87    C = Apfloat.ZERO;
88   else {
89    C = Fehler.divide(ApfloatMath.pow(FehlerAlt, 1));
90    CA[N] = C;
91   }
92  }
93
94  public void run(Funktion f, int Dig) {
95   this.Dig = Dig;
96   DispDig = this.Dig + MoreDigs;
97   CalcDig = DispDig + 2;
98   this.E = ApfloatMath.scale(new Apfloat("0.5", Apfloat.INFINITE),
99     (long) -1 * this.Dig);
100  this.f = f;
101  this.X = f.getX_Fixpunkt();
102
103  calc();
104
105  initOutputFormat();
106
107  iterate();
108 }
109
110 public void iterate() {
111  if (showSteps)
112   while (!abbruch()) {
113    print();
114    next();
115   }
116  else
117   while (!abbruch())
118    next();
119  print();
120  System.out.println();
```

```
121   }
122
123 }
```

```
1  package MainPackage;
2  import java.math.BigDecimal;
3  import java.text.DecimalFormat;
4
5  import Funktionen.Funktion;
6
7  import org.apfloat.Apfloat;
8  import org.apfloat.ApfloatMath;
9
10
11
12 public class Newtonverfahren implements Verfahren {
13  int N = 0, Dig, MoreDigs, DispDig, CalcDig;
14  Apfloat X, F, fAbl, E;
15  Apfloat Fehler, FehlerAlt, C;
16  Apfloat[] CA = new Apfloat[10000];
17  Funktion f;
18
19  boolean showSteps;
20  Apfloat[] NUM;
21
22  DecimalFormat Format;
23  String OUT;
24
25  public Newtonverfahren(boolean showSteps, int MoreDigs) {
26   this.showSteps = showSteps;
27   this.MoreDigs = MoreDigs;
28   initKonst();
29  }
30
31  public void initKonst() {
32   NUM = new Apfloat[11];
33   for (Integer k = 0; k < NUM.length; k++)
34    NUM[k] = new Apfloat(k.toString(), Apfloat.INFINITE);
35  }
36
37  public Apfloat f(Apfloat x) {
38   return f.f(x);
39  }
40
41  public Apfloat fAbl(Apfloat x) {
42   return f.f_Abl(x);
43  }
44
45  @Override
46  public void next() {
47   X = X.subtract(F.divide(fAbl));
48   calc();
49   N++;
```

```
50  }
51
52  public void initOutputFormat() {
53   StringBuilder str = new StringBuilder("0.");
54   for (int k = 0; k < DispDig; k++)
55    str.append("0");
56   Format = new DecimalFormat(str.toString());
57
58   int XLength = Format.format(X).length();
59   int FLength = Format.format(F).length() + 4;
60
61   OUT = String
62     .format("N = %%4d, X = %%%ds, C = %%%ds, F = %%%ds, fAbl = %%%ds%%n",
63       XLength, XLength, FLength, FLength);
64  }
65
66  public String toFormat(Apfloat XX) {
67   return Format.format(new BigDecimal(XX.toString(true)));
68  }
69
70  public void print() {
71   System.out.printf(OUT, N, toFormat(X), toFormat(C), toFormat(F), toFormat
         (fAbl));
72   return;
73  }
74
75  @Override
76  public boolean abbruch() {
77   if (F.signum() == 0 || ApfloatMath.abs(F).compareTo(E) == -1) {
78    return true;
79   } else if (fAbl.signum() == 0) {
80    System.out.println("Division durch Null!");
81    return true;
82   }
83   return false;
84  }
85
86  public void calc() {
87   F = f(X);
88   fAbl = fAbl(X);
89   FehlerAlt = Fehler;
90   Fehler = ApfloatMath.abs(X.subtract(f.solve())).divide(f.solve()).
         precision(CalcDig);
91   if (null == FehlerAlt)
92    C = Apfloat.ZERO;
93   else {
94    C = Fehler.divide(ApfloatMath.pow(FehlerAlt, 2));
95    CA[N] = C;
96   }
97  }
98
99  public void run(Funktion f, int Dig) {
100  this.Dig = Dig;
```

```
101     DispDig = this.Dig + MoreDigs;
102     CalcDig = DispDig + 2;
103     this.E = ApfloatMath.scale(new Apfloat("0.5", Apfloat.INFINITE),
104       (long) -1 * this.Dig);
105     this.f = f;
106     this.X = f.getX_Newton();
107 
108     calc();
109 
110     initOutputFormat();
111 
112     iterate();
113   }
114 
115   public void iterate() {
116     if (fAbl(X).signum() == 0)
117       System.out.println("Division durch Null!");
118     else {
119       if (showSteps)
120         while (!abbruch()) {
121           print();
122           next();
123         }
124       else
125         while (!abbruch())
126           next();
127       print();
128     }
129     System.out.println();
130   }
131 
132 }
```

```
1  package Funktionen;
2  import java.util.Set;
3  
4  import org.apfloat.Apfloat;
5  
6  /*
7   * Interface für Funktionen
8   */
9  public interface Funktion {
10   // Funktion zur Auswertung
11   public Apfloat f(Apfloat x);
12 
13   // Ableitung für das Newtonverfahren
14   public Apfloat f_Abl(Apfloat x);
15 
16   // Fixpunktgleichung für das Fixpunktverfahren
17   public Apfloat f_Fixpunkt(Apfloat x);
18 
19   // Undefinierte Werte dürfen nicht ausgewertet werden
20   public void setUndef(Set<Apfloat> undef);
```

```
21  public Set<Apfloat> getUndef();
22
23  // Randstellen des Intervalls bei der Bisektion
24  public void setIntervall(Apfloat A, Apfloat B);
25  public void setIntervall(Number A, Number B);
26  public Apfloat getA();
27  public Apfloat getB();
28
29  // Startwert für Newtonverfahren
30  public void setX_Newton(Apfloat X);
31  public void setX_Newton(Double X);
32  public Apfloat getX_Newton();
33
34  // Startwert für Fixpunktverfahren
35  public void setX_Fixpunkt(Apfloat X);
36  public void setX_Fixpunkt(Double X);
37  public Apfloat getX_Fixpunkt();
38
39  public void setSolve(Apfloat x);
40  public Apfloat solve();
41 }
```

```
1  package Funktionen;
2
3  import java.util.Set;
4
5  import org.apfloat.Apfloat;
6
7  public class GenericFunktion implements Funktion {
8   // Anzahl Nachkommastellen mit denen gerechnet werden soll
9   int stellen;
10
11  // Startwerte, falls setIntervall, etc. nicht verwendet werden
12  Apfloat A;
13  Apfloat B;
14  Apfloat X_Newton;
15  Apfloat X_Fixpunkt;
16  Apfloat solve; // Genaue Loesung fuer Fehlerberechnung
17
18  // Andere Felder
19  Set<Apfloat> undef;
20
21  public GenericFunktion(int stellen) {
22   this.stellen = stellen;
23  }
24
25  @Override
26  public Apfloat f(Apfloat x) {
27   if (undef.contains(x))
28    throw new IllegalArgumentException("Undefinierter Wert: " + x);
29   return null;
30  }
31
```

```
32  @Override
33  public void setUndef(Set<Apfloat> undef) {
34   this.undef = undef;
35  }
36
37  @Override
38  public Set<Apfloat> getUndef() {
39   return undef;
40  }
41
42  @Override
43  public void setIntervall(Apfloat A, Apfloat B) {
44   this.A = A;
45   this.B = B;
46  }
47
48  @Override
49  public void setIntervall(Number A, Number B) {
50   this.A = new Apfloat(A.toString(), Apfloat.INFINITE);
51   this.B = new Apfloat(B.toString(), Apfloat.INFINITE);
52  }
53
54  @Override
55  public Apfloat getA() {
56   return A;
57  }
58
59  @Override
60  public Apfloat getB() {
61   return B;
62  }
63
64  @Override
65  public void setX_Newton(Apfloat X) {
66   this.X_Newton = X;
67  }
68
69  @Override
70  public void setX_Newton(Double X) {
71   this.X_Newton = new Apfloat(X.toString(), stellen);
72  }
73
74  @Override
75  public Apfloat getX_Newton() {
76   return X_Newton;
77  }
78
79  @Override
80  public void setX_Fixpunkt(Apfloat X) {
81   this.X_Fixpunkt = X;
82  }
83
84  @Override
```

```
85   public void setX_Fixpunkt(Double X) {
86    this.X_Fixpunkt = new Apfloat(X.toString(), Apfloat.INFINITE);
87   }
88
89   @Override
90   public Apfloat getX_Fixpunkt() {
91    return X_Fixpunkt;
92   }
93
94   @Override
95   public Apfloat f_Fixpunkt(Apfloat x) {
96    return null;
97   }
98
99   @Override
100  public Apfloat f_Abl(Apfloat x) {
101   return null;
102  }
103
104  @Override
105  public void setSolve(Apfloat x) {
106   this.solve = x;
107  }
108
109  @Override
110  public Apfloat solve() {
111   return solve;
112  }
113 }
```

```
1
2  import org.apfloat.Apfloat;
3  import org.apfloat.ApfloatMath;
4
5  public class Polynomfunktion extends GenericFunktion implements Funktion {
6   int FixpunktExponent;
7
8   public Polynomfunktion(int stellen, int exp, Apfloat... fakt) {
9    this(stellen, false, exp, fakt);
10  }
11
12  public Polynomfunktion(int stellen, boolean Abl, int exp, Apfloat... fakt)
        {
13   super(stellen);
14
15   this.FixpunktExponent = exp;
16   n = fakt.length;
17   this.fakt = fakt;
18
19   if (!Abl)
20    setUpAbleitung();
21
22   A = new Apfloat("-10", stellen);
```

```
23   B = new Apfloat("10", stellen);
24   X_Newton = new Apfloat("1", stellen);
25   X_Fixpunkt = new Apfloat("1", stellen);
26  }
27
28  public Polynomfunktion(int stellen, boolean Abl, int exp, Number... fakt)
        {
29   super(stellen);
30   this.FixpunktExponent = exp;
31
32   n = fakt.length;
33   this.fakt = new Apfloat[n];
34   for (int k = 0; k < n; k++)
35    this.fakt[k] = new Apfloat(fakt[k].toString(), stellen);
36
37   if (!Abl)
38    setUpAbleitung();
39
40   A = new Apfloat("-10", stellen);
41   B = new Apfloat("10", stellen);
42   X_Newton = new Apfloat("1", stellen);
43   X_Fixpunkt = new Apfloat("1", stellen);
44  }
45
46  public Polynomfunktion(int stellen, int exp, Number... fakt) {
47   this(stellen, false, exp, fakt);
48  }
49
50  int n;
51  public Apfloat[] fakt;
52  Polynomfunktion f_Abl;
53
54  @Override
55  public Apfloat f(Apfloat x) {
56   Apfloat val = Apfloat.ZERO;
57   if (x.compareTo(val) == 0)
58    return fakt[n - 1];
59   for (int exp = 0; exp < n; exp++)
60    val = val.add(
61      fakt[n - exp - 1].multiply(
62        ApfloatMath.pow(x, exp).precision(stellen))
63        .precision(stellen)).precision(stellen);
64   return val;
65  }
66
67  @Override
68  public Apfloat f_Abl(Apfloat x) {
69   return f_Abl.f(x);
70  }
71
72  @Override
73  public Apfloat f_Fixpunkt(Apfloat x) {
74   Apfloat val = f(x);
```

```
75    val = val.subtract(fakt[n - FixpunktExponent - 1].multiply(
76      ApfloatMath.pow(x, FixpunktExponent).precision(stellen))
77      .precision(stellen));
78    val = val.negate();
79    val = val.divide(fakt[n - FixpunktExponent - 1]);
80    val = ApfloatMath.root(val, FixpunktExponent);
81    return val;
82   }
83
84   private void setUpAbleitung() {
85    if (n == 1)
86     f_Abl = new Polynomfunktion(stellen, true, n - 2, 0.0);
87    Apfloat[] faktAbl = new Apfloat[n - 1];
88    for (Integer exp = 1; exp < n; exp++)
89     faktAbl[n - exp - 1] = new Apfloat(exp.toString(), stellen)
90       .multiply(fakt[n - exp - 1]).precision(stellen);
91    f_Abl = new Polynomfunktion(stellen, true, n - 2, faktAbl);
92   }
93
94   @Override
95   public String toString() {
96    StringBuilder sb = new StringBuilder();
97    for (int k = 0; k < n - 2; k++)
98     sb.append(fakt[k].toString(true) + "x^" + (n - k - 1) + " + ");
99    sb.append(fakt[n - 2].toString(true) + "x + "
100     + fakt[n - 1].toString(true));
101   return sb.toString();
102  }
103 }
```

```
1  package Funktionen;
2
3  import org.apfloat.Apfloat;
4  import org.apfloat.ApfloatMath;
5
6  public class Exponentialfunktion extends GenericFunktion implements
       Funktion {
7   public Exponentialfunktion(int stellen) {
8    super(stellen);
9
10   // Konstanten initialisieren:
11   TWO = new Apfloat("2", stellen);
12   THREE = new Apfloat("3", stellen);
13   TEN = new Apfloat("10", stellen);
14  }
15
16  Apfloat TWO, THREE, TEN;
17
18  // Startwerte, falls setIntervall etc. nicht verwendet werden
19  Apfloat A = new Apfloat("1.0", Apfloat.INFINITE);
20  Apfloat B = new Apfloat("2.0", Apfloat.INFINITE);
21  Apfloat X_Newton = new Apfloat("1.5", Apfloat.INFINITE);
22  Apfloat X_Fixpunkt = new Apfloat("1.5", Apfloat.INFINITE);
```

```
23
24   @Override
25   public Apfloat f(Apfloat x) {
26    // f(x) = 2^x+3^x-10
27    return ApfloatMath.pow(TWO, x).add(ApfloatMath.pow(THREE, x))
28      .subtract(TEN);
29   }
30
31   @Override
32   public Apfloat f_Abl(Apfloat x) {
33    // f(x) = ln(2)*2^x + ln(3)*3^x
34    return ApfloatMath
35      .log(TWO)
36      .multiply(ApfloatMath.pow(TWO, x))
37      .add(ApfloatMath.log(THREE).multiply(ApfloatMath.pow(THREE, x)));
38   }
39
40   @Override
41   public Apfloat f_Fixpunkt(Apfloat x) {
42    // g(x) =log(10-2^x)/log(3)
43    return ApfloatMath.log(TEN.subtract(ApfloatMath.pow(TWO, x))).divide(
44      ApfloatMath.log(THREE));
45   }
46
47   @Override
48   public String toString() {
49    return "2^x+3^x-10";
50   }
51 }
```

```
1  package Funktionen;
2
3  import org.apfloat.Apfloat;
4  import org.apfloat.ApfloatMath;
5
6  public class SpezielleFunktion extends GenericFunktion implements Funktion
       {
7   public SpezielleFunktion(int stellen) {
8    super(stellen);
9
10   // Konstanten initialisieren
11   ONE = new Apfloat("1", stellen);
12   EE = ApfloatMath.exp(ApfloatMath.exp(ONE.negate()));
13  }
14
15  Apfloat ONE, EE;
16
17  // Startwerte, falls setIntervall, etc. nicht verwendet werden
18  Apfloat A = new Apfloat("-1.0", stellen);
19  Apfloat B = new Apfloat("1.0", Apfloat.INFINITE);
20  Apfloat X_Newton = new Apfloat("-0.5", Apfloat.INFINITE);
21  Apfloat X_Fixpunkt = new Apfloat("-0.5", Apfloat.INFINITE);
22
```

```
23   @Override
24   public Apfloat f(Apfloat x) {
25    // x^2-cos(|x|^x*nthroot(e, e)-1)
26    Apfloat val = x.precision(stellen); // x
27    val = ApfloatMath.abs(val); // |x|
28    val = ApfloatMath.pow(val, x); // |x|^x
29    val = val.multiply(EE); // |x|^x*sqrt[e]{e}
30    val = val.subtract(ONE); // |x|^x*sqrt[e]{e} - 1
31    val = ApfloatMath.cos(val); // cos(|x|^x*sqrt[e]{e} - 1)
32    val = val.negate(); // -cos(|x|^x*sqrt[e]{e} - 1)
33    val = val.add(ApfloatMath.pow(x, 2)); // -cos(|x|^x*sqrt[e]{e} - 1) +
34                // x^2
35    return val;
36   }
37
38   @Override
39   public Apfloat f_Abl(Apfloat x) {
40    // 2x+sin(|x|^x*nthroot(e, e)-1)*nthroot(e, e)*(ln|x|+1)*|x|^x
41    Apfloat val = x.precision(stellen);
42    val = ApfloatMath.abs(val);
43    val = ApfloatMath.pow(val, x);
44    val = val.multiply(EE);
45    val = val.subtract(ONE);
46    val = ApfloatMath.sin(val);
47    val = val.multiply(EE);
48    val = val.multiply(ApfloatMath.log(ApfloatMath.abs(x)).add(ONE));
49    val = val.multiply(ApfloatMath.pow(ApfloatMath.abs(x), x));
50    val = val.add(x.add(x));
51    return val;
52   }
53
54   @Override
55   public Apfloat f_Fixpunkt(Apfloat x) {
56    // x^2-cos(|x|^x*nthroot(e, e)-1)
57    Apfloat val = x.precision(stellen); // x
58    val = ApfloatMath.abs(val); // |x|
59    val = ApfloatMath.pow(val, x); // |x|^x
60    val = val.multiply(EE); // |x|^x*sqrt[e]{e}
61    val = val.subtract(ONE); // |x|^x*sqrt[e]{e} - 1
62    val = ApfloatMath.cos(val); // cos(|x|^x*sqrt[e]{e} - 1)
63    val = ApfloatMath.root(val, 2);
64    return val;
65   }
66  }
```

Erklärung der Eigenständigkeit

Hiermit erkläre ich, dass ich diese Arbeit selbstständig verfasst habe und nur die angegebenen Quellen verwendet habe. Die Mitwirkung von anderen Personen ist auf Beratung und Korrekturlesen beschränkt.

Ort, Datum	Unterschrift