

Belegarbeit Computational Physics I

Silvio Fuchs ^{*†}

12. März 2009

^{*}Betreuer: Frank Setzpfandt

[†]Bei Fragen zum Programm oder Verbesserungsvorschlägen: Silvio.Fuchs@uni-jena.de

Inhaltsverzeichnis

| | | |
|----------|--|-----------|
| 1 | Aufgaben | 4 |
| 2 | Vorbetrachtung | 5 |
| 2.1 | Grundkonzept | 5 |
| 2.2 | Werdegang des Programms | 5 |
| 3 | mathematische Grundlagen und deren Implementierung | 6 |
| 3.1 | Erzeugung eines verrauschten Signals | 6 |
| 3.1.1 | Implementierung der Verrauschung | 6 |
| 3.2 | numerisches Ableiten | 6 |
| 3.2.1 | Implementierung des numerischen Ableitungsverfahrens | 7 |
| 3.3 | Polynomfilter | 7 |
| 3.3.1 | Implementierung des Savitzky-Golay-Filters | 7 |
| 3.4 | Fourierfilter | 8 |
| 3.4.1 | Implementierung des Fourierfilters | 8 |
| 4 | Das Programm | 9 |
| 4.1 | Struktur | 9 |
| 4.1.1 | Aufprägen einer Verrauschung ("rausch.m") | 9 |
| 4.1.2 | einfaches numerisches Ableiten ("numabl.m") | 9 |
| 4.1.3 | Fourierfilter ("fourierfilter.m") | 9 |
| 4.1.4 | Polynomfilter ("polynomfilter.m") | 10 |
| 4.2 | Navigation | 10 |
| 4.3 | grafische Ausgabe | 10 |
| 5 | Behebung kleinerer Probleme/Debugging | 11 |
| 5.1 | grafische Probleme | 11 |
| 5.1.1 | Auflösung | 11 |
| 5.1.2 | Balken im Fourierfilter | 11 |
| 5.2 | numerische Probleme | 12 |
| 5.2.1 | Polynomfilter | 12 |
| 5.2.2 | analytische Ableitungen | 12 |
| 6 | Auswertung | 14 |
| 6.1 | einfaches numerisches Ableiten | 14 |
| 6.2 | Fourierfilter | 14 |
| 6.3 | Polynomfilter | 14 |

Inhaltsverzeichnis

| | | |
|----------|------------------------|-----------|
| 7 | Nachwort | 15 |
| 7.1 | Zeitaufwandt | 15 |

1 Aufgaben

Numerisches Ableiten

Alle gemessenen Signale sind durch das bei jeder Messung entstehende Rauschen mehr oder weniger verzerrt. Die Berechnung der Ableitung eines solchen Signals ist nicht trivial. Wenn die funktionelle Abhängigkeit der gemessenen Größe von der unabhängigen Variable, z.B. der Zeit, bekannt ist, kann eine Funktion angefittet und diese dann analytisch abgeleitet werden. Ist dies nicht der Fall, müssen die benötigten Ableitungen auf numerischem Wege berechnet werden.

Schreiben Sie ein Programm, das die erste und zweite Ableitung eines verrauschten Messsignals auf drei verschiedenen Wegen numerisch berechnet. Die zu nutzenden Methoden sind dabei:

- a) die direkte numerische Ableitung ohne vorherige Nutzung eines Filters
- b) die Filterung des Signals im Fourierraum und anschließende Ableitung mittels Methode a)
- c) die stückweise Anfittung eines Polynoms (Savitzky-Golay-Filter) an das gemessene Signal und Bestimmung der Ableitung durch Differenzierung des Polynoms an einem Punkt

Testen Sie Ihr Programm durch Ableitung eines Sinus-, Gauss- und Exponentialsignals, die jeweils durch ein zufälliges Rauschen verzerrt werden. Vergleichen Sie Ihre Ergebnisse auf geeignete Weise mit der analytischen Ableitung der nichtverrauschten Funktion und finden Sie Kriterien für den sinnvollen Einsatz der Ableitungsmethoden.

Numerical Recipes in C: <http://www.nrbook.com/a/bookcpdf.php>

2 Vorbetrachtung

2.1 Grundkonzept

Zunächst stellte sich mir die Frage, wie ich ein Programm zur Ableitung verrauschter Signale konzipieren müsse, um sämtliche Teilaufgaben zu erfüllen. Ich entschied mich schnell für eine GUI-Oberfläche, die in MATLAB relativ einfach zu implementieren ist. Diese grafische Oberfläche verpackt die große Informationsdichte eines solchen Programmes übersichtlich und kompakt, so dass dem Benutzer ein einfaches Arbeiten möglich ist. Ein reines Konsolenprogramm zu schreiben, halte ich für sinnlos, weil es erstens bei sämtlichen Eingaben die der Benutzer tätigen muss, zu Syntaxfehlern kommen kann (durch Fehleingaben) und zweitens, um ein wirklich anpassungsfähiges, vielseitiges Programm zu schreiben, zu viele Eingaben nötig sind.

2.2 Werdegang des Programms

Zunächst versuchte ich eine einfache Konsolenanwendung zu schreiben, welche lediglich ein Signal verrauschen und danach ableiten konnte. Ich bemerkte schnell, dass, wenn das Programm möglichst vielseitig werden sollte, der Benutzer sehr viele Eingaben tätigen muss. Also arbeitete ich mich in das GUI-Management von MATLAB ein, um eine grafische Oberfläche die intuitiv bedient werden kann und bei der es nicht zu Syntaxfehlern durch falsche Eingabe kommen kann zu programmieren. Zunächst schrieb ich ein GUI zum Verrauschen einer beliebigen Funktion, gefolgt vom numerischen Ableiten, dem Fourierfilter und schließlich dem Polynomfilter. Danach verknüpfte ich die Programme miteinander und programmierte die Navigation zwischen den GUIs, um dem User zu ermöglichen zwischen den Filtern zu springen. Bis zur finalen Programmversion behob ich noch einige Fehler, formatierte den Quellcode übersichtlich und kommentierte diesen aus.

3 mathematische Grundlagen und deren Implementierung

3.1 Erzeugung eines verrauschten Signals

Die, in diesem Fall, sinnvollste Methode ein verrauschtes Signal zu erzeugen ist ein Addieren zufälliger Zahlen zu den Funktionswerten einer unverrauschten Diskretisierung einer Funktion. Zufällig meint in diesem Fall einen Zufallsgenerator, welcher auf der Normalverteilung aufbaut. Wahrscheinlichkeit, dass x im Intervall $[a,b]$ liegt:

$$p(a \leq x \leq b) = \frac{1}{\sqrt{2\pi}\sigma} \int_a^b e^{-\frac{(t-\mu)^2}{2\sigma^2}} dt$$

σ^2 :Varianz μ :Mittelwert

3.1.1 Impelmentierung der Verrauschung

1. `eingabef=str2num(['@(x) ',get(handles.funktion,'string')]); %einlesen der Funktion`
2. `ywerte=eingabef(xwerte);`
3. `rausch=max(ywerte)/4*get(handles.slider1, 'Value');`
4. `yrausch=eingabef(xwerte)+random('norm',0,rausch,1,length(xwerte));`

3.2 numerisches Ableiten

Die wichtigste Grundlage für die zu bearbeitende Aufgabe bildet das numerische Ableiten. Dieser Algorithmus berechnet, auf dem Differenzenquotient basierend, die Werte aller Ableitungen von diskreten Signalen bzw. Wertepaaren.

Für diskrete Abstände der x -Werte bzw. zwischen der zeitlichen Erfassung des Signals gilt für die erste Ableitung:

$$f'(x_{i+1}) = \frac{f(x_{i+2}) - f(x_i)}{2(x_2 - x_1)} + R((x_2 - x_1)^2)$$

Der Fehler bei diesem Algorithmus ist proportional zum Quadrat der Intervallbreite. Für die zweite Ableitung gilt:

$$f''(x_{i+1}) = \frac{f(x_{i+2}) + f(x_i) - 2f(x_{i+1}))}{(x_2 - x_1)^2} + R((x_2 - x_1)^2)$$

Auch hier ist der Fehlerterm proportional zum Quadrat der Intervallbreite.

3.2.1 Implementierung des numerischen Ableitungsverfahrens

```
1. xabst=xwerte(2)-xwerte(1);
2. abl=(ywerte(3:length(xwerte))-ywerte(1:length(xwerte)-2))/(2*xabst); %1 Abl.
3. ablabl=(ywerte(3:length(xwerte))+ywerte(1:length(xwerte)-2)
   -2*ywerte(2:length(xwerte)-1))/(xabst^2); %2.Abl.
```

Durch diese Methode können die Ableitungen am linken und rechten Intervallende nicht berechnet werden. Es ist klar, dass diese Werte nicht belegt werden, also die Ableitung 2 Diskretisierungspunkte weniger hat.

3.3 Polynomfilter

Der Savitzky-Golay Filter arbeitet indem er an jeden Diskretisierungspunkt ein Polynom frei wählbaren Grades anfitet. Mit "Anfitten" wird ein Vorgang bezeichnet in welchem eine Kurve so durch Punkte gelegt wird, dass die Summe der Abstandsquadrate der Punkte zu der Kurve minimal wird. Zur Berechnung der Koeffizienten des Polynoms in einem Diskretisierungspunkt werden alle Punkte beachtet die in einer vorgegebenen Umgebung um diesen liegen. So wird ein verrauschtes Signal durch die "Glätte" eines Polynoms niedrigen Grades abgerundet und geglättet. Bei Benutzung höherer Polynomordnungen werden zwar die einzelnen Punkte des Signals besser durch das angefitete Polynom repräsentiert, das heist der Fehler zu den ungefilterten Daten ist geringer, aber die Glättung des Signals lässt nach.

3.3.1 Implementierung des Savitzky-Golay-Filters

```
1. xabst=xwerte(2)-xwerte(1); %berechnet Sttzstellenabstand
2. anzahl=str2num(get(handles.edit_anz,'string')); %Anzahl der Umgebungspunkte fr Polynom
3. ord=str2num(get(handles.edit_poly,'string')); %Ordnung des Polynoms
%erzeugen der Matrix fr die Ableitung
4. for i=1:ord
5.     ablm(i)=ord+1-i;
6. end
%erzeugen der Matrix fr die 2te Ableitung
7. for i=1:ord-1
8.     ablabl(i)=ord-i;
9. end
10. f=yrausch; %Funktionswerte des verrauschten Signals
11. x=xwerte; %Diskretisierungspunkte
12. fgef=f(anzahl/2+1:length(f)-anzahl/2);%Kernstck dh. Teilmenge bei der
    immer mit voller Breite gefittet wird
13. for i=1:length(fgef)%schleife fr Kernelement
14.     ply=polyfit(x(i:anzahl+i),f(i:anzahl+i),ord);
15.     gef(i+anzahl/2)=polyval(ply,x(anzahl/2+i));
16.     nabl(i+anzahl/2)=polyval(ablm.*ply(1:length(ply)-1),x(anzahl/2+i));
%Multiplikation mit dem Polynomkoeffizienten dadurch ableiten
17.     nablabl(i+anzahl/2)=polyval((ablm(1:ord-1).*ablablm).*ply(1:length(ply)-2),x(anzahl/2+i));
%Multiplikation mit dem Polynomkoeffizienten dadurch ableiten
18. end
```

Das durch den Filter gewonnene abgerundete Signal wird durch die analytische Ableitung des Polynoms, welches an einen Punkt angefitet wurde, berechnet. Es ist klar, dass die Punkte die nicht innerhalb einer Punkteumgebung, welche ein Anfiten eines Polynoms möglich macht, liegen, von diesem Filter nicht bearbeitet werden. (siehe Behebung kleinerer Probleme/Debugging)

3.4 Fourierfilter

Ein Fourierfilter analysiert das Frequenzspektrum eines Signals. Durch Manipulation im Frequenzraum (z.B. Herausfiltern hoher Frequenzen) und Rücktransformation in den Zeitraum wird ein verrauschtes Signal geglättet. Der Algorithmus zum Analysieren des Frequenzspektrums eines Signals ist in diesem Fall die FFT-Implementierung in MATLAB. Die Transformation berechnet sich aus folgenden Summen:

$$X(k) = \sum_{j=1}^N f(x_j) e^{-2\pi i(j-1)(k-1)/N} \text{ Amplitude der Frequenz } k$$

$$f(x_j) = \frac{1}{N} \sum_{k=1}^N X(k) e^{2\pi i(j-1)(k-1)/N} \text{ Rcktransformation}$$

Die Werte der Transformation sind komplex und komplex-konjugiert-spiegelsymmetrisch.

3.4.1 Implementierung des Fourierfilters

```

1. spek=fft(ywerte);
   %Berechnung nur der benötigten Werte da FFT komplex konjugiert symetrisch ist!
2. spek=spek(1:length(evalin('base','ywerte'))/2-mod(length(evalin('base','ywerte')),2)/2+1);
3. spek=real(spek); %Cosinusspektrum
4. speks=imag(spek); %Sinusspektum
5. trans=spekrc+i*spekrs;
.
.
.
   %Berechnung der Rcktransformation fr Gerade und ungerade Anzahl an Diskretisierungspunkten
6. if mod(length(evalin('base','xwerte')),2)==0
7.     yrausch_filter=real(ifft([trans conj(fliplr(trans(2:length(trans)-1)))]));
8. else
9.     yrausch_filter=real(ifft([trans conj(fliplr(trans(2:length(trans))))]));
10.end

```

4 Das Programm

4.1 Struktur

4.1.1 Aufprägen einer Verrauschung ("rausch.m")

Grundsätzlich teilt sich mein Programm in 4 Teilprogramme bzw. 4 Eingabefenster auf. Beim Start des Programms erscheint das Unterprogramm "rausch". Hier kann der Benutzer eine beliebige Funktion eingeben oder sich vorgegebene Funktionen aussuchen. Mit Hilfe eines Sliders ist es dem User möglich verschiedene Stärken für die Verrauschung auszuwählen, wobei sich die maximale Stärke nach dem maximalen Funktionswert im Intervall richtet. "rausch" aktualisiert die Plots bei sämtlichen Eingaben des Benutzers um bestmögliche Übersichtlichkeit zu gewährleisten. Es ist ausserdem möglich das Intervall sowie die Anzahl der Diskretisierungspunkte vorzugeben.

4.1.2 einfaches numerisches Ableiten ("numabl.m")

In diesem Teilprogramm kann mittels Knopfdruck die vorher festgelegte verrauschte Funktion zweimal abgeleitet werden. Die Ergebnisse werden mittels 4 Graphen dargestellt. Zum einen werden die erste und zweite Ableitung jeweils in ein Koordinatensystem gezeichnet, wobei die Ableitung des unverrauschten Signals als Referenz ebenfalls eingezeichnet wird. Zum anderen wird das Quadrat des Abstandes der verrauschten Ableitungen zur unverrauschten Ableitung in jedem Punkt berechnet und gezeichnet, sowie die Standardabweichung eben dieser beiden Ableitungen angezeigt. Dem Benutzer wird grafisch, sowie mittels Zahlenwerten eine Referenz gegeben, wie gut die eventuell gefilterten Ableitungen der wahren Ableitung entsprechen.

4.1.3 Fourierfilter ("fourierfilter.m")

In diesem, in meiner Implementierung aufwändigsten Programm, wird dem Benutzer die Möglichkeit gegeben, jede einzelne Frequenz des Sinus- und Kosinusspektrums des verrauschten Signals zu manipulieren. Ich halte dies für die beste Methode der manuellen Filterung, da ein einfacher Hoch- oder Tiefpass bei weitem nicht ein so großes "Filterpotential" hat, wie eine manuelle Anpassung der Frequenz. Bei einfachen Pässen ist es denkbar das wichtige Frequenzen die nicht von der Verrauschung stammen, sondern vom originalen unverrauschten Signal herausgefiltert werden und somit das Signal völlig verfälscht wird.

Meine Implementierung arbeitet mittels 3 Slidern mit denen die Breite des zu ändernden Frequenzbereiches, die Frequenz an sich und die Amplitude des jeweiligen Bereiches angepasst werden können. Ausserdem kann mit Hilfe eines Knopfdrucks vom Sinus- zum

Kosinusspektrum gewechselt werden. Auf der grafischen Oberfläche wird dann dynamisch bei jeder Veränderung der Amplitude das rücktransformierte Signal angezeigt, um dem Benutzer zu ermöglichen das Signal direkt nach seinen Wünschen anzupassen.

4.1.4 Polynomfilter ("polynomfilter.m")

In diesem Teilprogramm soll dem Benutzer ermöglicht werden, das vorher erzeugte ver-rauschte Signal mit Hilfe eines Polynomfilters zu glätten. Die Filterparameter müssen vom Benutzer eingegeben werden. Dieser muss die Anzahl der Umgebungspunkte (alle Punkte die zum Fit herangezogen werden, also nur gradzahlige Werte), sowie den Grad des Polynoms mit Hilfe zweier Eingabefenster festlegen. Durch den Ableitungsknopf, wird dann zum einen das gefilterte Signal angezeigt und zum anderen die selben Ausgabefenster die schon beim einfachen numerischen Ableiten verwendet wurden. Allerdings wird die numerische Ableitung des Signals durch die analytische Ableitung des angefiteten Polynoms gewonnen.

4.2 Navigation

Um dem Benutzer zu ermöglichen zwischen den verschiedenen Unterprogrammen zu wechseln, also beispielsweise verschiedene Filter nacheinander zu benutzen, wurden von mir "Buttons" in jedes Teilprogramm eingebracht, die die jeweils anderen Programme starten und das aktuelle beenden. Beim Fourierfilter hielt ich auch einen Resetbutton für sinnvoll, da der Benutzer eventuell gemachte Fehler beheben kann. Eine Schwierigkeit bestand darin die benötigten Variablen zwischen den GUIs zu übergeben. Ich entschied mich dazu die Variablen in den MATLAB-Workspace zu schreiben und wieder zu lesen. Dies löst das Problem und der Benutzer hat ausserdem die Möglichkeit mit den gefilterten Werten von eingegebenen Funktionen weiterführend zu arbeiten. Diese übergebenen Variablen sind selbsterklärend. Das Programm sollte grundsätzlich mit "rausch" aufgerufen werden, damit ersteinmal eine Funktion definiert werden kann. Die Fehlermeldung die anfänglich erscheint sollte ignoriert werden. Wenn der Benutzer ein Teilprogramm öffnet, ohne das die spezifischen Variablen im Workspace vorhanden sind, verweist das Unterprogramm automatisch auf "rausch".

4.3 grafische Ausgabe

Die grafische Ausgabe benötigte den Großteil der Arbeitszeit für das Programm. Ich werde dennoch auf diesen Teil meiner Arbeit nicht weiter eingehen, da im Quelltext die wichtigsten Dinge kommentiert sind. Eine ausführliche Beschreibung aller Aspekte der von mir verwendeten Befehle würde den Rahmen dieser Arbeit sprengen.

5 Behebung kleinerer Probleme/Debugging

5.1 grafische Probleme

5.1.1 Auflösung

Das Programm wurde auf einem Rechner mit einer Auflösung von 1280x1024 erstellt. Dies hat zur Folge, dass bei geringeren Auflösungen die GUIs nicht korrekt angezeigt werden, da das ursprüngliche GUI zu groß ist. Dieses Problem habe ich noch nicht gelöst, da ein auflösungsspezifisches Design mit MATLAB nicht realisierbar scheint. Dem Benutzer wird empfohlen mit einer Auflösung von 1280x1024 Pixel oder höher zu arbeiten.

5.1.2 Balken im Fourierfilter

Um den Balken der aktuell zu bearbeiteten Frequenz darzustellen, verwende ich 3 Slider. Ein großes und zeitaufwändiges Problem war es, die Sliderbereiche so anzugleichen, dass bei beliebiger Sliderstellung der Balken nur bis minimal zum linken und maximal zum rechten Intervallende reicht. Bei vorgegebener Breite muss also die Breite selber dynamisch an die einzustellende Frequenz angepasst werden. Meine Implementierung gliedert sich wie folgt:

```
1.frec=round(get(handles.slider_frec,'value'));
%rundet den Frequenzslider um krumme Zahlen zu vermeiden#
    %Hier wird das Rechteck \"uberpr\"uft diese Schleife passt den
    %Frecbereichsregler an die Stellung des Frequenzreglers an bei
    %Einerschritten sowie bei Spr\"ungen
    %Achtung etwas undurchsichtig da viele Bedingungen \"uberpr\"uft werden
    %viele Bed. n\"otig da der Sliderstep nicht unendlich sein darf usw.
    %wurde auf Funktionalit\"at getestet.
2.    if frec < length(spekr)/2 && frec~=1 && frec~=length(spekr)
    && frec+(frecb-1)/2 < length(spekr) && frec-(frecb-1)/2 > 1
3.        set(handles.slider_frecbr,'min',1,'max',frec*2-1,'SliderStep',
    [2/(frec*2-2) 11/(frec*2-2)]);
4.    elseif frec > length(spekr)/2 && frec~=1 && frec~=length(spekr)
    && frec+(frecb-1)/2 < length(spekr) && frec-(frecb-1)/2 > 1
5.        set(handles.slider_frecbr,'min',1,'max',(length(spekr)-frec)*2+1,
    'SliderStep',[2/((length(spekr)-frec)*2) 11/((length(spekr)-frec)*2)]);
6.    elseif frec==length(spekr)/2 && frec~=1 && frec~=length(spekr)
    && frec+(frecb-1)/2 < length(spekr) && frec-(frecb-1)/2 > 1
7.        set(handles.slider_frecbr,'min',1,'max',length(spekr),'SliderStep',.
    [2/(length(spekr)-1) 11/(length(spekr)-1)]);
8.    elseif frec==length(spekr) || frec==1
9.        frecb=1;
10.        set(handles.slider_frecbr,'value',frecb);
```

```

11.         set(handles.slider_frecbr,'min',1,'max',1.0000001,'SliderStep',[1 1]);
12.     elseif frec-(frecb-1)/2 < 1 || frec+(frecb-1)/2 > length(spekr)
    && frec~=length(spekr) && frec~=1
13.         if frec-(frecb-1)/2 < 1
14.             frecb=frecb+2*((frec-(frecb-1)/2)-1);
15.         elseif frec+(frecb-1)/2 > length(spekr)
16.             frecb=frecb-2*((frec+(frecb-1)/2)-length(spekr));
17.         end
18.         set(handles.slider_frecbr,'value',frecb);
19.         set(handles.slider_frecbr,'min',1,'max',frecb,'SliderStep',
    [2/(frecb-1) 11/(frecb-1)]);
20.     end

```

5.2 numerische Probleme

5.2.1 Polynomfilter

Dass der Polynomfilter mit der Umgebung eines zu filternden Punktes arbeitet, setzt der eigentliche Algorithmus erst nach der halben Umgebungsbreite um. Die vorhergehenden Werte werden nicht bearbeitet. Ich löste dieses Problem durch eine Filterung der Randpunkte mit verminderten Umgebungspunkten. Die Randpunkte werden also zunehmend ungenauer, weshalb auch die Ableitungen an den Rändern nicht mehr aussagekräftig sind. Der Benutzer sollte bei der Auswertung der Ergebnisse des Polynomfilter darauf achten besonders die Standardabweichung an den Rändern des Intervalls weniger stark zu wichten.

5.2.2 analytische Ableitungen

Um wie gefordert die numerische mit der analytischen Ableitung der drei Beispielfunktionen zu vergleichen, musste ich das Programm so gestalten, dass bei Eingabe einer der 3 Beispielfunktionen die Ableitung der unverrauschten Funktion nicht numerisch, sondern durch vorgegebene analytische Ableitungen errechnet wird. Ein CAS-System zu schreiben das alle Funktionen analytisch ableitet war nicht meine Aufgabe und würde den Rahmen dieses Projektes völlig sprengen.

```

1.xabst=xwerte(2)-xwerte(1); %berechnet St\ "utzstellenabstand
2.   if ywerte==sin(xwerte)
3.       abl=cos(xwerte);
4.       ablabl=-sin(xwerte);
5.       ablfall=1;
6.   elseif ywerte==exp(xwerte)
7.       abl=exp(xwerte);
8.       ablabl=abl;
9.       ablfall=1;
10.  elseif ywerte==exp((-1)*(xwerte.^2))
11.      abl=(-2)*xwerte.*exp((-1)*(xwerte.^2));
12.      ablabl=4*xwerte.^2.*exp((-1)*(xwerte.^2))-2*exp((-1)*xwerte.^2);
13.      ablfall=1;
14.  else
15.      abl=(ywerte(3:length(xwerte))-ywerte(1:length(xwerte)-2))/(2*xabst);
    %berechnet 1. Ableitung der Funktion

```

5 Behebung kleinerer Probleme/Debugging

```
16.         ablabl=(ywerte(3:length(xwerte))+ywerte(1:length(xwerte)-2)
-2*ywerte(2:length(xwerte)-1))/(xabst^2);% berechnet 2. Ableitung der Funktion
17.         ablfall=0;
% f\angt den fehler beim Plotten ab das unterschiedlich lange vektoren auftreten
18.     end
```

6 Auswertung

6.1 einfaches numerisches Ableiten

Beim einfachen numerischen Ableiten entstehen schon bei relativ geringer Verrauschung besonders in der zweiten Ableitung große Fehler. Das einfache numerische Ableiten eignet sich für eine aussagekräftige Ableitung nur bei hinreichend glatten Signalen. Bei verrauschten Signalen kann es praktisch deshalb nicht eingesetzt werden.

6.2 Fourierfilter

Der von mir implementierte Fourierfilter muss manuell eingestellt werden, das heist, dass letztendlich jede Frequenz angepasst werden kann. Bei genügend großem Aufwand liefert dieser Filter das beste Ergebnis. Also ein sehr gut geglättetes Signal. Um das Ausgangssignal vor der Verrauschung wieder zu erhalten, ist es nötig die Fouriertransformierte der Ausgangsfunktion zu kennen. Bei Unkenntnis kann mit Hilfe des Filters das Signal zwar glatt gemacht, aber auch völlig entfremdet werden. Eine automatisierte Filterung von verrauschten Signalen, bei denen das unverrauschte Signal nicht bekannt ist, was in der Praxis die Regel darstellt, ist also auf diesem Niveau nicht möglich. Denkbar sind einfache Filter wie Hoch-, Tief- und Bandpässe. Diese führen je nach Signalart nur zu mässigem Erfolg. Beispielsweise kann ein Signal welches unverrauscht schon sehr viele hohe Frequenzen enthält durch einen Hochpass eher entfremdet als geglättet werden.

6.3 Polynomfilter

Der Polynomfilter arbeitet mit nur einer Eingabe durch den Benutzer, was ihn natürlich für eine automatisierte Filterung favorisiert. Er liefert je nach Polynomordnung und Anzahl der Umgebungspunkte gute Ergebnisse. Die gefilterten Signale sind also hinreichend glatt und tragen bei nicht zu großer Verrauschung immer den Charakter des unverrauschten Signals. Eine Verfremdung wie beim Fourierfilter ist nicht denkbar. Ein weiterer Vorteil besteht darin, dass die Ableitungen des Signals nicht einfach numerisch, sondern durch die analytische Ableitung der Polynome geschieht. Dies spart Rechenaufwand und liefert genaue Ergebnisse. Die Genauigkeit des Polynomfilters nimmt an den Rändern ab!(siehe Kapitel "Behebung kleinerer Probleme/Debugging"). Je nach Polynomordnung kann ein Kompromiss zwischen genauer Approximation des verrauschten Signals (hohe Ordnung) und guter Glättung (niedrige Ordnung) eingegangen werden.

7 Nachwort

7.1 Zeitaufwandt

Das Programm zu schreiben erforderte sehr viel Zeit. Diese ging weit über das empfohlene Zeitmaß hinaus. Auch das Auffinden von Fehlern und deren Behebung verlangten eklatanten Zeitaufwandt. Durch die grafische Oberfläche und die Benutzerfreundlichkeit ist das fertige Programm recht komplex. Aus diesen Gründen halte ich diese Belegarbeit in einem Überschaubaren Rahmen und verwende weniger Zeit diese zu verfassen, als zum Erstellen des Programms nötig war. Ich bitte Sie dies in ihrer Bewertung zu berücksichtigen.