LEHRSTUHL FÜR NACHRICHTENTECHNIK

UNIVERSITÄT ERLANGEN-NÜRNBERG

Professor Dr.-Ing. W. Schüßler

Studienarbeit

Untersuchung von Algorithmen zur Bestimmung von Polynomnullstellen

Hochschullehrer: P.Steffen
Betreuer: M. Lang
bearbeitet von: B. Frenzel
Erlangen, den 27. März 1993

LEHRSTUHL FÜR NACHRICHTENTECHNIK

UNIVERSITÄT ERLANGEN-NÜRNBERG

Professor Dr.-Ing. W. Schüßler

Studienarbeit

für

Herrn cand.ing. Bernhard Frenzel

Untersuchung von Algorithmen zur Bestimmung von Polynomnullstellen

Die Bestimmung von Polynomnullstellen stellt eine Standardaufgabe in der Nachrichtentechnik dar. Bekannte Verfahren haben in der erreichbaren Genauigkeit bzw. dem hohen Rechen- und Speicheraufwand ihre Grenzen.

Herr Frenzel erhält die Aufgabe, auf der Basis eines vorhandenen FORTRAN-Programmes ein leistungsfähiges C-Programm zur Berechnung aller Nullstellen eines reellwertigen bzw. komplexwertigen Polynoms zu erstellen. Dabei wird besonderer Wert auf eine gute Strukturierung und eine ausreichende Kommentierung gelegt. Die Leistungsfähigkeit soll anhand von geeigneten Testpolynomen hinsichtlich des Rechen- und Speicheraufwandes sowie der erreichbaren Genauigkeit beurteilt werden. Zum Vergleich stehen ein Verfahren nach Jenkins und Traub sowie eines auf einer Eigenwertbestimmung beruhendes zur Verfügung.

Ausgabe : 04.11.1992

Abgabe :

(Priv.Doz.Dr.-Ing. habil P.Steffen)

ERKLÄRUNG

Ich versichere, daß ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und daß die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen hat. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen,

.....

Ratiborerstraße 4
8520 Erlangen

Inhaltsverzeichnis

1 Zusammenfassung **3**

2 Verwendete Abkürzungen und Formeln **4**

3 Einleitung **5**

4 Iterationsverfahren **6**
4.1 Allgemeine Bemerkungen 6
4.2 Das Newton-Verfahren 8
4.3 Das Sekanten-Verfahren 10
4.4 Das Muller-Verfahren 12

5 Verwendete Verfahren und Materialien **15**

6 Linearphasige FIR-Filter **16**

7 Mehrfache Nullstellen **17**

8 Aufbau des Programmes 'rootsf' **18**
8.1 Gliederung 18
8.2 Die Files 'header.h' und 'complex.c' 18
8.3 Die Hauptroutine null() 20
8.3.1 Struktur und Funktionsweise 20
8.3.2 Numerische Überlegungen 21
8.3.3 Bedeutung der im Programm auftretenden Größen 23
8.4 Das Muller-Verfahren muller() 24
8.4.1 Struktur und Funktionsweise 24
8.4.2 Numerische Überlegungen 26
8.5 Das Newton-Verfahren newton() 28
8.5.1 Struktur und Funktionsweise 28
8.5.2 Numerische Überlegungen 29

9 Ergebnisse und Vergleich mit anderen Testverfahren **30**
9.1 Testverfahren nach Jenkins und Traub 30
9.1.1 Das Abbruchkriterium 31
9.1.2 Das Konvergenzkriterium 32
9.1.3 Mehrfache oder benachbarte Nullstellen 33
9.1.4 Stabilität der Deflation 35
9.2 Kreisteilungspolynome 37
9.2.1 Vergleich von 'roots','rootsj' und 'rootsf' 37

9.2.2 Überlegungen zu Rechenzeit und Speicherbedarf 38
9.2.3 Untersuchungen zu 'rootsf' 40
9.3 Beispielhaft ausgewählte FIR-Filter 42

10 Zusammenfassung und Bewertung von 'rootsf' 43

11 Ausblick 45

Literatur 47

1 Zusammenfassung

Die Grundlage für die vorliegende Studienarbeit war ein von M. Lang erstelltes FORTRAN-Programm [7], das die Nullstellen reellwertiger Polynome mit einer Kombination aus den iterativen Verfahren von Muller und Newton bis zum Grad 800 berechnet. Das in C geschriebene, erweiterte und modifizierte Programm gemäß der Aufgabenstellung sollte zunächst auf einem PC durchgeführt werden. Dies erwies sich jedoch aufgrund folgender Punkte als ungeeignet:

1. erheblicher Zeitaufwand bei der Berechnung von Nullstellen von Polynomen hohen Grades
2. keine direkte Vergleichsmöglichkeit unter MS-DOS für die benötigte CPU-Zeit mit anderen Nullstellenberechnungsprogrammen
3. keine Möglichkeit der Auflistung des Zeitaufwandes der modularen Teile des Programmes

Daher wurde das Programm auf einer Workstation erstellt und erst nach Fertigstellung wieder auf einen PC übertragen.
Die Strukturierung des Programmes wurde modular gehalten, die einzelnen Programmteile schon teils im Programmtext ausfürlich kommentiert. Die wichtigsten Programmteile werden im folgenden noch einmal anhand von Struktogrammen erläutert bzw. zusammengefaßt.

Zum Test und Vergleich mit anderen Programmen zur Bestimmung von Nullstellen wurde das Programm über eine Gateway-Routine an das Programmpaket Matlab angebunden. Als Testpolynome wurden die von Jenkins und Traub [4] vorgeschlagenen Polynome sowie Kreisteilungspolynome und linearphasige FIR-Filter gewählt. Die Vergleichsprogramme waren das auf einer Eigenwertbestimmung beruhende, in Matlab implementierte Programm 'roots' und ein nach dem Verfahren nach Jenkins und Traub arbeitendes Programm 'cpoly', das im folgenden immer mit 'rootsj' bezeichnet wird.

Das Programm wurde sowohl in ANSI-C als auch in Kernighan-Ritchie-C verfaßt, um die Lauffähigkeit auch auf älteren Compilern zu gestatten, die noch kein ANSI-C verarbeiten können. Die Studienarbeit wurde zusammen mit einer Diskette mit allen notwendigen Programmen abgegeben.

2 Verwendete Abkürzungen und Formeln

LNT	Lehrstuhl für Nachrichtentechnik
TP	Tiefpaß
roots	in Matlab implementiertes Nullstellenbestimmungsprogramm auf der Grundlage der Eigenwertbestimmung der zu $P(x)$ gehörenden Begleitmatrix
rootsj	Nullstellenbestimmungsprogramm nach dem Verfahren von Jenkins und Traub (s. [5])
rootsf	lauffähige Version der in der Studienarbeit erstellten Funktion null()
f_ν	Funktionswert von $x^{(\nu)}$
$P(x)$	Originalpolynom, zu dem die Nullstellen bestimmt werden
$P_{red}(x)$	reduziertes Polynom nach Polynomdeflation
$\hat{P}(x)$	Näherung des Originalpolynomes
$\tilde{P}(x)$	Interpolationspolynom im Muller-Verfahren
x_0	gesuchte Nullstelle
$\tilde{x}_0$	genäherte Nullstelle = exakte Nullstelle von $\hat{P}(x)$
α_ν	exakte ν-te Nullstelle von P(x)
$\hat{\alpha}_\nu$	genäherte ν-te Nullstelle von $P(x)$ = exakte Nullstelle von $\hat{P}(x)$
DBL_MAX	größte darstellbare Zahl im double-Format; nach IEEE-P754-Floating-Point-Standard ist DBL_MAX $\approx 10^{308}$
log(x)	dekadischer Logarithmus der Zahl x
k	Iterationsindex
ϵ	Abbruchschranke im Muller- und Newtonverfahren
ε	Maschinengenauigkeit
m_ν	Vielfachheit der Nullstelle α_ν
PN-Diagramm	Pol-Nullstellen-Diagramm

3 Einleitung

Beim Entwurf von nichtrekursiven Filtern tritt häufig das Problem der Bestimmung von Polynomnullstellen auf, d.h. bei gegebenem Polynomgrad N und vorgegebenen Koeffizienten $p_0, p_1, \ldots, p_N$ sind die Nullstellen des Polynomes

$$P(x) = p_N x^N + p_{N-1} x^{N-1} + \ldots + p_1 x + p_0, \qquad p_N \neq 0$$

zu bestimmen. Hierbei ergeben sich nicht selten hohe Grade $N > 100$, die neben der erforderlichen Genauigkeit hohe Anforderungen an Laufzeit und Speicherbedarf stellen. Das am LNT von Jenkins und Traub [5] zur Verfügung stehende Programm ist für hohe Grade aufgrund der auftretenden Ungenauigkeiten ungeeignet. Das von Matlab zur Verfügung gestellte Nullstellenprogramm 'roots' hat zwar für hohe Grade noch eine hinreichende Genauigkeit, der Speicherplatzbedarf und die Rechenzeit steigen jedoch stark mit dem Polynomgrad an.

Nun sind zur Berechnung von Polynomnullstellen seit langem eine größere Anzahl von Algorithmen bekannt. Prinzipiell kann man hierbei zwischen den *iterativen Verfahren*, welche die Nullstellen nacheinander berechnen, und den *direkten Methoden*, die alle Nullstellen gleichzeitig berechnen, unterscheiden. Die ersten Verfahren benötigen dabei die Verwendung von expliziter oder impliziter Polynomdeflation, um die bereits berechneten Nullstellen abzudividieren. Durch diese Polynomdeflation werden die restlichen Nullstellen im Laufe der Iteration immer schneller gefunden. Allerdings kann durch eine 'ungünstige' Reihenfolge der gefundenen Nullstellen eine Verschlechterung der Kondition (siehe [13]) der verbleibenden Nullstellen auftreten, was einen Genauigkeitsverlust zur Folge hat.

Für dieses Programm wurde nun eine Kombination aus zwei iterativen Verfahren, dem Muller- und Newton-Verfahren, gewählt. Der Vorteil gegenüber direkten Methoden liegt hierbei in der höheren Rechengeschwindigkeit bei hochgradigen Polynomen. Der Nachteil der Verschlechterung der folgenden Nullstellen durch Deflation des Polynomes durch möglicherweise vorher ungenau bestimmte Nullstellen wird durch die Nachiteration im Newton-Verfahren mit Hilfe des ursprünglichen Polynomes beseitigt (siehe [13] S. 82ff).

4 Iterationsverfahren

4.1 Allgemeine Bemerkungen

Die hier vorgestellten Iterationsverfahren für Polynomnullstellen können als Sonderfälle eines allgemeinen Iterationsverfahrens für nichtlineare Gleichungen angesehen werden (s. [1] u. [11]). Hierbei wird zunächst der Ausdruck der skalaren und stetigen Funktion $p(x)$

$$p(x) = 0 \tag{1}$$

durch äquivalente Umformung auf die Form

$$g(x) = h(x) \tag{2}$$

gebracht. Die Funktionen $g(x)$ und $h(x)$ sind in Gleichung (2) wiederum stetig. Alle Werte für x, die Gleichung (1) erfüllen, sind dabei die Nullstellen oder Wurzeln der Funktion $p(x)$. Die Konstruktion einer konvergenten Zahlenfolge $[x^{(k)}]_{k=0}^{+\infty}$ nach der Vorschrift

$$g(x^{(k+1)}) = h(x^{(k)}) \tag{3}$$

mit gegebenem Startwert $x^{(0)}$ liefert ein Iterationsverfahren zur Bestimmung einer Nullstelle von $p(x)$. Der Grenzübergang einer solchen Folge führt dann auf die gesuchte Nullstelle x_0, denn wegen der geforderten Konvergenz gilt:

$$\lim_{k\to+\infty} x^{(k+1)} = \lim_{k\to+\infty} x^{(k)} = \tilde{x}_0,$$

mit $\tilde{x}_0$ als Grenzwert der Zahlenfolge $[x^{(k)}]_{k=0}^{+\infty}$. Aus der Vorschrift (3) mit

$$\lim_{k\to+\infty} g(x^{(k+1)}) = \lim_{k\to+\infty} h(x^{(k)})$$

folgt dann wegen der geforderten Stetigkeit für $h(x)$ und $g(x)$ im Grenzübergang:

$$\begin{aligned} g(\tilde{x}_0) &= h(\tilde{x}_0) \\ \Rightarrow p(\tilde{x}_0) &= g(\tilde{x}_0) - h(\tilde{x}_0) = 0 \end{aligned}$$

d.h. $\tilde{x}_0 = x_0$.

Hinreichend für die geforderte Konvergenz der Zahlenfolge ist, daß

- die Ableitungen $h'(x) = \frac{d}{dx}h(x)$ und $g'(x) = \frac{d}{dx}g(x)$ in einer Umgebung von x_0 stetig sind
- der Startwert $x^{(0)}$ innerhalb dieser Umgebung liegt und dort

$$|g'(x)| > |h'(x)| \tag{4}$$

  gilt.

Im folgenden sei nun $g(x) = x \Rightarrow g'(x) = 1$. Damit ergibt sich für die Ungleichung (4), daß $|h'(x)| < 1$ gelten muß, für Gleichung (2) folgt

$$x = h(x). \tag{5}$$

Wird für $p(x)$ ein Polynom $P(x)$ mit dem Grad N angenommen, so ist auch Stetigkeit und Differenzierbarkeit für $g(x)$ und $h(x)$ und deren Ableitungen sichergestellt. Die nun angegebenen Verfahren unterscheiden sich hauptsächlich in der Wahl von $h(x)$.
Es sei an dieser Stelle darauf hingewiesen, daß solche Iterationsverfahren nach Gleichung (5) nicht die exakten Nullstellen eines Polynomes $P(x)$ liefern, sondern bestenfalls Näherungen in den Grenzen der Rechengenauigkeit. Für ein Polynom $P(x)$ vom Grad N in der faktorisierten Form

$$P(x) = p_N(x - \alpha_1)^{m_1} \dots (x - \alpha_l)^{m_l}$$

mit den Nullstellen $\alpha_1, \alpha_2, \dots, \alpha_l$ und den zugehörigen Vielfachheiten $m_1, m_2, \dots, m_l$ findet man also höchstens N Näherungen $\hat{\alpha}_1, \hat{\alpha}_2, \dots, \hat{\alpha}_l$ der Form

$$\hat{P}(x) = p_N(x - \hat{\alpha}_1)(x - \hat{\alpha}_2) \dots (x - \hat{\alpha}_l),$$

wobei $\hat{P}(x)$ ein genähertes Polynom des ursprünglichen $P(x)$ darstellt. Die Nullstelle α_ν kann dann als 'gefunden' betrachtet werden, wenn die zugehörige Näherung $\hat{\alpha}_\nu$ im Rahmen der Rechengenauigkeit für die Weiterverarbeitung der Nullstelle den geforderten Genauigkeitsansprüchen genügt.

4.2 Das Newton-Verfahren

Ein sehr häufig verwendetes Verfahren zur numerischen Berechnung von Nullstellen ist das *Newton-Verfahren*. Hierbei wird in Gleichung (5) der Ausdruck $h(x)$ durch

$$h(x) = x - \frac{P(x)}{P'(x)} \qquad \text{mit} \qquad P'(x) \neq 0$$

ersetzt. Die Iterationsbeziehung ergibt sich somit zu:

$$x^{(k+1)} = x^{(k)} - \frac{P(x^{(k)})}{P'(x^{(k)})}, \qquad P'(x^{(k)}) \neq 0. \tag{6}$$

Aus Ungleichung (4) ergibt sich lokale Konvergenz des Verfahrens, wenn $|h'(x)| = \left|\frac{P''(x)P(x)}{P'^2(x)}\right| < 1$ in einer Umgebung der Nullstelle x_0 gilt. Für reellwertige Polynome und Nullstellen kann man das Verfahren graphisch folgendermaßen interpretieren:

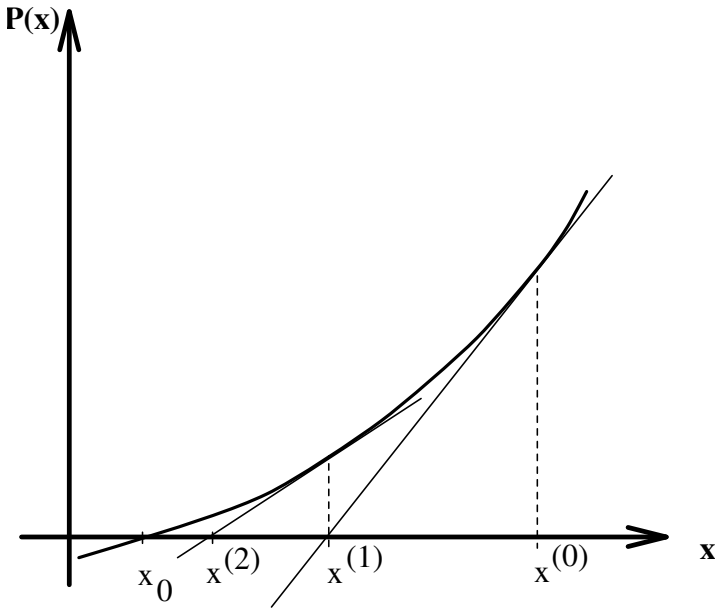

Abbildung 1: Newton-Verfahren

Das Polynom $P(x)$ besitze in x_0 eine reellwertige Nullstelle (Abbildung 1) und es gelte $P'(x^{(0)}) \neq 0$. Für die Steigung der Tangente an $(x^{(0)}, P(x^{(0)}))$ ergibt sich aus der Abbildung 1

$$P'(x^{(0)}) = \frac{P(x^{(0)})}{x^{(0)} - x^{(1)}}, \qquad x^{(0)} \neq x^{(1)},$$

und somit

$$x^{(1)} = x^{(0)} - \frac{P(x^{(0)})}{P'(x^{(0)})}, \qquad P'(x^{(0)}) \neq 0. \tag{7}$$

Bei gegebenem Startwert $x^{(0)}$ ist der Schnittpunkt $x^{(1)}$ der Tangente mit der x-Achse eine im allgemeinen verbesserte Näherung von x_0. Aus der Verallgemeinerung der Beziehung (7) folgt die Iterationsvorschrift (6). Das Newtonsche Verfahren ist jedoch nicht beschränkt auf die Berechnung der Wurzeln von reellwertigen Polynomen. Es kann ebenso zur Berechnung der Wurzeln von komplexwertigen Polynomen herangezogen werden. Die Iterationsvorschrift (6) ändert sich hierbei formal nicht.

In dieser Arbeit wurde von der Eigenschaft der lokalen Konvergenz des Newton-Verfahrens sowohl bei reellwertigen als auch bei komplexwertigen Polynomen Gebrauch gemacht. Da das Newton-Verfahren als ein Nachiterationsverfahren verwendet wurde, werden an dieser Stelle keine Überlegungen zur globalen Konvergenz angestellt. Bemerkungen hierüber finden sich z.B. in [12]. Beim Durchlaufen der Newton-Iteration wird davon ausgegangen, daß sich durch das Muller-Verfahren der Startwert bereits hinreichend nahe an der exakten Nullstelle befindet.

4.3 Das Sekanten-Verfahren

Ein weiteres Verfahren, das in dieser Arbeit als ein Nachiterationsverfahren erprobt wurde, ist das *Sekanten-Verfahren*. Hierbei wird $h(x)$ in (5) zu

$$h(x) = x - \frac{P(x)}{\beta}, \qquad \beta = \frac{P(x) - P(x_1)}{x - x_1}, \qquad x \neq x_1.$$

Im Vergleich zum Newton-Verfahren ist die Ableitung $P'(x)$ durch den Differenzenquotienten β ersetzt worden. Daraus folgt für die Iterationsbeziehung:

$$\begin{aligned} x^{(k+1)} &= x^{(k)} - P(x^{(k)}) \frac{x^{(k)} - x^{(k-1)}}{P(x^{(k)}) - P(x^{(k-1)})}, \\ &\text{mit} \qquad P(x^{(k)}) \neq P(x^{(k-1)}). \end{aligned} \tag{8}$$

Hierbei sind $x^{(0)}$ und $x^{(1)}$ zwei gegebene Startwerte in der Nähe der gesuchten Nullstelle x_0.

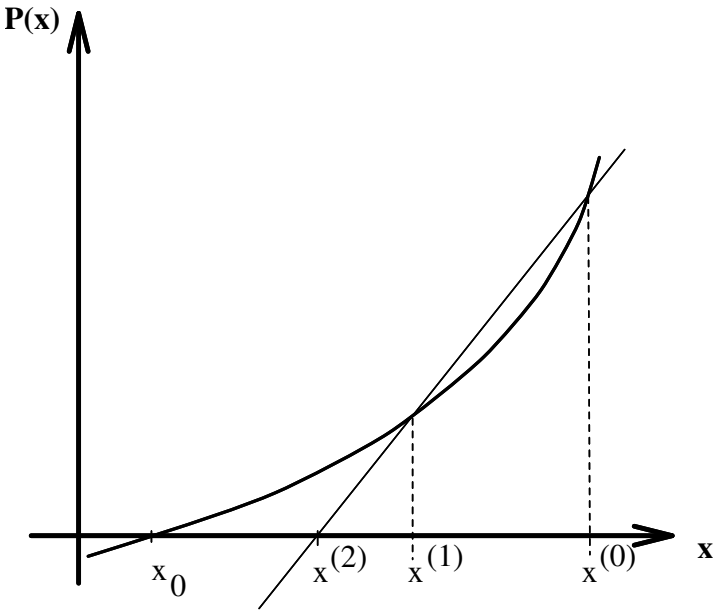

Abbildung 2: Sekanten-Verfahren

Das Sekanten-Verfahren kann graphisch folgendermaßen interpretiert werden:
Das Polynom $P(x)$ besitze in x_0 eine reellwertige Nullstelle (Abbildung 2). Es seien zwei Punkte $x^{(0)}$ und $x^{(1)}$ in der Umgebung der gesuchten Nullstelle x_0 vorgegeben.

Die Sekante durch die beiden Punkte $(x^{(0)}, P(x^{(0)}))$ und $(x^{(1)}, P(x^{(1)}))$ genügt dann der Gleichung

$$y = \frac{P(x^{(1)}) - P(x^{(0)})}{x^{(1)} - x^{(0)}}(x - x^{(1)}) + P(x^{(1)}). \tag{9}$$

Für $x = x^{(2)}$ ist $y = 0$. Löst man nun Gleichung (9) nach $x^{(2)}$ auf, so folgt:

$$x^{(2)} = x^{(1)} - P(x^{(1)})\frac{P(x^{(1)}) - P(x^{(0)})}{x^{(1)} - x^{(0)}} \tag{10}$$

Verallgemeinert man die Beziehung (10), so ergibt sich hieraus die Iterationsvorschrift (8) des Sekantenverfahrens.

Der Hauptvorteil dieses Verfahrens liegt darin, daß die Ableitung $P'(x)$ nicht explizit bestimmt werden muß, was eine erhebliche Zeitersparnis bedeuten kann. Allerdings benötigt man im Vergleich zum Newton-Verfahren einen zweiten Startwert. Ein weiterer Nachteil des Verfahrens ist die geringere Konvergenzordnung als beim Newton-Verfahren. Bei einfachen Nullstellen beispielsweise ist die Konvergenzordnung des Newton-Verfahrens $r = 2$, die des Sekanten-Verfahrens nur $r \approx 1.62$ (siehe [3]). Diese Tatsache kann unter Umständen bei einer geforderten Mindestgenauigkeit der Approximation einer Nullstelle nur durch eine Vergrößerung der Anzahl an Iterationsschritten im Sekantenverfahren kompensiert werden.

4.4 Das Muller-Verfahren

Das zentrale Verfahren des erstellten Programmes 'rootsf' ist das sog. *Muller-Verfahren* (siehe [11] und [3]). Dieses Verfahren ist eine Erweiterung des Sekantenverfahrens dadurch, daß die Sekante zu einem Polynom 2. Grades wird, eine Parabel also. Es ergibt sich somit folgende Iterationsbeziehung:

$$\begin{aligned} x^{(k+1)} &= x^{(k)} + h_k q_{k+1} \qquad \text{mit} \\ q_{k+1} &= \frac{-2C_k}{B_k \pm \sqrt{B_k^2 - 4A_kC_k}} \\ h_k &= x_k - x_{k-1} \\ A_k &= q_k P(x_k) - q_k(1+q_k)P(x_{k-1}) + q_k^2 P(x_{k-2}) \\ B_k &= (2q_k+1)P(x_k) - (1+q_k)^2 P(x_{k-1}) + q_k^2 P(x_{k-2}) \\ C_k &= (1+q_k)P(x_k). \end{aligned} \tag{11}$$

Hierbei ergeben sich aus q_{k+1} die Nullstellen der approximierten Parabel, wobei für den nächsten Iterationsschritt der betragsmäßig kleinere Wert für q_{k+1} zu wählen ist. Es ist also für den Ausdruck $B_k \pm \sqrt{B_k^2 - 4A_kC_k}$ der betragsmäßig größere Wert zu nehmen. Sollte dieser Ausdruck verschwinden, so schlägt Muller vor, $|q_{k+1}| = 1$ zu setzen.

Die Herleitung von Gleichung (11) ergibt sich folgendermaßen (siehe [3]):
Mit Hilfe der Polynominterpolation in der Form von Lagrange [1] läßt sich ein Approximationspolynom $\tilde{P}(x)$ von höchstens m-ten Grades angeben, das an $m+1$ verschiedenen Stützstellen $x^{(\nu)}, \nu = 1(1)(m+1)$ mit $P(x)$ identisch ist. Das Interpolationspolynom hat dann folgende Form:

$$\begin{aligned} \tilde{P}(x) &= \sum_{i=0}^{m} L_i(x) P(x^{(i)}), \qquad \text{mit} \\ L_i(x) &= \frac{(x-x^{(0)})\dots(x-x^{(i-1)})(x-x^{(i+1)})\dots(x-x^{(m)})}{(x^{(i)}-x^{(0)})\dots(x^{(i)}-x^{(i-1)})(x^{(i)}-x^{(i+1)})\dots(x^{(i)}-x^{(m)})} \end{aligned}$$

Beim Spezialfall der quadratischen Interpolation ($m = 2$) ergibt sich somit:

$$\begin{aligned} \tilde{P}(x) &= L_0(x)P(x^{(0)}) + L_1(x)P(x^{(1)}) + L_2(x)P(x^{(2)}) \\ &= \frac{(x-x^{(1)})(x-x^{(2)})}{(x^{(0)}-x^{(1)})(x^{(0)}-x^{(2)})} P(x^{(0)}) + \frac{(x-x^{(0)})(x-x^{(2)})}{(x^{(1)}-x^{(0)})(x^{(1)}-x^{(2)})} P(x^{(1)}) \\ &\quad + \frac{(x-x^{(0)})(x-x^{(1)})}{(x^{(2)}-x^{(0)})(x^{(2)}-x^{(1)})} P(x^{(2)}) \end{aligned} \tag{12}$$

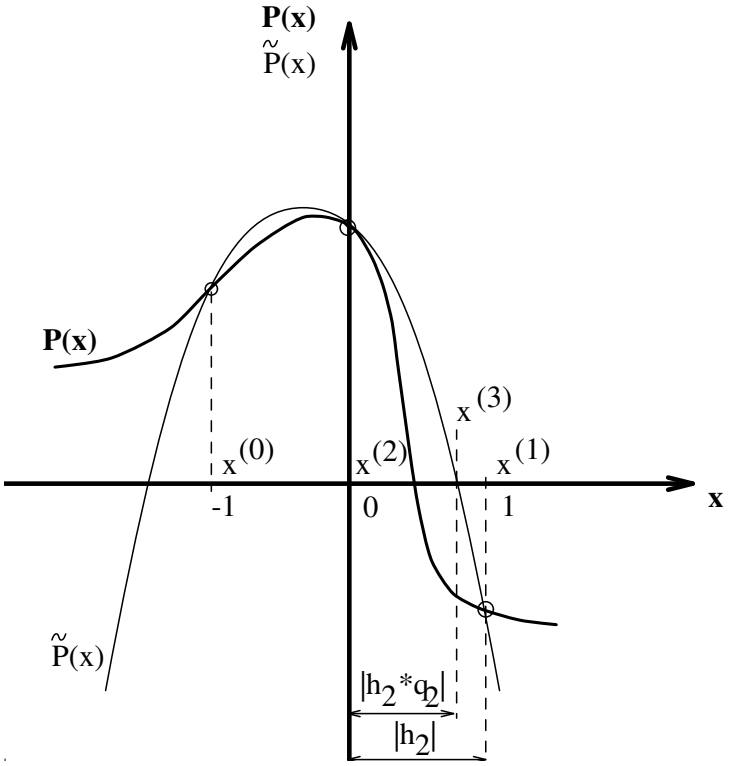

Abbildung 3: Muller-Verfahren (Initialisierung)

Die Startwerte $x^{(0)}, x^{(1)}, x^{(2)}$ liefern nun die drei zugehörigen Stützpunkte $(x^{(0)}, P(x^{(0)})); (x^{(1)}, P(x^{(1)})); (x^{(2)}, P(x^{(2)}))$. Mit den Abkürzungen

$$\begin{aligned} h_2 &= x^{(2)} - x^{(1)} \\ h_1 &= x^{(1)} - x^{(0)} \\ h &= x - x^{(2)} \end{aligned}$$

folgt aus (12):

$$\begin{aligned} \tilde{P}(x) = \tilde{P}(x^{(2)} + h) &= \frac{h(h+h_2)}{(h_2+h_1)h_1} P(x^{(0)}) - \frac{h(h+h_2+h_1)}{h_2 h_1} P(x^{(1)}) \\ &\quad + \frac{(h+h_2)(h+h_2+h_1)}{(h_2+h_1)h_2} P(x^{(2)}) \end{aligned} \tag{13}$$

Faßt man nun in (13) die Potenzen der Variablen h zusammen und führt die Abkürzungen

$$q_2 = \frac{h_2}{h_1}, \qquad q = \frac{h}{h_2}$$

ein, so ergibt sich die Parabelgleichung:

$$\tilde{P}(x) = \tilde{P}(x^{(2)} + h_2 q) = \frac{A_2 q^2 + B_2 q + C_2}{1 + q_2} \tag{14}$$

mit

$$\begin{aligned} A_2 &= q_2 P(x^{(2)}) - q_2(1+q_2)P(x^{(1)}) + q_2^2 P(x^{(0)}) \\ B_2 &= (2q_2+1)P(x^{(2)}) - (1+q_2)^2 P(x^{(1)}) + q_2^2 P(x^{(0)}) \\ C_2 &= (1+q_2)P(x^{(2)}) \end{aligned}$$

Bis auf die Variable q sind somit alle Parameter in (14) durch die Wahl der Startwerte festgelegt. Wie beim Sekantenverfahren werden nun die Nullstellen der approximierenden Parabel, also von Gleichung (14) bestimmt. Aus $A_2 q^2 + B_2 q + C_2 = 0$ folgen dann die beiden Nullstellen

$$q_{01,02} = \frac{-2C_2}{B_2 \pm \sqrt{B_2^2 - 4A_2 C_2}} \tag{15}$$

und damit die Lösungen $x_{01,02}$ der Gleichung $\tilde{P}(x^{(2)} + q h_2) = 0$ zu:

$$x_{01,02} = x^{(2)} + q_{01,02} h_2$$

Die betragsmäßig kleinere Nullstelle wird dann als neue Näherung für die gesuchte Nullstelle x_0 von $P(x)$ verwendet. Verworfen wird sowohl die zweite Parabelnullstelle, als auch der 'älteste' Iterationswert, in diesem Falle also $x^{(0)}$. Verallgemeinert man diese Überlegungen, so gelangt man zur Iterationsvorschrift (11) des Muller-Verfahrens. Eine graphische Veranschaulichung liefert Abbildung 3.

5 Verwendete Verfahren und Materialien

Für das erstellte Programm 'rootsf' wurde das Muller-Verfahren gewählt, da seine Konvergenzordnung r bei einfachen ($r \approx 1.84$) und auch bei doppelten ($r \approx 1.23$) Nullstellen noch recht gut ist. Globale Konvergenz des Verfahrens konnte zwar nicht nachgewiesen werden, jedoch ist mit den gegebenen Startwerten in allen untersuchten Fällen Konvergenz erreicht worden (siehe [3]).

Als Nachiterationsverfahren wurde zunächst das Sekantenverfahren gewählt. Es zeigte sich jedoch, daß zum Erreichen der gleichen Genauigkeit der Lösung wie unter Verwendung des Newton-Verfahrens sehr viel mehr Iterationsschritte nötig waren, was eine Vergrößerung der Rechenzeit bedeutete. Gerade bei hochgradigen Polynomen mit Grad $N > 100$ dauerte somit das Sekanten-Verfahren sogar länger als das Newton-Verfahren. Deshalb wurde trotz des Nachteils der Berechnung der lokalen Ableitung das Newton-Verfahren als Nachiterationsverfahren gewählt.
Zur Berechnung der Funktionswerte und der Polynomdeflation wurde das einfache bzw. doppelreihige Horner-Schema verwendet, zur Berechnung der Ableitungen das vollständige Horner-Schema (siehe [3]).

Das Programm wurde auf einer HP-Apollo-Workstation 9000/705 unter dem Betriebssystem HP-UNIX Version A.08.07 erstellt. Sämliche Testergebnisse in dieser Studienarbeit bezüglich Zeit und Rechengenauigkeit beziehen sich auf diese Maschine. Als weiterer Rechner stand ein PC HP Vectra 386/25 zur Verfügung. Sowohl auf diesem als auch auf der Workstation wurde die fertige Funktion als MEX-File in die Matlab-Versionen 3.5 (PC) und 4.0 (Workstation) eingebunden. Als Literatur standen die in der Literaturangabe aufgeführten Materialien zur Verfügung.

6 Linearphasige FIR-Filter

In der Studienarbeit wurde versucht, auf den Fall der Nullstellenberechnung des Zählerpolynomes $Z(z)$ eines linearphasigen FIR-Filters

$$H(z) = z^{-N} Z(z) = z^{-N} \sum_{\nu=0}^{N} b_\nu z^\nu$$

separat einzugehen. Dies hat eine Reihe von speziellen Eigenschaften (siehe [10]). Eine davon ist, daß wenn eine Nullstelle $z_{0\nu}$ nicht auf dem Einheitskreis liegt, sich dazu spiegelbildlich eine weitere bei $z_{0\lambda} = \frac{1}{z_{0\nu}^*}$ befindet. Bei Reellwertigkeit von $Z(z)$ folgt sogar ein Quadrupel von Nullstellen, das sich nur aus der Bestimmung einer einzigen Nullstelle der oben genannten Art ergibt. Die Ausnutzung dieser Eigenschaft ergäbe also einen wesentlichen Zeitvorteil, da mit der Bestimmung einer einzigen Nullstelle im günstigsten Fall der Grad des Polynomes um vier reduziert werden könnte.

Beim Abdividieren des Paares $z_{0\nu}$ und $z_{0\lambda}$ ergab sich jedoch bei einigen linearphasigen TP-Filtern höheren Grades, daß durch das Auftreten eines hohen Dynamikbereiches für die Koeffizienten des abdividierten Polynomes (z.B. $\approx 10^{-18} \ldots \approx 10^{18}$ bei einem TP vom Grad 800) das neue, abdividierte Polynom gänzlich andere Nullstellen als das ursprüngliche Polynom enthielt. Dies war beispielsweise daran zu erkennen, daß die Spiegeleigenschaft der Koeffizienten des abdividierten Polynomes bereits nach einigen Schritten verloren ging.

Zwei mögliche Lösungsansätze zur Beseitigung des auftretenden Problems sind im Ausblick dieser Studienarbeit erläutert.

7 Mehrfache Nullstellen

Mehrfache Nullstellen der Vielfachheit m_ν sind im allgemeinen schlecht konditioniert und lassen sich mit einer ungefähren Genauigkeit nach Gleichung (26) berechnen. Zur Verbesserung der Genauigkeit wurde in der Studienarbeit nun folgende Überlegung angestellt:

Hat ein Polynom $P(x)$ an der Stelle α_ν eine m_ν-fache Nullstelle, so sind auch die ersten $m_\nu - 1$ Ableitungen an der Stelle α_ν gleich Null.
Damit gilt nun sicherlich für eine Näherung $\hat{\alpha}_\nu$ der mehrfachen Nullstelle α_ν

$$P'(\hat{\alpha}_\nu) < \kappa \tag{16}$$

für ein hinreichend kleines κ. Durch Polynomdeflation wird nun ein Polynom $P_{red}(x)$ gebildet und Bedingung (16) für $P_{red}(x)$ überprüft. Der letzte Schritt wird sooft durchgeführt, bis Bedingung (16) nicht mehr erfüllt ist. Die Anzahl der Schritte gibt dann die Vielfachheit der Nullstelle wieder. Durch Polynomdeflation wird außerdem schrittweise die Kondition der Nullstelle durch Reduktion der Vielfachheit von α_ν in $P_{red}(x)$ verbessert. Für das zuletzt berechnete P_{red} kann dann durch Iteration im Newton-Verfahren die Nullstelle α_ν mit $\hat{\alpha}_\nu$ als Startwert sehr viel genauer als das bereits bestimmte $\hat{\alpha}_\nu$ bestimmt werden.

Das eben vorgestellte Verfahren konnte jedoch nicht realisiert werden, da sich Probleme bei der Wahl von κ in Gleichung (16) ergaben. Die Überlegung, κ auf die halbe Maschinengenauigkeit festzulegen, scheiterte daran, daß sich bei einfachen Nullstellen teilweise sehr kleine Ableitungen $P'(\hat{\alpha}_\nu)$ in der Größenordnung von κ ergaben. Durch die Erfüllung der Bedingung (16) wurden dann einfache Nullstellen als mehrfach erkannt und falsch abgespalten. Das Restpolynom enthielt dann Nullstellen, die vollkommen verschieden zu denen des Originalpolynoms waren.

8 Aufbau des Programmes 'rootsf'

8.1 Gliederung

Das Programm 'rootsf' ist der Übersichtlichkeit wegen in verschiedene Files aufgeteilt worden.

File	Inhalt
null.c	Hauptprogramm; überprüft die Eingaben auf formale Fehler und führt die Iterationen durch
muller.c	das Muller-Verfahren
newton.c	das Newton-Verfahren
tools.c	Funktionen zur Funktionswertberechnung und Polynomdeflation
complex.c	komplexe Funktionen
header.h	Includefile; enthält Definitionen und Deklarationen aller Funktionen
test.c	Testumgebung zum Testen des Programmes auf Betriebssystemebene
gate35.c	Gateway-Routine für Matlab Version 3.5
gate40.c	Gateway-Routine für Matlab Version 4.0

Tabelle 1: benötigte Files für die Nullstellenberechnungsfunktion null()

Eine Zusammenfassung zeigt Tabelle 1. Die drei Files 'test.c', 'gate35.c' und 'gate40.c' sind hierbei nur zum Test oder zur Anbindung an Matlab erstellt worden und gehören nicht zur eigentlichen Routine. Die Abhängigkeit der einzelnen Files untereinander zeigt Abbildung 4. Hierbei wird z.B. das Muller-Verfahren von 'null.c' aus aufgerufen usw. Das fehlende Includefile 'header.h' und das File 'complex.c' wird von allen Dateien des Hauptprogrammes eingebunden bzw. aufgerufen. Diese sind daher nicht mehr explizit aufgeführt.

8.2 Die Files 'header.h' und 'complex.c'

Das Includefile 'header.h' beinhaltet sämtliche für die einzelnen Funktionen benötigten Definitionen und globale Deklarationen. Es werden vom Includefile nur diejenigen Teile in die Funktionen eingebunden, die von der entsprechenden Funktion benötigt werden.
Das File 'complex.c' enthält im wesentlichen die in [9] angegebenen Funktionen

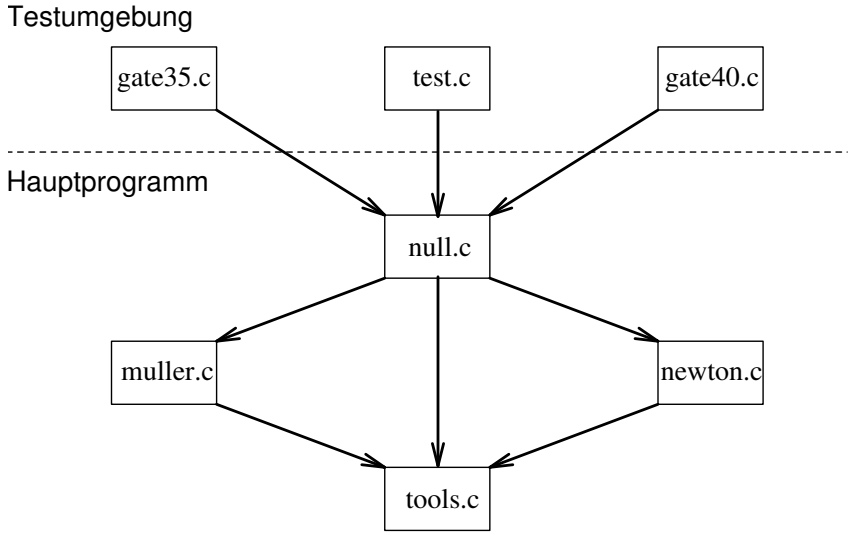

Abbildung 4: Abhängigkeit der einzelnen Files von 'rootsf'

zur Durchführung komplexer Operationen. Von zeitkritischen Ausnahmen abgesehen werden sämtliche komplexen Rechnungen mit Hilfe dieser Operationen durchgeführt. Die komplexen Zahlen werden in 'header.h' mit Hilfe einer Stuktur festgelegt und als ein neuer Zahlentyp 'dcomplex' definiert mit der Anweisung:

typedef struct DCOMPLEX { double r, i; } dcomplex;

Sämtliche Rechnungen erfolgen mit doppelter Maschinengenauigkeit, was das 'd' im Zahlentyp 'dcomplex' verdeutlichen soll. Für Gleitkommarechnung ergibt sich somit nach dem IEEE-P754-Floating-Point-Standard eine Wortbreite von 64 bit pro Real- bzw. Imaginärteil, d.h. also ein Speicherplatzbedarf von 16 Byte pro komplexer Zahl. Der Zahlenbereich ist dann auf $\approx 10^{\pm 308}$ festgelegt. Die Maschinengenauigkeit liegt bei der hier verwendeten HP-Apollo-Workstation bei $\approx 2.2 \cdot 10^{-16}$. Das Programm ist so aufgebaut, daß diese Zahlenwerte aus dem Standard-Includefile 'float.h' ausgelesen werden und davon abhängig je nach verwendetem Rechner die Abbruchschranken und sonstigen Grenzen für 'rootsf' ermittelt werden. Dieses Vorgehen sollte ein hohes Maß an Portabilität ermöglichen.

8.3 Die Hauptroutine null()

8.3.1 Struktur und Funktionsweise

Die Struktur der Funktion null() ist in Abbildung 5 dargestellt.

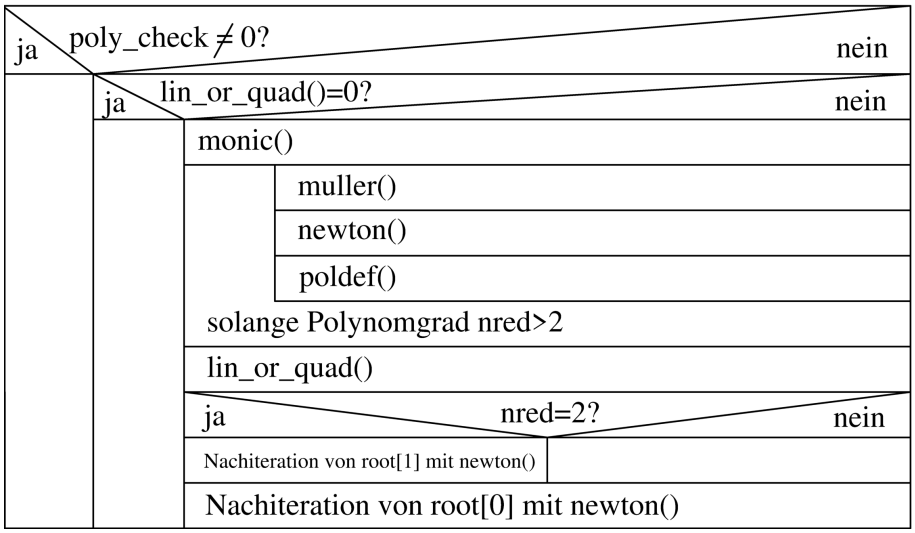

Abbildung 5: Struktur der Funktion null()

Zunächst werden in der Funktion poly_check() die übergebenen Koeffizienten auf ihre formale Richtigkeit hin überprüft:

1. Fehler bei negativem Polynomgrad, Fehlercode = 1
2. Fehler, wenn Koeffizientenvektor = Nullvektor, Fehlercode = 2
3. Fehler, wenn Koeffizientenvektor = Konstante, Fehlercode = 3
4. mögliche Nullstellen bei Null werden abgespalten und der Polynomgrad entsprechend reduziert, kein Fehler $\Rightarrow$ Fehlercode = 0

Bei auftretendem Fehler wird null() mit dem entsprechenden Fehlercode beendet, ansonsten werden die Nullstellen des gegebenenfalls bereits reduzierten Polynomes in lin_or_quad() bei einem Polynomgrad kleiner oder gleich 2 mit den entsprechenden Formeln berechnet und null() beendet. Ist der Grad größer als 2, wird in monic() der Koeffizient der höchsten Potenz auf 1 normiert. Dies hat den Sinn, bei sehr kleinen oder sehr großen Koeffizienten Unterlauffehler beziehungsweise Überlauffehler zu beseitigen. Dies sollte aber nicht darüber hinwegtäuschen, daß bei einer starken Streuung der Koeffizienten diese Maßnahme keinerlei Verbesserung der Anfälligkeit gegen derartige Fehler bringt, sondern ausschließlich

die Genauigkeit der Eingangskoeffizienten verschlechtert.

In der Hauptschleife wird dann mit dem reduzierten Polynom mit Hilfe des Muller-Verfahrens muller() eine Anfangsnäherung gesucht, die im folgenden Newton-Verfahren newton() mit dem Originalpolynom nachiteriert wird, wobei newton() außerdem einen Schätzwert für die Genauigkeit der bestimmten Nullstelle liefert. In der Funktion poldef() wird dann die gefundene Nullstelle beziehungsweise bei reellen Koeffizienten gegebenenfalls das konjugiert komplexe Nullstellenpaar abgespalten. Ist der Grad des Restpolynomes kleiner oder gleich 2, wird die Hauptschleife beendet.

Eine Anfangsnäherung der Nullstellen des verbleibenden Polynomes wird mit lin_or_quad() mit Hilfe der Formeln für lineare oder quadratische Gleichungen berechnet, in newton() wird wieder mit dem Originalpolynom nachiteriert und im Anschluß null() beendet.

8.3.2 Numerische Überlegungen

Es soll hier noch einmal auf die im vorherigen Abschnitt angesprochene Vorgehensweise bezüglich Nachiteration und Polynomdeflation kurz eingegangen werden.

Die hier vorgenommene Nachiteration mit Hilfe des Originalpolynomes verbessert die Genauigkeit schlecht konditionierter Nullstellen sicherlich nicht (siehe [13]). Dies ist auch deutlich an den Ergebnissen zu erkennen, die sich aus Tests mit mehrfachen Nullstellen ergaben, die allesamt schlecht konditioniert sind. Trotzdem wurde das Originalpolynom zur Nachiteration herangezogen, weil es eine Genauigkeitsverbesserung in all den Fällen sicherstellt, in denen durch die Polynomdeflation eine Verschlechterung der Kondition auftritt.

Wie in [13] beschrieben ist bei der Polynomdeflation weiterhin wichtig, 'daß die günstigste Reihenfolge, in der die Nullstellen berechnet werden sollten, unabhängig von der Kondition der Nullstellen ist.' Werden die Nullstellen in einer betragsmäßig aufsteigenden Reihenfolge bestimmt, 'so ergeben sich alle Nullstellen mit einer Genauigkeit, die im wesentlichen von ihrer Kondition bestimmt ist und nicht von der Genauigkeit der vorher abdividierten Nullstellen.' Aus diesem Grunde ist auch das Muller-Verfahren in der dargestellten Art als Iterationsverfahren gewählt worden, weil es i.a. die Nullstellen in betragsmäßig aufsteigender Reihenfolge liefert.

Die im Komplexen anfallenden Punkt- und Strichoperationen bei der Polynomdeflation sind aus Zeitgründen durch Makrooperationen ersetzt worden, die sich

im Includefile 'header.h' befinden. Ebenso ist die Funktionswertberechnung in der Funktion fdvalue() aufgrund eines angefertigten Zeitdiagrammes der einzelnen Funktionen durch Makroaufrufe ersetzt worden. Durch Einsetzen von Makrooperationen an den entsprechenden Stellen wurde eine Geschwindigkeitsverbesserung der Gesamtlaufzeit von $\approx 50\%$ erreicht.

8.3.3 Bedeutung der im Programm auftretenden Größen

Im folgenden werden die in null() auftretenden, signifikanten Variablen erläutert, um einen schnelles Verständnis des vorliegenden Programmes zu erleichtern.

Eingabegröße	Bedeutung
p[]	Koeffizientenvektor des Originalpolynomes der Dimension 1x$(N+1)$ mit $N+1$-Elementen $p[0] = p_0, \ldots, p[N] = p_N$ des Polynomes $P(x) = p_0 + p_1 x + p_2 x^2 + \ldots + p_{N-1} x^{N-1} + p_N x^N$
pred[]	Hilfspolynom mit gleicher Form wie p[], wobei zu Beginn $P_{red}(x) = P(x)$ gilt
n	Zeiger auf Polynomgrad N
flag	flag $= 0 \Rightarrow$ reelle Koeffizienten flag $= 1 \Rightarrow$ komplexe Koeffizienten, bzw. es wird mit komplexem Algorithmus gerechnet. Sind die Koeffizienten reell, so sind alle Imaginärteile der Koeffizienten zu Null zu setzen

interne Größe	Bedeutung
nred	Grad des reduzierten Polynomes
ns	mit dem Muller-Verfahren bestimmte Anfangsnäherung der Nullstelle
newerr	geschätzter Fehler der aktuell bestimmten Nullstelle
error	zeigt Fehler in der Eingabe von p[] an
red	Deflation um red Nullstellen pro Iterationsschritt
diff	Anzahl der Nullstellen bei 0

Ausgabegröße	Bedeutung
root[]	Vektor der bestimmten Nullstellen
maxerr	Zeiger auf den geschätzten, maximalen Fehler aller bestimmten Nullstellen

8.4 Das Muller-Verfahren muller()

8.4.1 Struktur und Funktionsweise

Die Grundstruktur des Muller-Verfahrens, realisiert in der Funktion muller(), geht aus Abbildung 6 hervor.

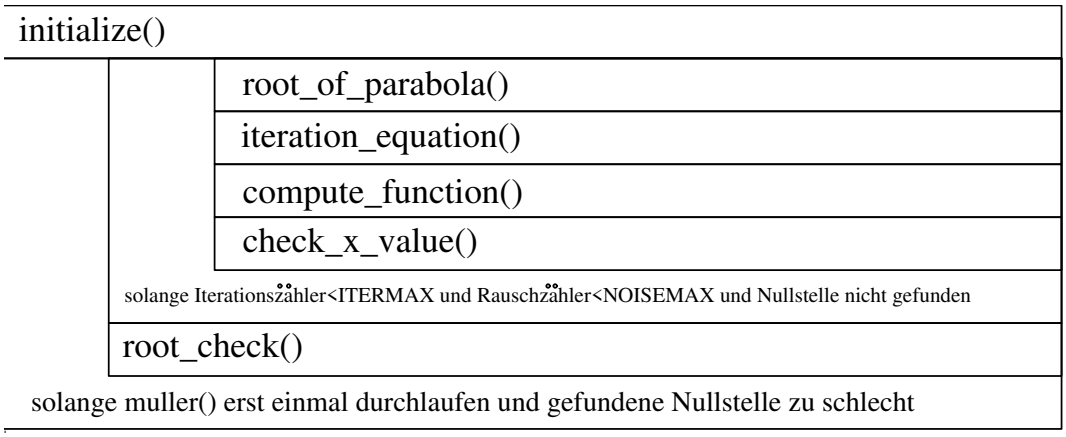

Abbildung 6: : Struktogramm von muller()

Zunächst werden in der Funktion initialize() die in [3] vorgeschlagenen Startwerte $x^{(0)}, x^{(1)}$ und $x^{(2)}$ und die zugehörigen künstlichen Funktionswerte f_0, f_1 und f_2 festgelegt, sowie das zugehörige h_1, h_2 und q_2 bestimmt. Der Startwert der besten Näherung der Nullstelle ist beliebig und hier willkürlich auf Null festgesetzt worden.

Das eigentliche Muller-Verfahren ist nun mit Hilfe von zwei ineinander verschachtelten Schleifen realisiert worden. Die äußere Schleife wird benötigt, wenn die von der inneren Schleife bestimmte Nullstelle so schlecht ist, daß bei einer Übergabe dieser Nullstelle an das Newton-Verfahren ein Funktionsüberlauf bei der Funktionswertberechnung und somit ein Abbruch der Routine zu erwarten ist. Für diese Überprüfung dient die Funktion root_check(). Beim zweiten Durchlauf der inneren Schleife wird dann mit veränderten Startwerten $x^{(0)}, x^{(1)}$ und $x^{(2)}$ gearbeitet.

Der Abbruch der Routine ohne äußere Schleife trat beispielsweise bei dem Kreisteilungspolynom $P(x) = x^{2000} - 1$ auf. Auch ein Ersetzen der künstlichen Funktionswerte durch die exakten beim Start von muller() brachte hierbei keine Abhilfe.

In der inneren Schleife werden mit der Funktion root_of_parabola() zunächst die Nullstellen der approximierten Parabel berechnet und die betragsmäßig kleinere zur Weiterverarbeitung in q_2 gespeichert.
Bei der Durchführung des anschließenden Iterationsschrittes mit der Funktion iteration_equation() wird zunächst nur die Änderung vom neuen zum alten x-Wert berechnet. Ist der relative Betrag dieser Änderung bezüglich der Änderung im vorhergehenden Iterationsschritt größer als MAXDIST, so wird betragsmäßig die Änderung beschränkt auf einen Kreis um den Ursprung mit Radius MAXDIST, wobei die Richtung der Änderung beibehalten wird. Erst dann wird der Iterationsschritt ausgeführt und das neue $x^{(2)}$ berechnet. Eine Begründung dieses Vorgehens ist im folgenden Abschnitt angegeben.

Anschließend wird die Funktion compute_function() abgearbeitet, deren Struktogramm Abbildung 7 wiedergibt.

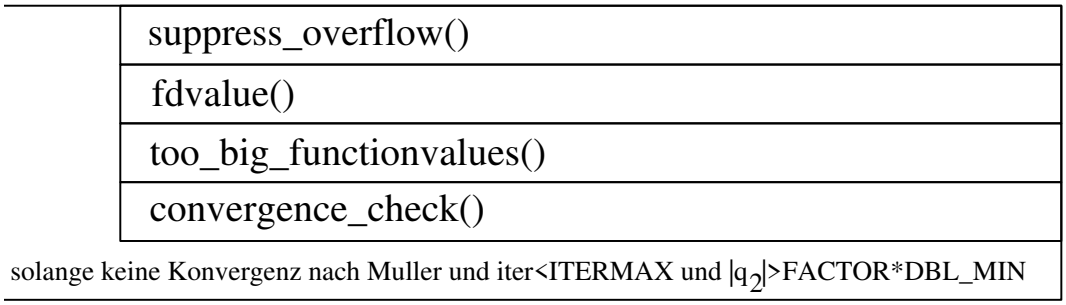

Abbildung 7: Struktogramm von compute_function()

Bevor nun eine Funktionswertberechnung $f_2 = P(x^{(2)})$ durchgeführt wird, schätzt die Funktion suppress_overflow() den Funktionswert f_2 ab, um einen möglichen Funktionsüberlauf von f_2 zu vermeiden. Dieser tritt aber genau dann auf, wenn gilt:

$$|P_{red}(x^{(2)})| = |p_{nred} \cdot (x^{(2)})^{nred} + \ldots + p_{1red}x^{(2)} + p_{0red}| > \text{DBL\_MAX}$$

Bei großem $x^{(2)}$ wird der Betrag $|(x^{(2)})^{nred}|$ den größten Einfluß auf den linken Ausdruck haben. Bei Wahl einer geeigneten Schranke a wird also bei

$$|(x^{(2)})^{nred}| > a \tag{17}$$

ein Überlauf auftreten. Logarithmiert man die Ungleichung (17), so ergibt sich mit der Wahl von a zu $a = 10^{\text{BOUND6}}$:

$$nred \cdot \log(|x^{(2)}|) > \text{BOUND6} \tag{18}$$

Tritt dieser Fall ein, so halbiert die Funktion die Strecke zwischen neuem und altem $x^{(2)}$ und setzt den daraus resultierenden Wert als neuen Wert für $x^{(2)}$ fest. Nun wird wieder die Abschätzung (18) überprüft und ggf. ein neues $x^{(2)}$ bestimmt usw. Der aktuelle Wert für $x^{(2)}$ 'wandert' also ggf. wieder in Richtung seines Wertes im vorhergehenden Iterationsschritt, in dem f_2 noch berechnet werden konnte.
Im Anschluß führt die Funktion fdvalue() die Funktionswertberechnung $f_2 = P_{red}(x^{(2)})$ durch, die Funktion too_big_functionvalues() berechnet das Betragsquadrat $|f_2|^2$. Droht hierbei ein Funktionsüberlauf, so wird $|f_2|^2$ aus der Summe aus Real- und Imaginärteil abgeschätzt. Die Funktion muller() arbeitet ausschließlich mit dem Betragsquadrat von f_2, welches gleichbedeutend wie der Betrag von f_2 verwendet werden kann. Es wird hierbei lediglich der Aufruf der Wurzelfunktion eingespart.
In der Funktion convergence_check() ist der Vorschlag von Muller zur Sicherstellung der Konvergenz bei den geg. Startwerten realisiert. Gilt hierbei

$$\left|\frac{P_{red}^2(x^{(2)})}{P_{red}^2(x^{(1)})}\right| > 100$$

so werden nach Muller q_2 und h_2 halbiert und $x^{(2)}$ neu berechnet. Liegt bereits Konvergenz nach Muller vor, so wird die Funktion compute_function() beendet.

Die Funktion check_x_value() (Abbildung 6) überprüft zunächst, wie sich die Funktionswerte des so bestimmten, neuen Näherungswertes $x^{(2)}$ ändern. Gilt hierbei

$$0.99 \leq \left|\frac{P(x^{(2)})}{P(x^{(1)})}\right|^2 \leq 1.01,$$

so werden die Werte für q_2 und h_2 einmalig verdoppelt. Ergibt sich dagegen, daß $|f_2|^2 < |f_{b2}|^2$, wobei f_{b2} der Funktionswert der bisher besten Approximation ist, so wird die beste Approximation dem aktuellen Wert gleichgesetzt. Gilt außerdem:

$$\left|\frac{x^{(2)} - x^{(1)}}{x^{(2)}}\right| < \epsilon,$$

dann ist die Abbruchbedingung mit der Abbruchschranke ϵ erfüllt und muller() gibt den aktuellen Wert an null() zurück.

8.4.2 Numerische Überlegungen

Dieser Abschnitt beinhaltet Überlegungen zur Vorgehensweise und zur Festlegung der Grenzen im Muller-Verfahren muller().
Die in der Funktion iteration_equation() durchgeführte Begrenzung der relativen

Änderung von einem Iterationswert zum neuen $x^{(2)}$ hat sich als zweckmäßig erwiesen, da Fälle aufgetreten sind, in denen ohne diese Begrenzung sich das neue $x^{(2)}$ sehr weit von der gesuchten Nullstelle entfernt hat. Das Annähern an die entsprechende Nullstelle war zwar immer noch erkennbar, jedoch hätte es zu viele Iterationsschritte benötigt, um wieder in die Nähe der Nullstelle zu gelangen. Das Verfahren brach somit bei der festgelegten, maximalen Zahl von Iterationsschritten vorzeitig ab, und es ergaben sich aufgrund der schlechten Näherung Überlauffehler im folgenden Newtonverfahren.

Weiterhin wurde ein Rauschzähler eingeführt, der aktiviert wird, wenn die Beziehung

$$\left| \frac{|x^b| - |x^{(2)}|}{|x^b|} \right| < \text{NOISESTART}$$

gilt, d.h. also wenn die Differenz der Beträge zwischen der bisher besten Näherung x^b und der aktuellen Näherung $x^{(2)}$ bezüglich des Betrages der bisher besten Näherung kleiner als eine Grenze NOISESTART ist. Ergibt sich in NOISEMAX aufeinanderfolgenden Iterationsschritten keine Verbesserung der besten Näherung x^b, so bricht muller() vorzeitig ab. Anderenfalls, bei einer Verbesserung von x^b, wird der Rauschzähler in der Funktion check_x_value() auf null zurückgesetzt.

Die Grenzen wie Iterationszähler, Rauschzähler usw. wurden weitgehend empirisch festgelegt, wobei versucht wurde, die Funktion zeitoptimal sowie abbruchsicher zu gestalten. Die Schranke CONVERGENCE in der Funktion convergence_check() wurde hierbei aus [3] übernommen, die eine Schranke zur Gewährleistung der Konvergenz nach Muller darstellt.

8.5 Das Newton-Verfahren newton()

8.5.1 Struktur und Funktionsweise

Die Struktur des Newton-Verfahrens newton() geht aus Abbildung 8 hervor.

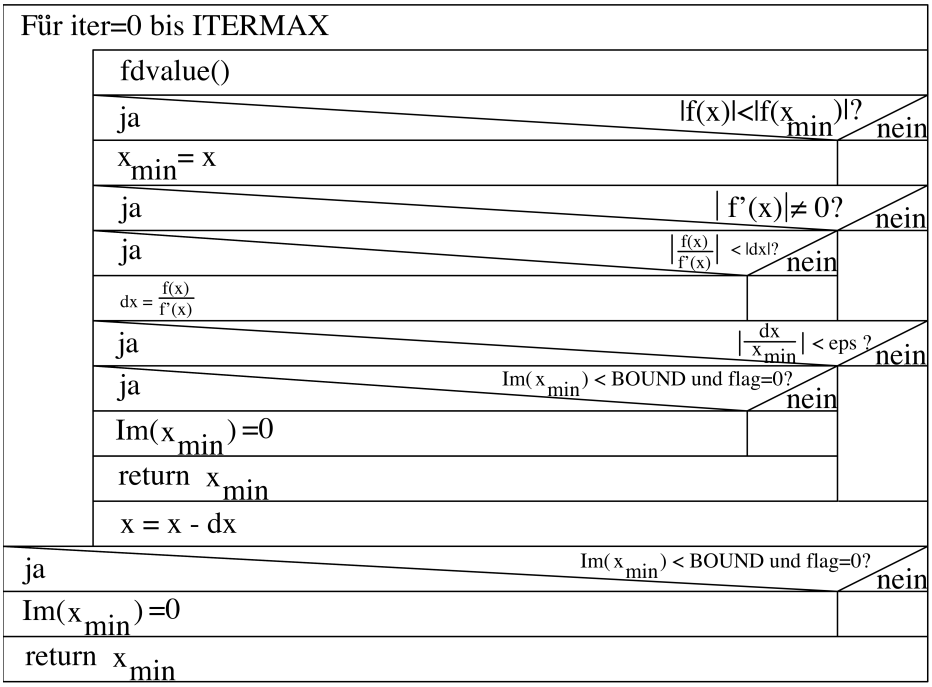

Abbildung 8: Struktogramm der Funktion newton()

Zunächst werden in der Hauptschleife an der Stelle x der Funktionswert f und die zugehörige Ableitung f' gebildet. Ist der aktuelle Funktionswert $f(x)$ kleiner als der Funktionswert der bisher besten Näherung x_{min}, so ist die aktuelle Näherung die beste.
Falls nun der Ausdruck

$$\left|\frac{f(x)}{f'(x)}\right| \tag{19}$$

mit der aktuellen Näherung x kleiner ist als der Betrag von dx aus dem letzten Iterationsschritt, so wird der Ausdruck (19) als neues dx verwendet. Anderenfalls wird das alte dx aus dem letzten Iterationschritt übernommen. Hiermit wird das 'Ausreißen' der Iteration aus der Umgebung der Nullstelle vermieden, wenn sich z.B. ein Sattelpunkt in der näheren Umgebung der Nullstelle befindet.
Ist nun die neue Änderung

$$dx < \epsilon, \tag{20}$$

so ist die Nullstelle gefunden und newton() wird beendet. Anderenfalls wird der Iterationsschritt ausgeführt und die Hauptschleife von neuem durchlaufen. Ist nach ITERMAX Schritten die Abbruchbedingung (20) noch nicht erfüllt, so wird der bis dorthin beste x-Wert zurückgegeben.
Weiterhin liefert die Funktion newton() einen Schätzwert ξ_{gesch} für den Fehler der gefundenen Näherung der Nullstelle nach der Beziehung

$$\xi_{gesch} = \left| \frac{dx}{x_{min}} \right| \tag{21}$$

Die Gleichung (21) ergibt sich aus der Annahme, daß für die x-Werte in einer näheren Umgebung der gesuchten Nullstelle x_0 die Iterationsschrittweite $|dx|$ immer weiter abnimmt (Veranschaulichung siehe auch Bild 1). Unter Vernachlässigung der folgenden Iterationsschrittweiten liefert dann $|dx|$ an der Stelle der besten Näherung der Nullstelle eine Abschätzung des absoluten Fehlers der gefundenen Näherung. Der relative Fehler der Näherung nach Gleichung (21) folgt dann aus der Division durch den Betrag der gefundenen, besten Näherung x_{min}.

8.5.2 Numerische Überlegungen

Bei Rückgabe der besten Näherung der Nullstelle wird entschieden, ob die Nullstelle als reell oder komplex angenommen werden soll. Ist $Im(x_{min}) <$ BOUND, so wird der Imaginärteil zu Null gesetzt, die Nullstelle also als reell festgesetzt, falls im *reellen Modus* ($flag = 0$) gearbeitet wird. Hierin liegt ein prinzipielles Problem bei der Berechnung von Nullstellen, nämlich die Unterscheidung von mehrfachen Nullstellen zu nahe zusammenliegenden Nullstellen. In dieser Routine wurde die Schranke BOUND auf die halbe Stellenzahl der Maschinengenauigkeit festgesetzt. Soll dieses Unterscheidungskriterium nicht verwendet werden, so kann auch im *komplexen Modus*, d.h. mit $flag = 1$, gearbeitet werden. Allerdings wird dann jede Nullstelle einzeln berechnet und somit kein konjugiert komplexes Nullstellenpaar mehr abgespalten.

Das Unterscheidungskriterium wird sinnvollerweise erst nach Beendigung des Newton-Verfahrens festgelegt. Träfe man eine Entscheidung über die Reellwertigkeit einer Nullstelle bereits nach dem Muller-Verfahren, so wäre für das gesamte Newton-Verfahren bei reellem Startwert die Reellwertigkeit der Nullstelle festgelegt, obwohl es sich durchaus um eine komplexe Nullstelle handeln könnte. Da sich die Näherung des Muller-Verfahrens i.a. noch weiter von der tatsächlichen Nullstelle entfernt befindet als die Nullstellennäherung des Newton-Verfahrens, vergrößerte sich dadurch der Näherungsfehler.

9 Ergebnisse und Vergleich mit anderen Testverfahren

Das Programm 'rootsf' ist nach Fertigstellung mit den beiden Nullstellenberechnungsprogrammen 'roots' und 'rootsj' verglichen worden.
Die Funktion 'roots' bildet zur Nullstellenberechnung die zum Polynom gehörende Begleitmatrix und berechnet hiermit unter Verwendung von Eigenwertroutinen aus dem Programmpaket EISPACK die zugehörigen Eigenwerte, die eine Näherung der Polynomnullstellen darstellen. Die Funktion 'rootsj' ist ein auf dem Verfahren von Jenkins und Traub basierendes Programm und gehört ebenso wie 'roots' zu den wenigen Programmen, die auch zur Bestimmung von Nullstellen schlecht konditionierter Polynome geeignet sind.
Um die Testläufe mit den unterschiedlichen Testpolynomen nach Jenkins und Traub [4] und auch für die Kreisteilungspolynome zu systematisieren, wurde 'rootsf' als Funktion über die Erzeugung eines MEX-Files an Matlab Version 4.0 auf der HP-Apollo-Workstation angebunden. Das zweite, für Matlab externe Programm 'rootsj' speicherte seine Ergebnisse in ein Textfile, das dann von Matlab aus geladen und ausgewertet worden ist. Zur Systematisierung der Auswertung sind Matlab-Files erstellt worden, welche die drei Nullstellenberechnungsfunktionen sukzessive aufrufen, aus den Ergebnissen die relativen Fehler berechnen, die für die Nullstellenberechnung benötigte Zeit auswerten und die Ergebnisse in einer Tabelle zusammenfassen. Ebenso in die Tabelle aufgenommen wurde der von 'rootsf' erzeugte, geschätzte Fehler der Lösung.

9.1 Testverfahren nach Jenkins und Traub

Die in diesem Abschnitt vorgestellten Testpolynome und die zugehörigen Ergebnisse bezüglich der drei Programme 'roots','rootsj' und 'rootsf' basieren im wesentlichen auf den von Jenkins und Traub vorgeschlagenen Testpolynomen (siehe [4]). Ebenfalls aus [4] übernommen wurden die Kriterien, nach denen die Nullstellenberechnungsprogramme überprüft wurden, nämlich:

- Test des Abbruchkriteriums
- Test auf Konvergenzschwierigkeiten
- Test auf Unterscheidung zwischen benachbarten und mehrfachen Nullstellen
- Test der Stabilität der Deflation

Auf die außerdem empfohlenen Tests mit zufällig erzeugten Polynomen wurde im folgenden verzichtet.

Der in den Tabellen angegebene, reale Fehler ξ_{real} ergibt sich aus der Beziehung

$$\xi_{real} = \max \left| \frac{\alpha_\nu - \hat{\alpha}_\nu}{\alpha_\nu} \right|. \tag{22}$$

Hierbei sind die Werte von α_ν die exakten Nullstellen und die Werte von $\hat{\alpha}_\nu$ die Näherungen der entsprechenden Nullstellen des jeweiligen Programmes. Der reale Fehler ξ_{real} ergibt sich dann aus dem Maximum aller auftretenden, relativen Fehler. Der tabellierte, geschätzte Fehler des Programmes 'rootsf' folgt aus der Gleichung (21).

9.1.1 Das Abbruchkriterium

Zum Test des Abbruchkriteriums wurde das Polynom

$$P_1(x) = B(x - A)(x + A)(x - 1) \tag{23}$$

verwendet. Test des Abbruchkriteriums bedeutet hierbei zu überprüfen, ob eine konvergierende Sequenz von Iterationen dann abbricht, wenn bei der Polynomauswertung der Einfluß von Rundungsfehlern dominant wird.
Im Polynom $P_1(x)$ wird durch großes und kleines A der Einfluß von betragsmäßig großen und kleinen Nullstellen auf das Abbruchkriterium überprüft, durch großes und kleines B der Einfluß von betragsmäßig großen und kleinen Polynomkoeffizienten.
Die Ergebnisse zeigt Tabelle (2).

		realer Fehler			gesch. Fehler	Zeit in s		
A	B	roots	rootsj	rootsf	rootsf	roots	rootsj	rootsf
$1 \cdot 10^{-10}$	$1 \cdot 10^{-10}$	0	0	0	0	0.01	0.05	0.02
$1 \cdot 10^{-10}$	$1 \cdot 10^{10}$	0	0	0	0	0.01	0.02	0.01
$1 \cdot 10^{10}$	$1 \cdot 10^{-10}$	0	$4.648 \cdot 10^{-3}$	0	$3.584 \cdot 10^{-17}$	0.01	0.02	0.01
$1 \cdot 10^{10}$	$1 \cdot 10^{10}$	0	$4.648 \cdot 10^{-3}$	0	$3.584 \cdot 10^{-17}$	0.00	0.03	0.01

Tabelle 2: Polynom $P_1(x) = B(x - A)(x + A)(x - 1)$

Die besten Ergebnisse hinsichtlich Zeit und Genauigkeit liefern hierbei 'roots' und 'rootsf'. Beide Programme finden fehlerlos im Rahmen der Rechengenauigkeit die gesuchten Nullstellen in etwa gleicher Zeit. Auffallend sind hierbei die um ungefähr zwei Mal längere Rechendauer und der große relative Fehler von $\approx 10^{-3}$ des Programmes 'rootsj' in der 3. und 4. Zeile. Von allen drei Funktionen liefert es hierbei die schlechtesten Ergebnisse.

Als weiteres Polynom zum Testen des Abbruchkriteriums fand das Polynom

$$P_2(x) = \prod_{\nu=0}^{N} (x - 10^{-\nu}) \tag{24}$$

Verwendung. In dem durch (24) angegebenen Polynom liegen also für größer werdendes N immer mehr Nullstellen sehr dicht bei 0. Auffallend an den Ergebnissen aus Tabelle 3 ist, daß 'rootsf' für $N = 7$ wesentlich mehr Zeit benötigt als die

	realer Fehler			gesch. Fehler	Zeit in s		
N	roots	rootsj	rootsf	rootsf	roots	rootsj	rootsf
5	$6.939 \cdot 10^{-16}$	$1.735 \cdot 10^{-17}$	$1.735 \cdot 10^{-17}$	$1.307 \cdot 10^{-16}$	0.01	0.01	0.03
7	$1.388 \cdot 10^{-15}$	$1.735 \cdot 10^{-16}$	$1.735 \cdot 10^{-16}$	$1.576 \cdot 10^{-16}$	0.01	0.02	0.09

Tabelle 3: $P_2(x) = \prod_{\nu=0}^{N} (x - 10^{-\nu})$

anderen Funktionen. Das liegt daran, daß das Muller-Verfahren für die kleinsten Nullstellen bei 10^{-7} und 10^{-6} je Nullstelle zweimal vollständig durchlaufen wird, ohne eine Näherung gemäß der Abbruchbedingung zu erhalten.

9.1.2 Das Konvergenzkriterium

Zur Überprüfung des Konvergenzkriteriums wird in [4] das Polynom

$$P_3(x) = \prod_{\nu=0}^{N} (x - \nu) \tag{25}$$

vorgeschlagen. Das auf den ersten Blick numerisch einfach erscheinende Polynom hat im Vergleich zu anderen Polynomen verhältnismäßig schlecht konditionierte Nullstellen (siehe auch [13]) und ist damit gut zum Test der Konvergenz von Nullstellenberechnungsprogrammen geeignet. Die Ergebnisse sind in Tabelle 4 wiedergegeben.
Es wurden der Fehler und die Zeit nur bis zum Grad $N = 15$ untersucht, da bei höheren Graden die Eingangskoeffizienten nicht mehr genau dargestellt werden konnten und sich somit bereits ein Fehler in den Eingangsdaten ergeben hätte. Die realen Fehler aller Routinen stimmen sowohl untereinander, als auch mit dem geschätzten Fehler weitgehend überein. Auffallend ist jedoch der erheblich größere Zeitaufwand der Funktion 'rootsf' ab dem Grad $N = 10$. Die Ursache hierfür liegt wiederum darin, daß beispielsweise für den Grad $N = 15$ alle ITERMAX Iterationsschritte in muller() für die Nullstellen 3, 6 und 9 durchlaufen werden, ohne eine ausreichende Anfangsnäherung derart zu liefern, daß der Rauschzähler die Routine beenden könnte.

	realer Fehler			gesch. Fehler	Zeit in s		
N	roots	rootsj	rootsf	rootsf	roots	rootsj	rootsf
1	0	0	0	$2.220 \cdot 10^{-16}$	0.01	0.03	0.00
2	0	$2.220 \cdot 10^{-16}$	0	$2.220 \cdot 10^{-16}$	0.00	0.01	0.00
3	$5.921 \cdot 10^{-16}$	0	$7.401 \cdot 10^{-17}$	$4.534 \cdot 10^{-25}$	0.01	0.03	0.00
4	$1.021 \cdot 10^{-14}$	0	$1.776 \cdot 10^{-15}$	$1.776 \cdot 10^{-15}$	0.01	0.02	0.01
5	$5.003 \cdot 10^{-14}$	$2.013 \cdot 10^{-14}$	$1.036 \cdot 10^{-15}$	$1.776 \cdot 10^{-14}$	0.01	0.02	0.01
6	$2.236 \cdot 10^{-13}$	$3.497 \cdot 10^{-14}$	$5.695 \cdot 10^{-14}$	$1.089 \cdot 10^{-13}$	0.01	0.04	0.01
7	$8.777 \cdot 10^{-13}$	$1.561 \cdot 10^{-13}$	$6.370 \cdot 10^{-13}$	$2.425 \cdot 10^{-13}$	0.01	0.01	0.02
8	$1.180 \cdot 10^{-11}$	$5.218 \cdot 10^{-13}$	$1.217 \cdot 10^{-12}$	$1.929 \cdot 10^{-12}$	0.02	0.01	0.02
9	$1.062 \cdot 10^{-10}$	$3.949 \cdot 10^{-12}$	$4.937 \cdot 10^{-12}$	$7.370 \cdot 10^{-12}$	0.02	0.03	0.04
10	$4.365 \cdot 10^{-11}$	$7.854 \cdot 10^{-11}$	$1.805 \cdot 10^{-11}$	$4.312 \cdot 10^{-11}$	0.02	0.02	0.05
11	$5.448 \cdot 10^{-10}$	$3.373 \cdot 10^{-11}$	$1.534 \cdot 10^{-10}$	$1.772 \cdot 10^{-10}$	0.02	0.02	0.06
12	$6.751 \cdot 10^{-9}$	$8.073 \cdot 10^{-10}$	$2.196 \cdot 10^{-10}$	$2.205 \cdot 10^{-9}$	0.02	0.02	0.08
13	$1.121 \cdot 10^{-7}$	$1.550 \cdot 10^{-8}$	$5.197 \cdot 10^{-9}$	$4.507 \cdot 10^{-9}$	0.03	0.01	0.10
14	$4.421 \cdot 10^{-8}$	$3.770 \cdot 10^{-8}$	$1.497 \cdot 10^{-8}$	$5.249 \cdot 10^{-8}$	0.03	0.02	0.12
15	$3.848 \cdot 10^{-7}$	$2.496 \cdot 10^{-7}$	$9.540 \cdot 10^{-8}$	$5.714 \cdot 10^{-8}$	0.04	0.03	0.13

Tabelle 4: Polynome $P_3(x) = \prod_{\nu=0}^{N} (x - \nu)$

9.1.3 Mehrfache oder benachbarte Nullstellen

Mehrfache oder nahe zusammenliegende Nullstellen bereiten den meisten Nullstellenberechnungsprogrammen Probleme, weil sie schlecht konditioniert sind; die Grenzgenauigkeit von Nullstellen mit der Vielfachheit m_ν beträgt ungefähr

$$\varepsilon^{\frac{1}{m_\nu}}, \tag{26}$$

wobei ε die Maschinengenauigkeit ist. Entsprechend erhöht sich die Laufzeit, da die Programme in der Regel bestrebt sind, die Nullstellen bis auf Maschinengenauigkeit zu bestimmen, d.h. die Maximalzahl der Iterationsschritte durchlaufen wird. Zum Test der Programme wurden die Polynome

$$\begin{aligned}
P_4(x) &= (x-0.1)^3(x-0.5)(x-0.6)(x-0.7) \\
P_5(x) &= (x-0.1)^4(x-0.2)^3(x-0.3)^2(x-0.4) \\
P_6(x) &= (x-0.1)(x-1.001)(x-0.998)(x-0.99999) \\
P_7(x) &= (x-0.001)(x-0.01)(x-0.1)(x-0.1+A\cdot j)(x-0.1-A\cdot j)(x-1)(x-10)
\end{aligned}$$

$$P_8(x) = (x+1)^5$$
$$P_9(x) = (x^{10} - 10^{-20})(x^{10} + 10^{20})$$

verwendet. Die Ergebnisse gehen aus den Tabellen 5 bis 7 hervor.

	realer Fehler			gesch. Fehler	Zeit in s		
ν	roots	rootsj	rootsf	rootsf	roots	rootsj	rootsf
4	$2.124 \cdot 10^{-5}$	$3.068 \cdot 10^{-6}$	$8.771 \cdot 10^{-6}$	$3.248 \cdot 10^{-6}$	0.01	0.04	0.05
5	$7.076 \cdot 10^{-4}$	$2.626 \cdot 10^{-5}$	$3.988 \cdot 10^{-4}$	$1.492 \cdot 10^{-4}$	0.03	0.03	0.09
6	$1.529 \cdot 10^{-5}$	$2.918 \cdot 10^{-4}$	$1.002 \cdot 10^{-5}$	$4.116 \cdot 10^{-6}$	0.01	0.03	0.03

Tabelle 5: Polynome $P_{\nu=4}(x), P_{\nu=5}(x), P_{\nu=6}(x)$

	realer Fehler			gesch. Fehler	Zeit in s		
A	roots	rootsj	rootsf	rootsf	roots	rootsj	rootsf
0	$1.627 \cdot 10^{-5}$	$3.923 \cdot 10^{-8}$	$6.978 \cdot 10^{-6}$	$2.406 \cdot 10^{-6}$	0.01	0.03	0.04
$1 \cdot 10^{-10}$	$1.627 \cdot 10^{-5}$	$3.923 \cdot 10^{-8}$	$6.978 \cdot 10^{-6}$	$2.406 \cdot 10^{-6}$	0.02	0.01	0.03
$1 \cdot 10^{-9}$	$1.363 \cdot 10^{-5}$	$3.923 \cdot 10^{-8}$	$8.620 \cdot 10^{-6}$	$1.841 \cdot 10^{-6}$	0.02	0.02	0.03
$1 \cdot 10^{-8}$	$1.601 \cdot 10^{-5}$	$3.923 \cdot 10^{-8}$	$4.983 \cdot 10^{-6}$	$1.145 \cdot 10^{-6}$	0.01	0.02	0.03
$1 \cdot 10^{-7}$	$8.674 \cdot 10^{-6}$	$3.923 \cdot 10^{-8}$	$4.140 \cdot 10^{-6}$	$3.632 \cdot 10^{-6}$	0.01	0.04	0.03
$1 \cdot 10^{-6}$	$1.046 \cdot 10^{-5}$	$3.968 \cdot 10^{-8}$	$2.918 \cdot 10^{-6}$	$1.289 \cdot 10^{-6}$	0.02	0.02	0.04

Tabelle 6: Polynom $P_7(x)$

	realer Fehler			gesch. Fehler	Zeit in s		
ν	roots	rootsj	rootsf	rootsf	roots	rootsj	rootsf
8	$9.625 \cdot 10^{-4}$	$1.380 \cdot 10^{-13}$	$6.535 \cdot 10^{-4}$	$4.454 \cdot 10^{-4}$	0.02	0.02	0.04
9	$8.527 \cdot 10^{-16}$	$3.268 \cdot 10^{-15}$	$1.421 \cdot 10^{-16}$	$1.933 \cdot 10^{-16}$	0.07	0.03	0.05

Tabelle 7: Polynome $P_{\nu=8}(x), P_{\nu=9}(x)$

Die Ergebnisse der Tabelle 5 sind für alle Routinen in etwa gleich. Durch die Vielfachheit der Nullstellen und durch die Nullstellen von P_6 nahe der 1 haben alle drei Programme große relative Fehler mit $\approx 10^{-5}$. Auffallend ist dabei die hohe Rechenzeit von 'rootsf' für P_5 mit 0.09 Sekunden. Dies läßt sich damit begründen,

daß bei der schlecht konditionierten, vierfachen Nullstelle das Muller-Verfahren die zweite Iterationsschleife mit neuen Startwerten durchläuft, um versuchsweise eine bessere Näherung der Nullstelle zu finden. Die gleiche Ursache hat die im Vergleich zu den anderen beiden Routinen doppelt so große Rechenzeit beim Polynom P_8 aus Tabelle 7. auffallend bei P_8 ist weiterhin der sehr gute reale Fehler des Programmes 'rootsj'. Da dies das einzig auffallend gute Ergebnis bei der Bestimmung von mehrfachen Nullstellen ist, ist vermutlich der Spezialfall, daß alle Nullstellen eines Polynomes gleich sind, gesondert behandelt worden (siehe auch [7]). Die Nullstellen des Polynomes $P_9(x)$ aus Tabelle 7 werden von 'rootsf' am besten bestimmt. Die Fehler aller Funktionen liegen in etwa im Bereich der Maschinengenauigkeit. Die Nullstellen des Polynomes $P_7(x)$ aus Tabelle 6 bestimmt die Funktion 'rootsj' mit dem geringsten Fehler im Bereich von $\approx 10^{-8}$. Die Funktion 'rootsf' liefert hier Ergebnisse, die um zwei Zehnerpotenzen schlechter sind als jene von 'rootsj', die Resultate von 'roots' sind sogar um drei Zehnerpotenzen ungenauer als die von der Funktion 'rootsj'. Die restlichen Zahlenwerte weisen keine weiteren nennenswerten Unterschiede zu den anderen Verfahren auf.
In allen Beispielen mit mehrfachen Nullstellen nimmt die erreichbare Genauigkeit bis auf die Ausnahme von 'rootsj' bei P_8 wie erwartet nach der Beziehung (26) ab.

Die bisherigen Ergebnisse lassen erkennen, daß die Genauigkeit der bestimmten Nullstellen für fast alle Polynome bei allen drei Funktionen in der gleichen Größenordnung liegen. Bei mehrfachen Nullstellen ist der Fehler von 'rootsj' etwas geringer als bei 'rootsf' und 'roots'. Die Rechenzeiten der drei Funktionen sind zwar leicht unterschiedlich, die Abweichungen sind jedoch nicht gravierend.

9.1.4 Stabilität der Deflation

Stabilität der Deflation soll hier bedeuten, daß durch die Abdivision einer oder zweier Nullstellen die mit dem dividierten Polynom berechneten Nullstellen nur unwesentlich ungenauer sind als diejenigen, die man mit dem ursprüglichen Polynom berechnet hätte. Da durch die Nachiteration mit dem ursprünglichen Polynom im Newton-Verfahren die mögliche Verschlechterung der Konditionen durch die Deflation beseitigt wird, sind von der Funktion 'rootsf' recht gute Ergebnisse zu erwarten. Die Resultate für die verwendeten Polynome

$$\begin{aligned} P_{10}(x) &= (x-A)(x-1)(x-\frac{1}{A}) \\ P_{11}(x) &= \prod_{\nu=1-M}^{M-1}(x-e^{\frac{j\nu\pi}{2M}}) \cdot \prod_{\nu=M}^{3M}(x-0.9\cdot e^{\frac{j\nu\pi}{2M}}) \end{aligned}$$

zeigen Tabelle 8 und 9.

	realer Fehler			gesch. Fehler	Zeit in s		
A	roots	rootsj	rootsf	rootsf	roots	rootsj	rootsf
$1 \cdot 10^3$	$2.274 \cdot 10^{-16}$	$2.220 \cdot 10^{-19}$	0	$1.139 \cdot 10^{-16}$	0.01	0.02	0.01
$1 \cdot 10^6$	$2.328 \cdot 10^{-16}$	$2.220 \cdot 10^{-22}$	$2.118 \cdot 10^{-28}$	$1.164 \cdot 10^{-16}$	0.01	0.01	0.00
$1 \cdot 10^9$	$4.441 \cdot 10^{-16}$	$2.068 \cdot 10^{-25}$	$2.068 \cdot 10^{-25}$	$1.192 \cdot 10^{-16}$	0.01	0.03	0.01

Tabelle 8: Polynom $P_{10}(x)$

	realer Fehler			gesch. Fehler	Zeit in s		
M	roots	rootsj	rootsf	rootsf	roots	rootsj	rootsf
5	$2.220 \cdot 10^{-15}$	$1.974 \cdot 10^{-15}$	$2.467 \cdot 10^{-16}$	$1.820 \cdot 10^{-16}$	0.16	0.03	0.06
10	$3.824 \cdot 10^{-15}$	$2.202 \cdot 10^{-12}$	$2.467 \cdot 10^{-16}$	$1.825 \cdot 10^{-16}$	0.93	0.05	0.14
12	$3.997 \cdot 10^{-15}$	$1.891 \cdot 10^{-4}$	$2.220 \cdot 10^{-16}$	$1.364 \cdot 10^{-16}$	1.45	0.07	0.19
15	$6.661 \cdot 10^{-15}$	$1.001 \cdot 10^{-1}$	$7.401 \cdot 10^{-16}$	$9.812 \cdot 10^{-17}$	2.62	0.11	0.25
20	$1.739 \cdot 10^{-14}$	$1.539 \cdot 10^{-1}$	$8.635 \cdot 10^{-16}$	$1.825 \cdot 10^{-16}$	6.11	0.18	0.37
25	$1.210 \cdot 10^{-13}$	$1.111 \cdot 10^{-1}$	$3.701 \cdot 10^{-15}$	$1.875 \cdot 10^{-16}$	11.81	0.30	0.49
50	$1.555 \cdot 10^{-9}$	$3.508 \cdot 10^{2}$	$2.445 \cdot 10^{-13}$	$1.121 \cdot 10^{-14}$	96.42	26.11	2.11

Tabelle 9: Polynom $P_{11}(x)$

Wie aus Tabelle 8 ersichtlich, liefern alle drei Programme gute Ergebnisse bezüglich Zeit und Fehler. Auffallend ist, daß der Fehler von 'rootsf' für dieses Polynom stark überschätzt wird. Die besten Ergebnisse sowohl bezüglich Zeit, als auch bezüglich Fehler zeigt das Programm 'rootsf'.
Für das Polynom $P_{11}(x)$ ist zum einen auffallend, daß das Programm 'rootsj' von Jenkins und Traub für höhere Grade vollkommen versagt, wobei sich der Grad N von $P_{11}(x)$ aus $N = 4 \cdot M$ ergibt. Dieser Sachverhalt ist von den Autoren selbst bereits in [4] vermerkt worden. Weiterhin ist der unverhältnismäßig hohe Zeitaufwand von 'roots' auffällig. Die deutlich besten Ergebnisse liefert hierbei wieder die Funktion 'rootsf'.

Beim Polynom $P_{11}(x)$ liegen die Nullstellen auf einem Halbkreis mit dem Radius 1 in der rechten Halbebene der komplexen Ebene und auf einem Halbkreis mit dem Radius 0.9 in der linken Halbebene. Diese Nullstellenverteilung ist ähnlich der eines nichtrekursiven Filters. In diesem Fall läge ein Hochpaß vor. Gerade hierbei zeigen sich die oben genannten Schwächen der Programme 'roots' und 'rootsj' und die Stärke des Programmes 'rootsf'.

9.2 Kreisteilungspolynome

Da nichtrekursive Systeme Nullstellenverteilungen entlang des Einheitskreises haben, sind im folgenden Kreisteilungspolynome betrachtet worden. Zur Untersuchung wurden die beiden rellen Polynome

$$\begin{aligned} P_1(x) &= x^N + 1 \\ P_2(x) &= x^N - 1 \end{aligned}$$

und das komplexwertige Polynom

$$P_3(x) = j(x^N - 1)$$

herangezogen. Es folgt zunächst ein Vergleich aller Programme bezüglich $P_1(x)$. Ebenfalls mit Hilfe des Polynoms $P_1(x)$ wird die Rechenzeit und der Speicherbedarf von 'roots' und 'rootsf' untersucht und im Anschluß das Programm 'rootsf' unter Verwendung der Polynome $P_2(x)$ und $P_3(x)$ betrachtet.

9.2.1 Vergleich von 'roots','rootsj' und 'rootsf'

Die Ergebnisse der drei Programme bezüglich des Polynomes $P_1(x)$ zeigt Tabelle 10. Betrachtet man zunächst die Rechenzeiten, so liefert die Funktion 'rootsf' die

	realer Fehler			gesch. Fehler	Zeit in s		
N	roots	rootsj	rootsf	rootsf	roots	rootsj	rootsf
10	$5.551 \cdot 10^{-16}$	$6.661 \cdot 10^{-16}$	$2.220 \cdot 10^{-16}$	$1.784 \cdot 10^{-16}$	0.03	0.03	0.02
30	$1.332 \cdot 10^{-15}$	$8.882 \cdot 10^{-15}$	$2.220 \cdot 10^{-16}$	$8.775 \cdot 10^{-17}$	0.11	0.06	0.05
50	$1.332 \cdot 10^{-15}$	$8.898 \cdot 10^{-6}$	$2.220 \cdot 10^{-16}$	$1.964 \cdot 10^{-16}$	0.33	0.14	0.11
60	$1.665 \cdot 10^{-15}$	$3.440 \cdot 10^{-5}$	$2.220 \cdot 10^{-16}$	$1.652 \cdot 10^{-16}$	0.48	0.22	0.13
70	$1.332 \cdot 10^{-15}$	$3.560 \cdot 10^{-1}$	$2.220 \cdot 10^{-16}$	$1.110 \cdot 10^{-16}$	0.72	0.25	0.16

Tabelle 10: Polynom $P_1(x) = x^N + 1$

besten Ergebnisse, die Funktion 'roots' die bei weitem schlechtesten. So berechnet beispielsweise 'rootsf' beim Grad $N = 70$ die gesuchten Nullstellen circa 4.5 mal schneller als 'roots'. Die Fehler sind in allen Fällen bei 'rootsf' am geringsten und in etwa vergleichbar mit denen von 'roots'. Auffallend ist hier vor allem die starke Vergrößerung des Fehlers von 'rootsj' bei steigendem Polynomgrad N. Aus diesem Grund wurde nur bis zum Grad $N = 70$ ausgewertet, da hierbei 'rootsj' bereits einen maximalen Fehler der berechneten Nullstellen von $\approx 30\%$ aufweist. Die zu den von 'rootsj' und 'rootsf' berechneten Nullstellen gehörigen PN-Diagramme zeigen die Abbildungen 9 und 10.

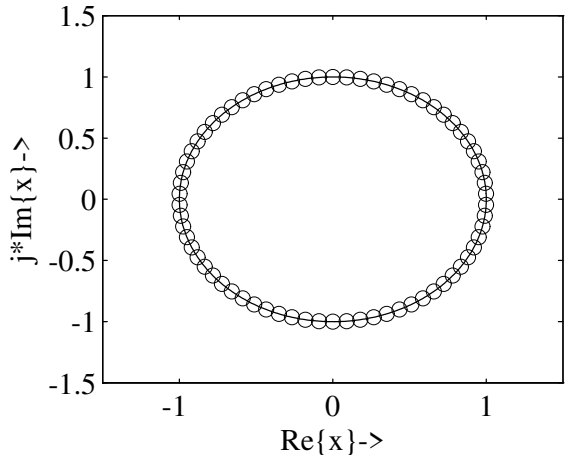

Abbildung 9: PN-Diagramm von 'rootsf' für das Polynom $P(x) = x^{70}+1$

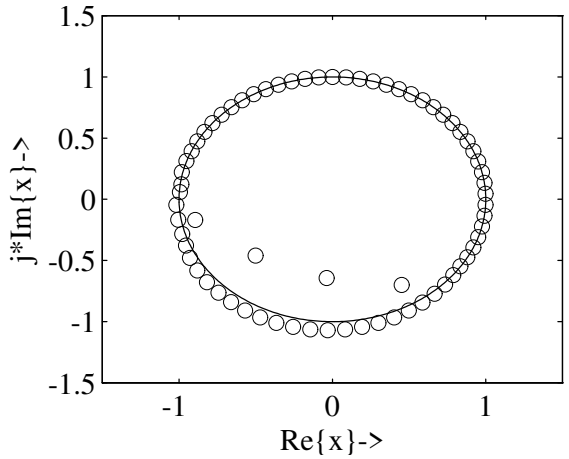

Abbildung 10: PN-Diagramm von 'rootsj' für das Polynom $P(x) = x^{70}+1$

9.2.2 Überlegungen zu Rechenzeit und Speicherbedarf

	Rechenzeit in s	
N	roots	rootsf
50	0.32	0.01
100	1.72	0.30
150	5.29	0.41
200	11.84	0.62
250	26.90	0.87
300	50.27	1.17
350	87.35	1.47
400	139.67	1.75

Tabelle 11: Polynom $P_1(x) = x^N + 1$

Wie aus dem vorherigen Abschnitt bereits zu erkennen, ist die Funktion 'rootsj' aufgrund des auftretenden, großen Fehlers bei der Berechnung von Nullstellen bei Polynomen höheren Grades ($N > 60 \ldots 70$) vollkommen ungeeignet. Es bleibt, die beiden Funktionen 'roots' und 'rootsf' miteinander zu vergleichen. Wie die Ergebnisse aus Tabelle 10 vermuten lassen, vergrößert sich die Rechenzeit von 'roots' bezüglich 'rootsf' mit zunehmendem Polynomgrad N. Die Resultate dieser Untersuchungen zeigt Tabelle 11, eine graphische Veranschaulichung liefert Abbildung 11. Die Nullstellen wurden von 'rootsf' und 'roots' bei den Ergebnissen aus Tabelle

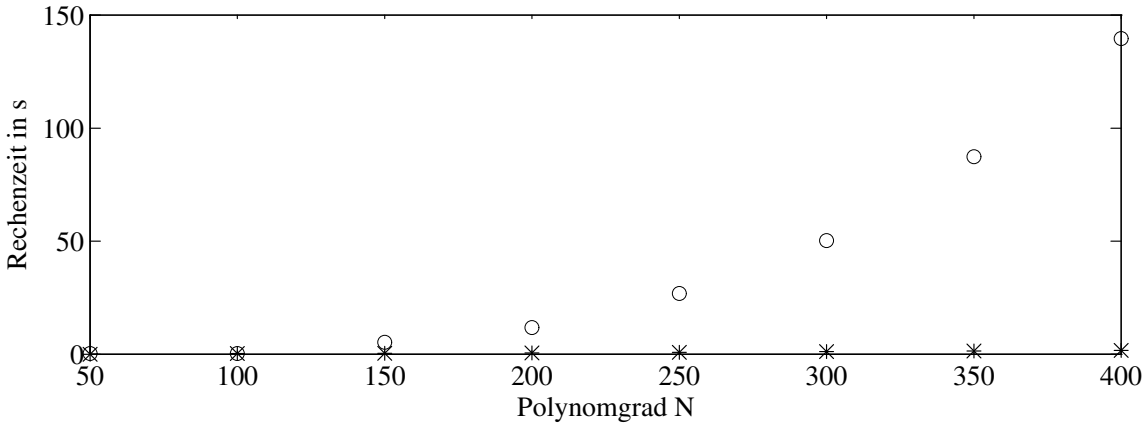

Abbildung 11: Rechenzeit von 'rootsf' ($*$) und 'roots' (o) für $P(x) = x^N + 1$

11 in etwa mit Maschinengenauigkeit berechnet, so daß auf eine explizite Angabe des Fehlers entsprechend der vorhergehenden Auswertungen verzichtet wurde.
Wie aus der Tabelle 11 ersichtlich, ist für den Grad $N = 400$ 'roots' bereits ca. 80 mal langsamer als die Funktion 'rootsf'. Um die Nullstellen von $P(x) = x^{400} + 1$ zu berechnen, bildet 'roots' eine Begleitmatrix der Dimension 401x401, wobei im double-Format jedes Element im reellen Fall einen Speicherbedarf von 8 Byte hat. Nur für die Begleitmatrix allein ergibt sich somit bereits ein Speicherbedarf von $\approx$ 1.2MByte. Im Vergleich hierzu benötigt sowohl im reellen, als auch im komplexen Fall die Funktion 'rootsf' für die Koeffizienten $2 \cdot 401 \cdot 8 \approx 6.3$kByte. Um sicherzugehen, daß Auslagerungen von Daten durch das Betriebssystem auf Festplatten auch auf einem PC die Zeitmessungen nicht beeinflussen, wurden die Messungen nur bis zum Grad $N = 400$ durchgeführt.

9.2.3 Untersuchungen zu 'rootsf'

Zur weiteren Untersuchung der Funktion 'rootsf' wurden die Polynome $P_2(x)$ und $P_3(x)$ verwendet. Die zugehörigen Resultate zeigen Tabelle 12 und Tabelle 13.

N	realer Fehler	gesch. Fehler	Zeit in s
100	$1.024 \cdot 10^{-15}$	$1.868 \cdot 10^{-16}$	0.26
300	$1.024 \cdot 10^{-15}$	$8.828 \cdot 10^{-17}$	1.20
500	$8.968 \cdot 10^{-16}$	$2.069 \cdot 10^{-16}$	2.61
700	$1.144 \cdot 10^{-15}$	$2.105 \cdot 10^{-16}$	4.55
1000	$1.024 \cdot 10^{-15}$	$2.202 \cdot 10^{-16}$	8.62
1300	$9.930 \cdot 10^{-16}$	$1.891 \cdot 10^{-16}$	13.76
1500	$1.201 \cdot 10^{-15}$	$1.983 \cdot 10^{-16}$	18.02
1700	$1.159 \cdot 10^{-15}$	$2.220 \cdot 10^{-16}$	22.27
2000	$1.106 \cdot 10^{-15}$	$2.059 \cdot 10^{-16}$	30.61
3000	$1.392 \cdot 10^{-15}$	$2.184 \cdot 10^{-16}$	67.07
4000	$1.106 \cdot 10^{-15}$	$2.046 \cdot 10^{-16}$	118.25
5000	$1.024 \cdot 10^{-15}$	$2.220 \cdot 10^{-16}$	189.66
10000	$1.047 \cdot 10^{-15}$	$2.196 \cdot 10^{-16}$	914.18

Tabelle 12: Polynom $P_2(x) = x^N - 1$

Wie zu erkennen liegen die realen Fehler aller Testpolynome im Bereich der Maschinengenauigkeit. Dies ergibt sich aus der sehr guten Konditionierung der Nullstellen von Kreisteilungspolynomen. Ebenfalls auffällig ist die wie bei allen Kreisteilungspolynomen gute Übereinstimmung zwischen realem und geschätztem Fehler. Die Rechenzeit zur Berechnung des Polynomes $P_3(x)$ ist im Vergleich zu der von $P_2(x)$ ungefähr um den Faktor 2 langsamer. Dies ist damit zu erklären, daß für das Polynom $P_3(x)$ mit seinen komplexen Koeffizienten alle Nullstellen einzeln berechnet werden müssen, also ungefähr das Doppelte an Rechenaufwand vorliegt.
Eine Darstellung der Rechenzeit in Abhängigkeit vom Polynomgrad liefern für beide Polynome die Abbildungen 12 und 13.
Die in den Darstellungen eingezeichneten Funktionen der Rechenzeit t in Abhängigkeit vom Polynomgrad N sind in beiden Fällen Polynome 3. Grades, wobei sich für $N > 100$ und $P_2(x)$

$$t(N) \approx 3.945 \cdot 10^{-10} N^3 + 4.760 \cdot 10^{-6} N^2 + 4.438 \cdot 10^{-3} N - 6.448 \cdot 10^{-1}$$

N	realer Fehler	gesch. Fehler	Zeit in s
100	$1.024 \cdot 10^{-15}$	$2.199 \cdot 10^{-16}$	0.49
200	$1.180 \cdot 10^{-15}$	$2.217 \cdot 10^{-16}$	1.20
500	$8.618 \cdot 10^{-16}$	$2.159 \cdot 10^{-16}$	5.66
1000	$1.024 \cdot 10^{-15}$	$2.187 \cdot 10^{-16}$	17.51
2000	$1.106 \cdot 10^{-15}$	$2.208 \cdot 10^{-16}$	63.28
5000	$1.024 \cdot 10^{-15}$	$2.216 \cdot 10^{-16}$	370.44
10000	$1.024 \cdot 10^{-15}$	$2.212 \cdot 10^{-16}$	1776.97

Tabelle 13: Polynom $P_3(x) = j * (x^N - 1)$

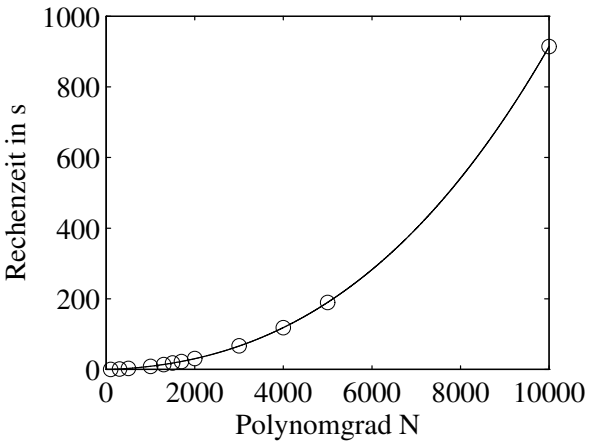

Abbildung 12: Rechenzeit in Sekunden für das Polynom $P_2(x) = x^N - 1$

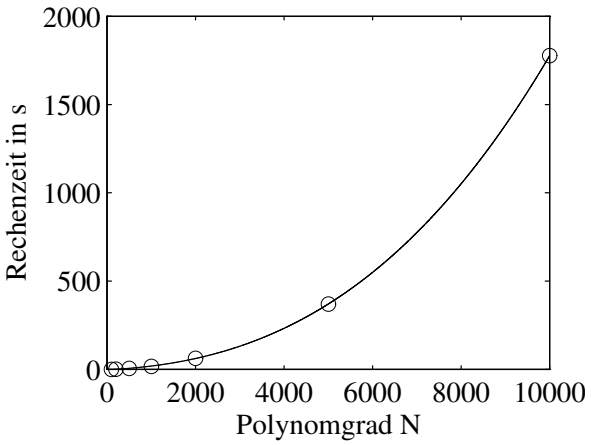

Abbildung 13: Rechenzeit in Sekunden für das Polynom $P_3(x) = j(x^N - 1)$

und für P_3

$$t(N) \approx 8.062 \cdot 10^{-10} N^3 + 8.587 \cdot 10^{-6} N^2 + 1.137 \cdot 10^{-2} N - 1.628$$

ergibt. Die angegebenen Beziehungen sind nur als Näherungen zu verstehen. Die dargestellten Polynome wurden durch einen Gauß-Ausgleich gewonnen.

9.3 Beispielhaft ausgewählte FIR-Filter

Im Laufe der Studienarbeit wurden für eine Reihe von TP-FIR-Filtern die Nullstellenverteilung berechnet. Die Bilder 14 und 15 stellen hierbei die PN-Diagramme eines TP-Filters 150. Grades dar, die Abbildung 16 ein TP-Filter 800. Grades.

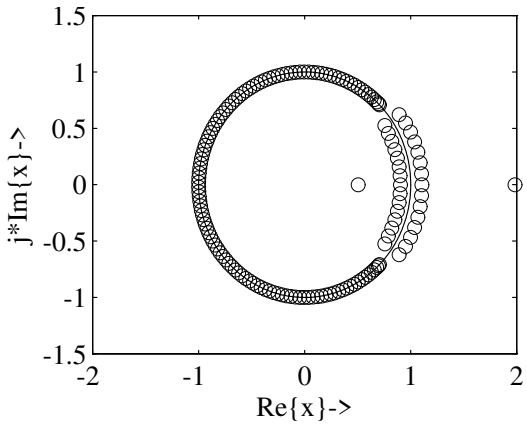

Abbildung 14: PN-Diagramm für TP-Filter 150. Grades mit 'rootsf' berechnet

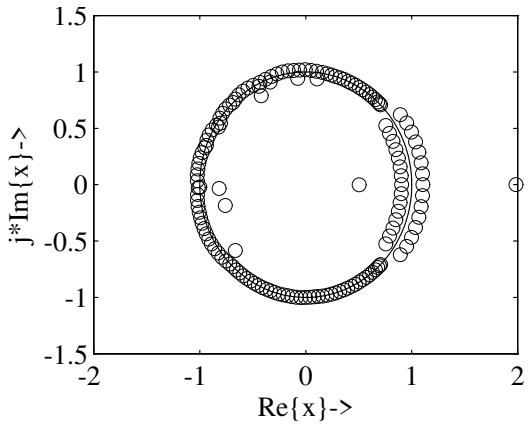

Abbildung 15: PN-Diagramm für TP-Filter 150. Grades mit 'rootsj' berechnet

Die Nullstellen in den Bildern 14 und 16 sind hierbei mit 'rootsf', die Nullstellen der Abbildung 15 mit 'rootsj' erstellt worden. Der Versuch, auch das TP-Filter 800. Grades mit Hilfe von 'rootsj' zu berechnen, führte zum Absturz des Programmes. Aufgrund der großen Rechenzeit ist an dieser Stelle die Funktion 'roots' nicht mehr näher untersucht worden.
Die Rechenzeiten von 'rootsf' lagen bei 0.74 Sekunden (150. Grad) und 10.52 Sekunden (800. Grad) und der maximale relative Fehler der bestimmten Nullstellen im Bereich $\approx 10^{-14}$ (150. Grad) und $\approx 10^{-12}$ (800. Grad).
Die Rechenzeit von 'rootsj' lagen für das TP-Filter 150. Grades bei 1.12 Sekunden, der maximale Fehler betrug $\approx 22\%$.

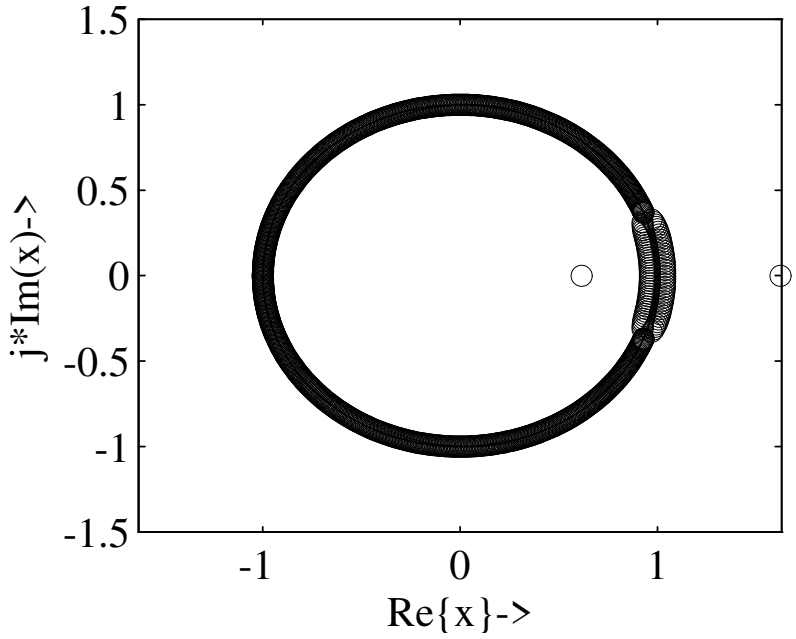

Abbildung 16: PN-Diagramm für TP-Filter 800. Grades mit 'rootsf' berechnet

10 Zusammenfassung und Bewertung von 'rootsf'

Aus den Ergebnissen der letzten Abschnitte kann man ablesen, daß die Schwächen des Programmes 'rootsf' in der Berechnung von mehrfachen beziehungsweise nahe zusammenliegenden Nullstellen liegen. Allerdings liefern die beiden Vergleichsprogramme 'roots' und 'rootsj' bis auf eine Ausnahme ähnlich schlechte Ergebnisse. Der Nachteil des in diesen Fällen höheren Zeitaufwandes der Funktion 'rootsf' läßt sich aus der Notwendigkeit der großen maximal zulässigen Iterationszahl im Muller-Verfahren zur Berechnung von Nullstellen hochgradiger Polynome ableiten.
Der Hauptvorteil des Programmes liegt in der Bestimmung von Polynomnullstellen hohen Grades. Hierbei versagen aus Zeit beziehungsweise Rechengenauigkeitsgründen die am LNT vorliegenden Nullstellenprogramme 'roots' und 'rootsj' gänzlich. Gerade bei der Bestimmung nichtrekursiver Filter können aber mitunter hohe Polynomgrade auftreten.

Zusammenfassend läßt sich somit sagen:

Liegen die Nullstellen eines Polynomes alle auf einer Stelle oder liegen mehrfache Nullstellen vor, so sollte für Polynome bis zum Grad $N \approx 60$ die Funktion 'rootsj' verwendet werden.
Ansonsten hat bis zum Grad $N \approx 60$ die Funktion 'rootsf' gegenüber 'rootsj' leichte Genauigkeitsvorteile. Die Geschwindigkeitsvorteile von 'rootsj' sind hierbei verschwindend gering. Für höhere Grade scheidet 'rootsj' aufgrund der großen Fehler bei der Berechnung der Nullstellen aus.
Gegenüber 'roots' besitzt 'rootsf' für sämtliche Polynome leichte bis mittlere Genauigkeitsvorteile und mit größer werdendem N immer größere Geschwindigkeitsvorteile.
Mit 'rootsf' ist es nun auch möglich, Nullstellen von Polynomen hohen Grades in vergleichsweise kurzer Rechenzeit mit geringem Fehler zu berechnen. Somit stellt die Funktion 'rootsf' eine deutliche Verbesserung gegenüber bekannten, guten Verfahren zur Bestimmung von Polynomnullstellen dar.

11 Ausblick

Wie im Kapitel über linearphasige Filter bereits angeschnitten ist die Berechnung der Nullstellen bezüglich der Rechenzeit nur suboptimal, da aus der Kenntnis einer nicht auf dem Einheitskreis liegenden, komplexen Nullstelle im besten Fall ein Quadrupel von Nullstellen abgespalten werden kann. Die Lösungsansätze zu einer weiteren Verbesserung des Programmes bezüglich des im Kapitel über linearphasige Filter angesprochenen Problems könnten hierfür nun sein:

- Suche der Nullstellen nur innerhalb und auf dem Einheitskreis, wobei dann auch nur diese, gegebenenfalls unter Ausnutzung der Reellwertigkeit des Systems, abgespalten werden $\Rightarrow$ alle restlichen Nullstellen außerhalb des Einheitskreises sind dann automatisch bestimmt.
- 'Merken' der am Einheitskreis gespiegelten Nullstelle und Abspalten der Nullstelle erst im nächsten Iterationsschritt, wobei dann möglicherweise die Koeffizienten des abgespaltenen Polynomes nicht so fehlerbehaftet sind.

Der zweite Lösungsansatz ergibt sich hierbei aus der Vermutung, daß die Nullstellenpaare, die die gleiche Phase und unterschiedlichen Betrag besitzen, bei der Polynomdeflation numerische Schwierigkeiten hervorrufen.

Literatur

[1] Bronstein I. N. et al.:
Taschenbuch der Mathematik, Verlag Harri Deutsch, Thun 1987

[2] Burrus C. S.:
An Algorithm for Factoring a Polynomial of a Complex Variable, Electrical and Computer Engineering Dept. and Computational Mathematics Laboratory, Rice University, Houston, April 1992

[3] Engeln-Müllges et al.:
Numerische Mathematik für Ingenieure, B.I.-Wissenschaftsverlag, Zürich 1987

[4] M.A. Jenkins und J.F. Traub:
Principles for Testing Polynomial Zerofinding Programs, ACM Transactions on Mathematical Software, Vol. 1, No. 1, March 1975, 26-34

[5] M.A. Jenkins und J.F. Traub:
Zeros of a Complex Polynomial, Communications of the ACM, Vol. 15, Feb. 1972, 97-99

[6] Kernighan B. W. et al.:
The C programming language, Prentice Hall, New Jersey 1988

[7] Lang M.:
FORTRAN-Nullstellenberechnungsprogramm unter Verwendung der Muller Methode, Lehrstuhl für Nachrichtentechnik Erlangen 1991

[8] The MathWorks, Inc.:
Matlab for 80386-based MS-DOS Personal Computers, User‘s Guide, South Natick 1989

[9] Press W. H. et al.:
Numerical Recipies in C, Cambridge University Press, Cambridge 1990

[10] Schüßler, H. W..:
Netzwerke, Signale und Systeme, Band 2, Springer-Verlag, Berlin 1990

[11] Schwetlick und Kretzschmar:
Numerische Verfahren für Naturwissenschaftler und Ingenieure, Fachbuchverlag Leipzig, Leipzig 1991

[12] Törnig und Spellucci:
Numerische Mathematik für Ingenieure und Physiker, Springer-Verlag, Berlin 1988

[13] Wilkinson J. H.
Rundungsfehler, Springer-Verlag, Berlin 1969