Fachgebiet Theorie der Signale

Studienarbeit St 88

Thema: **Untersuchung der Verstärkungssteuerung einer Telefon-Freisprecheinrichtung**

Beschreibung: Es ist ein Konzept zur Verstärkungssteuerung vorgeschlagen, um die relativ schlechten Pegelverhältnisse beim Freisprechen zu verbessern. Dazu ist Software in der Programmiersprache C zu er stellen, mit der die Verstärkungssteuerung simuliert werden kann und mit deren Hilfe sich allgemeine Untersuchungen an Sprach sequenzen vornehmen lassen. Aufgrund dieser Ergebnisse sind die Parameter der Verstärkungssteuerung optimal einzustellen.

Bearbeiter: Harald Zottmann

Betreuer: Dipl.-Ing. Peter Heitkämper

Beginn: 01.07.1992

Ende: 01.06.1993

Ich versichere, daß ich die vorliegende Studienarbeit nur unter Verwendung der angegebenen Hilfsmittel und ohne unzulässige fremde Hilfe angefertigt habe.

Darmstadt, den 01.06.1993

Inhaltsverzeichnis

		Seite
1	Einleitung	1
2	Konzept der Verstärkungssteuerung	3
3	Eigenschaften des SAM-Schätzers	6
3.1	Mathematische Analyse des SAM-Schätzers	9
3.1.1	Lösung für Gleichverteilung	16
3.1.2	Lösung für Laplaceverteilung	19
3.1.3	Lösung für Normalverteilung	32
3.2	Verifikation der Analyse für weiße Signale	25
3.2.1	Verifikation für Gleichverteilung	27
3.2.2	Verifikation für Laplaceverteilung	29
3.2.3	Verifikation für Normalverteilung	32
3.3	Zusammenfassung der Verifikation für weiße Signale	34
3.4	Verifikation der Analyse für farbige gaußverteilte Signale	36
3.5	Abweichung zwischen Normal- und Laplaceverteilung	49
4	Kennlinie der Verstärkungssteuerung	51
4.1	Auswirkung der SAM-Parameter auf den Klirrfaktor	55
4.2	Das Glättungfilter	58
5	Programmbeschreibung (SPV)	61
5.1	Start des Programms und Kommandozeilenoptionen	62
5.2	Das Hauptmenü	63
5.3	Verzeichnis wechseln	64
5.4	Parameter einstellen und SAM-Korrekturfaktor berechnen	64
5.5	Eingang erzeugen	66
5.6	Kennlinie der Verstärkungssteuerung zeigen	69
5.7	Histogramm zeigen	69
5.8	Zeitfunktion zeigen	72
5.9	Korrelation zeigen	72
5.10	Spektrum zeigen	72
5.11	File löschen	73
5.12	File umbenennen	73
5.13	File bearbeiten	74
5.14	Files verketten	74
5.15	Files kombinieren	76
5.16	Autokorrelation (AKF)	77
5.17	Kreuzkorrelation (KKF)	78

5.18 Fouriertransformation (FFT) 79
5.19 Schätzwertverlauf (SAM) 80
5.20 Verstärkungsverlauf (AMP) 80
5.21 Ausgangssignal (SIG) 80

6 Programmerweiterung (SPV) 81

7 Programmbedienung (SPVGEN) 83

8 Zusammenfassung und Ergebnis 86

Literaturverzeichnis 87

Anhang A Herleitung des Leistungskorrekturfaktors 88

Anhang B Zusammenhang zwischen Filterkoeffizient und Zeitkonstante 90

Anhang C Listing des Quellcodes von SPV 92

SPVINC.H 93
SPV.C 97
SPVFUNC.C 178
SPVEDIT.C 209
SPVSIRP.C 215
FILEREQ.C 222

Anhang D Listing des Quellcodes von SPVGEN 231

Anhang E Inhalt der Diskette 238

Anhang F Tabelle der Korrekturfaktoren 240

1 Einleitung

Im Gegensatz zum normalen Telefon, wo Mikrophon und Lautsprecher in einem Telefonhörer integriert sind, und damit eine feste Einheit bilden, treten beim Betrieb von Freisprechtelefonen mit freistehendem Mikrophon und Lautsprecher diverse Probleme auf.

Der feste Abstand von Mikrophon und Lautsprecher, sowie der gleichbleibende Besprechungsabstand, ermöglichen beim konventionellen Telefon die Wahl von festen Verstärkungs- bzw. Dämpfungsfaktoren für Mikrophon und Lautsprecher. Beim Freisprechtelefon kann man diesbezüglich keine allgemeingültigen Annahmen machen. Der Abstand des Sprechers zum Mikrophon und die damit verbundene Sprachsignal-Eingangsleistung kann in weiten Grenzen variieren, so daß eine variable Verstärkungssteuerung nötig wird, um eine möglichst gleichbleibende Sendeleistung, wie beim normalen Telefon, zu erhalten. Weiterhin gilt es störende Hintergrundgeräusche zu dämpfen.

Zur Unterdrückung des Rückkopplungspfeifens, das durch die akustische Kopplung von Lautsprecher und Mikrophon entsteht, wird ein Verfahren notwendig, welches erkennt wann der ferne Gesprächspartner spricht und eine Verstärkung dieser Signale verhindert. Gleichzeitig muß aber die Möglichkeit des Gegensprechens erhalten bleiben.

Im vorliegenden Bericht wird ein Konzept zur variablen Verstärkung von Sprachsignalen, wie es beim Telefonfreisprechen mit freistehendem Mikrophon und Lautsprecher Verwendung findet, untersucht.

In ***Kapitel 2*** wird zuerst das gesamte Konzept der Verstärkungssteuerung vorgestellt und mit einem kurzen Überblick beschrieben.

Kapitel 3 widmet sich der mathematischen Analyse des hierbei eingesetzten SAM-Schätzers, der den Kurzzeitmittelwert des Betrags der Eingangsfunktion schätzt. Diese Analyse führt auf eine Gleichung zur Berechnung des sich einstellenden SAM-Endwerts bei bekannter Dichtefunktion und Leistung des Eingangssignals. Die Gleichung wird am Beispiel der Gleich-, Laplace- und Normalverteilung gelöst und für weiße und farbige (realistische) Signale verifiziert.

Kapitel 4 befaßt sich mit der dem Konzept zugrundeliegenden Verstärkungssteuerung, die in Abhängigkeit vom Kurzzeitmittelwert einen Verstärkungsfaktor bestimmt. Hierbei werden Klirrfaktormessungen und das Verhalten der Verstärkungssteuerung bei einem Leistungssprung des Eingangssignals ausgewertet.

Zur Durchführung dieser Messungen wurde das Programm SPV in der Programmiersprache C entwickelt, dessen Beschreibung ***Kapitel 5*** beinhaltet. Für Programmierer, die SPV um zusätzliche Programmfunktionen erweitern wollen, wird in ***Kapitel 6*** beschrieben, welche grundsätzlichen Schritte hierfür notwendig sind. ***Kapitel 7*** enthält einige Beispiele zur Bedienung des Programms SPVGEN, das ebenfalls im Rahmen dieser Arbeit entstand. SPVGEN dient der Erzeugung von verschiedenen Musterfolgen, die mit SPV weiterbearbeitet werden können.

In ***Kapitel 8*** erfolgt die Zusammenfassung aller wichtigen Ergebnisse aus den vorangegangenen Kapiteln.

2 Konzept der Verstärkungssteuerung

Zur Verbesserung der schlechten Pegelverhältnisse beim Telefonfreisprechen ist eine Verstärkungssteuerung der Sendesignale vorgeschlagen [1]. Die Wirkungsweise dieses Algorithmus, der mit einem DSP realisiert wird, ist in Abbildung 2.1 veranschaulicht. Er soll dazu dienen, Hintergrundgeräusche zu unterdrücken und Sprachsignale variabel zu verstärken. Das Verfahren nutzt die Tatsache, daß zu sendende Sprachsignale eine größere Leistung haben als die zu dämpfenden Störgeräusche im Hintergrund. Somit ist es möglich, die Verstärkung abzusenken wenn niemand spricht, und bei einsetzender Sprache anzuheben.

Die Verstärkungssteuerung erfolgt also in Abhängigkeit von der Eingangsleistung über eine Kennlinie, die sich aus drei Bereichen zusammensetzt. Für kleine Eingangsleistungen wird angenommen, daß das anliegende Signal keine Sprache ist. Der Verstärkungsfaktor nimmt dann einen kleinen konstanten Faktor an. Dieser Bereich der Kennlinie (Abbildung 2.2) wird im folgenden Linearitätsbereich genannt. Beim Übergang zu leisen Sprachsignalen steigt die Eingangsleistung etwas an. In diesem Bereich soll der Verstärkungsverlauf überproportional nach einem Polynom $(p-1)$-ter Ordnung anwachsen. Dies entspricht in logarithmischer Darstellung der Ausgangsleistung über der Eingangsleistung, einer Geraden mit der Steigung p. Dieser Bereich wird Expansionsbereich genannt. Für Eingangsleistungen die einen Schwellwert Y_0 überschreiten, sollen die Verstärkungswerte mit steigender Eingangsleistung abnehmen, um eine Übersteuerung des Systems zu vermeiden und eine konstante Ausgangsleistung zu ermöglichen. Dieser Bereich der Kennlinie heißt Kompressionsbereich.

Die über die Kennlinie ermittelten Verstärkungswerte werden tiefpaßgefiltert, um schnelle Schwankungen der Verstärkungswerte und damit verbundene Verzerrungen zu vermeiden. Das Sendesignal wird nun mit den so ermittelten Verstärkungswerten multipliziert und zum fernen Hörer übertragen.

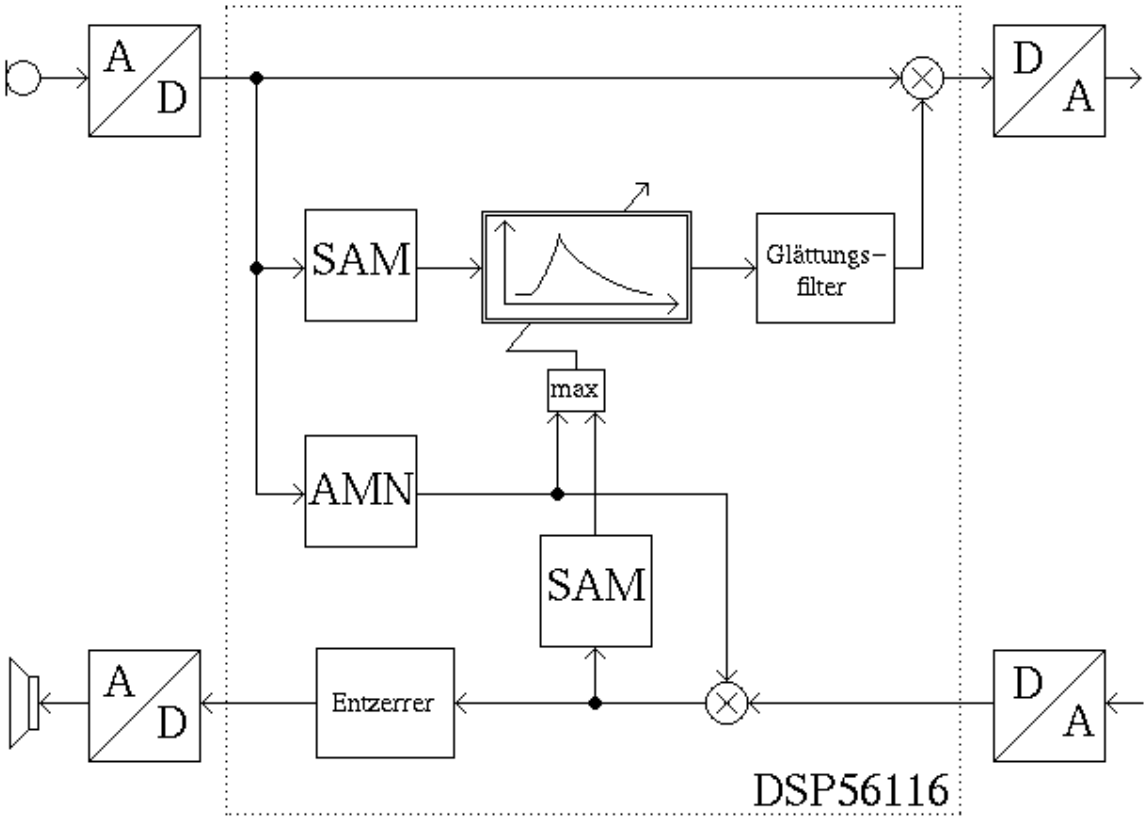

Abbildung 2.1: Blockschaltbild der Gesamtschaltung

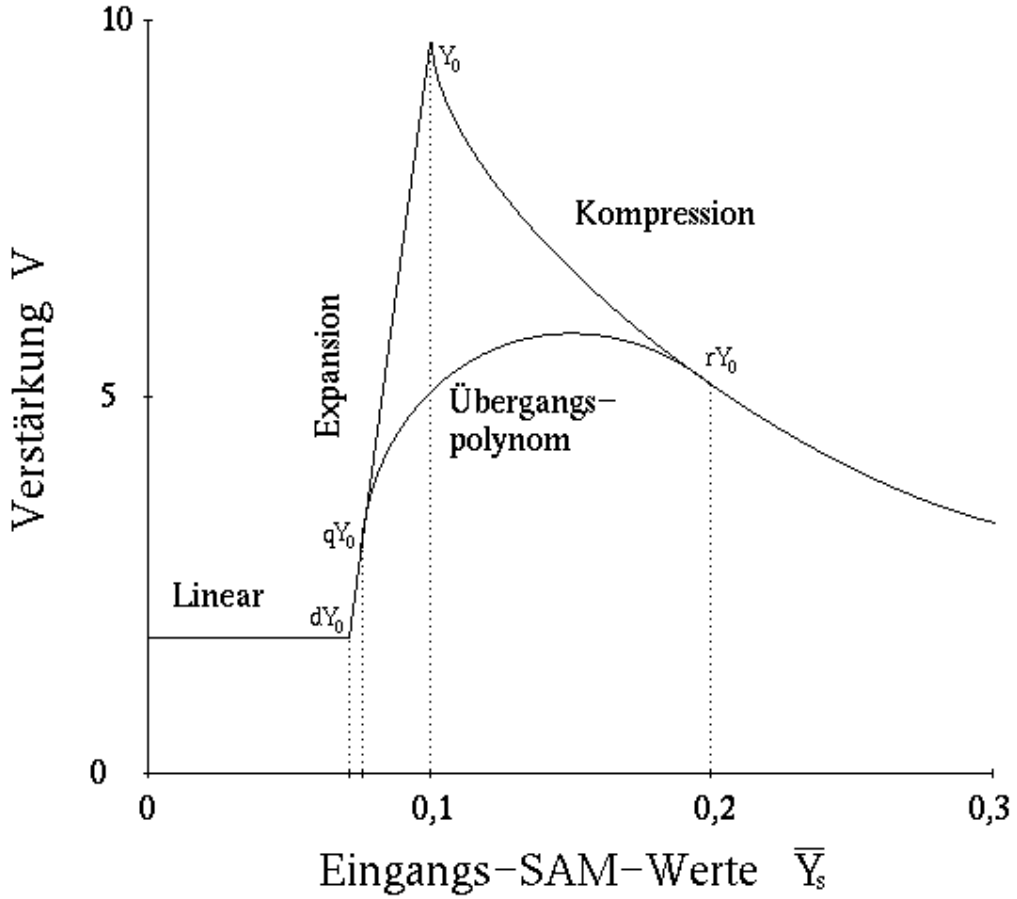

Abbildung 2.2: Kennlinie zur Berechnung der Verstärkungsfaktoren

3 Eigenschaften des SAM-Schätzers

Dem Gesamtkonzept liegt die Annahme zugrunde, daß zu sendende Sprachsignale eine höhere Eingangsleistung aufweisen als die Störgeräusche der Umgebung. Demnach kann man aufgrund der Größe der Beträge der Abtastwerte eines Intervalls eine Aussage darüber machen, ob das anliegende Signal zu einer Sprachpassage gehört, oder ob es sich um Hintergrundgeräusche handelt. Die Steuerung der Verstärkung über den Kennlinienverlauf, erfordert somit das Vorhandensein eines guten Schätzwerts der Leistung des Eingangssignals.

Zur Bildung eines solchen Schätzwerts kann man ein rekursives Filter erster Ordnung verwenden. Als Eingangssignal für dieses Filter kommen sowohl die Quadrate als auch die Beträge der Abtastwerte in Frage. In diesem Fall entscheidet man sich für die Beträge der Abtastwerte, da mit ihnen die spätere Rechnung vereinfacht wird. Um dem Leistungsanstieg bei einsetzender Sprache folgen zu können, darf die Zeitkonstante des Filters nicht zu hoch gewählt werden. Andererseits führen kurze Zeitkonstanten zu großen Verstärkungsschwankungen, und somit auch zu starken Verzerrungen des Signals. Die Anforderung nach einer kurzen Zeitkonstanten bei einsetzender Sprache auf der einen Seite, und einer langen Zeitkonstanten zur Verringerung der Verzerrungen auf der anderen Seite, läßt sich mit einem normalen AMN-Schätzer (engl. average magnitude), der nur eine Zeitkonstante hat, nicht erfüllen.

Ausgehend von diesen Bedingungen modifiziert man den AMN-Schätzer dahingehend, daß er je eine Zeitkonstante für ansteigende Eingangsleistung und eine andere für sinkende Eingangsleistung erhält. Der aus diesen Überlegungen hervorgehende SAM-Schätzer (engl. short-term average magnitude) bildet also einen Schätzwert über kurze Intervalle. Die Koeffizienten des Filters sind abhängig vom Verhältnis zwischen dem Betrag des Eingangssignals $Y(k)$ und dem letzten Schätzwert $Y_s(k-1)$. Die Variable k bezeichnet hierbei die Abtastzeitpunkte $t = kT$, wobei T das Abtastintervall ist.

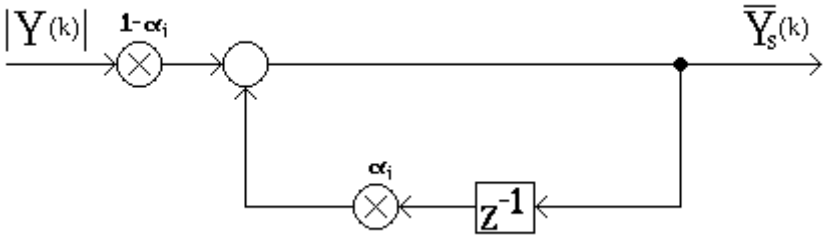

Abbildung 3.1: SAM-Schätzer

Die Gleichung zur Berechnung des Schätzwerts ist also gegeben durch [1]:

$$\overline{Y}_s(k) = (1-\alpha_i)|Y(k)| + \alpha_i \overline{Y}_s(k-1)$$

mit:

$$\alpha_i = \begin{cases} \alpha_f & |Y(k)| \le \overline{Y}_s(k-1) \\ \alpha_r & |Y(k)| > \overline{Y}_s(k-1) \end{cases}$$

Der Index *f* steht hierbei für *falling* und besagt damit, daß dieser Parameter bei sinkender Eingangsleistung aktiv ist. Der Index *r* steht dementsprechend für *rising* und kennzeichnet den Koeffizienten der bei steigender Eingangsleistung aktiv ist.

Damit der SAM-Schätzer nicht instabil wird, muß für die verwendeten Koeffizienten folgende Bedingung gelten:

$$0 \le \alpha_i \le 1$$

Dieses Verfahren erlaubt einer Erhöhung der Eingangsleistung mit einer kleinen Zeitkonstanten (< 10 ms) zu folgen, und zugleich kurze Leistungseinbrüche, wie sie bei Sprachsignalen üblich sind, mit einer langen Gesamtzeitkonstanten (> 60 ms) zu überbrücken.

Kapitel 3: Eigenschaften des SAM-Schätzers

Durch die unterschiedlichen Zeitkonstanten des SAM-Schätzers ist der sich einstellende mittlere Endwert, bei einem Eingangssignal konstanter Leistung, abhängig von dem Parameterpaar α_r und α_f. Dies kann man sich leicht veranschaulichen, wenn man zwei Extremfälle für die Parameterwahl betrachtet.

1. Fall $\alpha_r = 0$ und $\alpha_f = 1$

Für diese Parameter würde der SAM-Schätzer einer Erhöhung der Eingangsleistung ohne Zeitverzögerung folgen. Bei sinkender Eingangsleistung würde er jedoch wegen der unendlichen Zeitkonstanten seinen alten Wert beibehalten. Somit würde der SAM-Schätzer als Spitzenwertdetektor arbeiten.

2. Fall $\alpha_r = 1$ und $\alpha_f = 0$

In diesem Fall könnte der SAM-Schätzer dem Eingangssignal wegen der unendlichen Zeitkonstanten überhaupt nicht mehr folgen.

Um den SAM-Schätzwert mit dem Schätzwert eines anderen Schätzers, der nur eine Zeitkonstante hat, vergleichen zu können, muß der SAM-Schätzwert, wie in Abbildung 3.2 gezeigt, unabhängig von diesen Parametern gemacht werden. In der mathematischen Analyse des SAM-Schätzers wird daher zuerst eine Abhängigkeit zwischen dem sich einstellenden mittleren Endwert des SAM-Schätzers und den SAM-Parametern α_r und α_f gesucht.

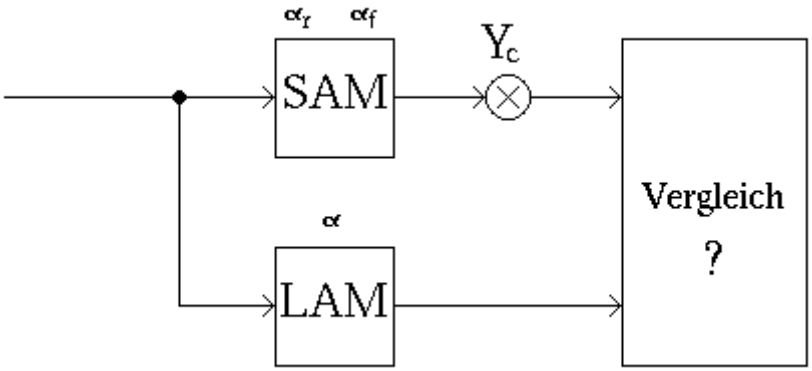

Abbildung 3.2: Korrekturfaktor

3.1 Mathematische Analyse des SAM-Schätzers

Zur Analyse werden die Signale durch Zufallsprozesse modelliert. Da α von Y abhängig ist, ist auch α ein Zufallsprozeß. Ausgehend von der schon gezeigten Formel für den Schätzwert wird der Erwartungswert gebildet:

$$E\{\overline{Y}_s(k)\} = E\{(1-\alpha_i)|Y(k)|\} + E\{\alpha_i \overline{Y}_s(k-1)\}$$

$$E\{(1-\alpha_i)|Y(k)|\} = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} (1-\alpha_i)|y|\, f_{\alpha y \overline{y}_s}(\alpha, y, \overline{y}_s)\, d\alpha\, dy\, d\overline{y}_s$$

$$E\{\alpha_i \overline{Y}_s(k-1)\} = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} \alpha_i \overline{y}_s\, f_{\alpha y \overline{y}_s}(\alpha, y, \overline{y}_s)\, d\alpha\, dy\, d\overline{y}_s$$

Für die beiden Summanden ergibt sich in ausführlicher Schreibweise:
Die gemeinsame Dichtefunktion läßt sich aufspalten in:

$$f_{\alpha y \overline{y}_s}(\alpha, y, \overline{y}_s) = f_\alpha(\alpha)\, f_{y \overline{y}_s}(y, \overline{y}_s \,|\, \alpha)$$

Dabei gilt:

$$f_\alpha(\alpha) = p_{\alpha_r}\delta(\alpha - \alpha_r) + p_{\alpha_f}\delta(\alpha - \alpha_f)$$

$$p_{\alpha_r}, p_{\alpha_f} : \text{Wahrscheinlichkeit mit der } \alpha_r \text{ bzw. } \alpha_f \text{ wirkt.}$$

und:

Kapitel 3.1: Mathematische Analyse des SAM-Schätzers

$$f_{y\bar{y}_s}(y,\bar{y}_s \mid \alpha): \text{Bedingte Verbunddichte von Y und } \bar{Y}_s$$

wenn man weiss, das α ausgewählt wurde.

$$f_{y\bar{y}_s}(y,\bar{y}_s \mid \alpha = \alpha_r) = \begin{cases} \frac{1}{p_{\alpha_r}} f_{y\bar{y}_s}(y,\bar{y}_s) & |Y| > \bar{Y}_s \\ 0 & \text{SONST} \end{cases}$$

$$f_{y\bar{y}_s}(y,\bar{y}_s \mid \alpha = \alpha_f) = \begin{cases} \frac{1}{p_{\alpha_f}} f_{y\bar{y}_s}(y,\bar{y}_s) & |Y| \leq \bar{Y}_s \\ 0 & \text{SONST} \end{cases}$$

$$\begin{aligned} f_{\alpha y \bar{y}_s}(\alpha, y,\bar{y}_s) = \; & p_{\alpha_r}\, \delta(\alpha - \alpha_r)\, f_{y\bar{y}_s}(y,\bar{y}_s \mid \alpha = \alpha_r) \\ & + p_{\alpha_f}\, \delta(\alpha - \alpha_f)\, f_{y\bar{y}_s}(y,\bar{y}_s \mid \alpha = \alpha_f) \end{aligned}$$

Kapitel 3.1: Mathematische Analyse des SAM-Schätzers

Die beiden Wahrscheinlichkeiten $p_{\alpha r}$ und $p_{\alpha f}$ lassen sich wie folgt berechnen:

$$p_{\alpha_r} = p(\{\eta \,||\, Y(k)| > \overline{Y}_s(k-1)\}) = \int_{-\infty}^{+\infty}\int_{-Y}^{+Y} f_{y\,\overline{y}_s}(y,\overline{y}_s)\, d\overline{y}_s\, dy$$

$$p_{\alpha_f} = p(\{\eta \,||\, Y(k)| \le \overline{Y}_s(k-1)\}) = \int_{-\infty}^{+\infty}\int_{-\infty}^{-Y} f_{y\,\overline{y}_s}(y,\overline{y}_s)\, d\overline{y}_s\, dy$$

$$+ \int_{-\infty}^{+\infty}\int_{+Y}^{+\infty} f_{y\,\overline{y}_s}(y,\overline{y}_s)\, d\overline{y}_s\, dy$$

Damit kann man nun für die gemeinsame Dichtefunktion schreiben:
Die Dichtefunktion wieder eingesetzt in die Formeln für die einzelnen Erwartungswerte liefert für den ersten Summanden:

$$E\{(1-\alpha_i)|Y(k)|\} = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} (1-\alpha_r)\, p_{\alpha_r}\, |y|\, f_{y\,\overline{y}_s}(y,\overline{y}_s \,|\, \alpha = \alpha_r)\, dy\, d\overline{y}_s$$
$$+ \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} (1-\alpha_f)\, p_{\alpha_f}\, |y|\, f_{y\,\overline{y}_s}(y,\overline{y}_s \,|\, \alpha = \alpha_f)\, dy\, d\overline{y}_s$$

Durch Einsetzen der bedingten Verbunddichte wird dieser Ausdruck zu:

$$E\{(1-\alpha_i)|Y(k)|\} = (1-\alpha_r) \int_{-\infty}^{+\infty}\int_{-Y}^{+Y} |y|\, f_{y\,\overline{y}_s}(y,\overline{y}_s)\, d\overline{y}_s\, dy$$

$$+ (1-\alpha_f) \int_{-\infty}^{+\infty}\int_{-\infty}^{-Y} |y|\, f_{y\,\overline{y}_s}(y,\overline{y}_s)\, d\overline{y}_s\, dy$$

$$+$$

$$+ (1-\alpha_f) \int_{-\infty}^{+\infty}\int_{+Y}^{+\infty} |y|\, f_{y\,\overline{y}_s}(y,\overline{y}_s)\, d\overline{y}_s\, dy$$

Für den zweiten Summanden ergibt sich analog:

$$E\{\alpha_i \overline{Y}_s(k-1)\} = \alpha_r \int_{-\infty}^{+\infty}\int_{-Y}^{+Y} \overline{y}_s \, f_{y\,\overline{y}_s}(y,\overline{y}_s)\, d\overline{y}_s\, dy$$

$$+ \alpha_f \int_{-\infty}^{+\infty}\int_{-\infty}^{-Y} \overline{y}_s \, f_{y\,\overline{y}_s}(y,\overline{y}_s)\, d\overline{y}_s\, dy$$

$$+ \alpha_f \int_{-\infty}^{+\infty}\int_{+Y}^{+\infty} \overline{y}_s \, f_{y\,\overline{y}_s}(y,\overline{y}_s)\, d\overline{y}_s\, dy$$

$$E\{\overline{Y}_s(k)\} = \int_{-\infty}^{+\infty}\int_{-Y}^{+Y} ((1-\alpha_r)|y| + \alpha_r \overline{y}_s)\, f_{y\,\overline{y}_s}(y,\overline{y}_s)\, d\overline{y}_s\, dy$$

$$+ \int_{-\infty}^{+\infty}\int_{-\infty}^{-Y} ((1-\alpha_f)|y| + \alpha_f \overline{y}_s)\, f_{y\,\overline{y}_s}(y,\overline{y}_s)\, d\overline{y}_s\, dy$$

$$+ \int_{-\infty}^{+\infty}\int_{+Y}^{+\infty} ((1-\alpha_f)|y| + \alpha_f \overline{y}_s)\, f_{y\,\overline{y}_s}(y,\overline{y}_s)\, d\overline{y}_s\, dy$$

Der Erwartungswert des Schätzwerts schreibt sich nun mit beiden Summanden zu:
Da die hier benötigte Verbunddichte nicht bekannt ist, wird folgende Vereinfachung gemacht:

$$Annahme: Y = Y(k) \text{ und } \overline{Y}_s = \overline{Y}_s(k-1) \text{ sind statistisch unabhÑngig.}$$

Diese Annahme ist streng genommen nur dann zulässig, wenn auch aufeinanderfolgende Werte des Eingangssignals statistisch unabhängig sind, da bei sehr kleinen Zeitkonstanten des SAM-Schätzers, das von ihm zurückgekoppelte Signal gerade dem um einen Takt verzögerten Wert des Eingangssignals entspricht. Da die Zeitkonstanten in der Praxis jedoch so gewählt werden, daß die SAM-Werte stark tiefpaßgefiltert sind, sind Eingangs- und Ausgangssignal des SAM-Schätzers statistisch unabhängig.

Den Erwartungwert kann man damit umschreiben in:

$$E\{\overline{Y}_s(k)\} = \int_{-\infty}^{+\infty}\int_{-Y}^{+Y} ((1-\alpha_r)|y| + \alpha_r \overline{y}_s)\, f_y(y) f_{\overline{y}_s}(\overline{y}_s)\, d\overline{y}_s\, dy$$

$$+ \int_{-\infty}^{+\infty}\int_{-\infty}^{-Y} ((1-\alpha_f)|y| + \alpha_f \overline{y}_s)\, f_y(y) f_{\overline{y}_s}(\overline{y}_s)\, d\overline{y}_s\, dy$$

$$+ \int_{-\infty}^{+\infty}\int_{+Y}^{+\infty} ((1-\alpha_f)|y| + \alpha_f \overline{y}_s)\, f_y(y) f_{\overline{y}_s}(\overline{y}_s)\, d\overline{y}_s\, dy$$

Aufspalten des Terms führt auf:

$$\int_{-\infty}^{+\infty}\int_{-Y}^{+Y} (1-\alpha_r)|y|\, f_y(y) f_{\overline{y}_s}(\overline{y}_s)\, d\overline{y}_s\, dy + \int_{-\infty}^{+\infty}\int_{-Y}^{+Y} \alpha_r \overline{y}_s\, f_y(y) f_{\overline{y}_s}(\overline{y}_s)\, d\overline{y}_s\, dy$$

$$+ \int_{-\infty}^{+\infty}\int_{-\infty}^{-Y} (1-\alpha_f)|y|\, f_y(y) f_{\overline{y}_s}(\overline{y}_s)\, d\overline{y}_s\, dy + \int_{-\infty}^{+\infty}\int_{-\infty}^{-Y} \alpha_f \overline{y}_s\, f_y(y) f_{\overline{y}_s}(\overline{y}_s)\, d\overline{y}_s\, dy$$

$$+ \int_{-\infty}^{+\infty}\int_{+Y}^{+\infty} (1-\alpha_f)|y|\, f_y(y) f_{\overline{y}_s}(\overline{y}_s)\, d\overline{y}_s\, dy + \int_{-\infty}^{+\infty}\int_{+Y}^{+\infty} \alpha_f \overline{y}_s\, f_y(y) f_{\overline{y}_s}(\overline{y}_s)\, d\overline{y}_s\, dy$$

Die nachfolgende Integration liefert:

$$E\{\overline{Y}_s(k)\} = \int_{-\infty}^{+\infty} (1-\alpha_r)|y|\, f_y(y)(F_{\overline{y}_s}(+Y) - F_{\overline{y}_s}(-Y))\, dy$$

$$+ \int_{-\infty}^{+\infty} \alpha_r \overline{y}_s\, f_y(y)(F_{\overline{y}_s}(+Y) - F_{\overline{y}_s}(-Y))\, dy$$

$$+ \int_{-\infty}^{+\infty} (1-\alpha_f)|y|\, f_y(y)(1-(F_{\overline{y}_s}(+Y) - F_{\overline{y}_s}(-Y)))\, dy$$

$$+ \int_{-\infty}^{+\infty} \alpha_f \overline{y}_s\, f_y(y)(1-(F_{\overline{y}_s}(+Y) - F_{\overline{y}_s}(-Y)))\, dy$$

Zur Lösung dieses Ausdrucks wird eine weitere Vereinfachung gemacht:

$$\textit{Annahme}: \overline{Y}_s \textit{ ist ein fester Wert.} \quad _ \quad f_{\overline{y}_s}(y) = \delta(y - \overline{Y}_s)$$

Für diese Annahme wurde vorausgesetzt, daß die Leistung des Eingangssignals konstant über der Zeit ist. Diese Bedingung ist in der Regel jedoch nicht erfüllt. Je länger aber die Zeitkonstanten für den SAM-Schätzer gewählt werden, desto geringer ist die Standard-

$$F_{\overline{y}_s}(+Y) - F_{\overline{y}_s}(-Y) = \begin{cases} 0 & |Y| \le \overline{Y}_s \\ 1 & |Y| > \overline{Y}_s \end{cases}$$

$$1 - (F_{\overline{y}_s}(+Y) - F_{\overline{y}_s}(-Y)) = \begin{cases} 1 & |Y| \le \overline{Y}_s \\ 0 & |Y| > \overline{Y}_s \end{cases}$$

abweichung im zeitlichen Verlauf des Schätzwerts. Dieser Zusammenhang wurde auch durch Versuchsmessungen im Rahmen dieser Arbeit belegt. Somit stellt die gemachte Vereinfachung für alle in der Praxis sinnvollen Zeitkonstanten eine gültige Näherung dar. Man erhält also:

Damit kann man den Erwartungswert schreiben als:

$$\begin{aligned} E\{\overline{Y}_s\} = \overline{Y}_s = & \int_{-\infty}^{-\overline{Y}_s} (1-\alpha_r)|y| f_y(y)dy + \int_{+\overline{Y}_s}^{+\infty} (1-\alpha_r)|y| f_y(y)dy \\ & + \int_{-\infty}^{-\overline{Y}_s} \alpha_r \overline{Y}_s f_y(y)dy + \int_{+\overline{Y}_s}^{+\infty} \alpha_r \overline{Y}_s f_y(y)dy \\ & + \int_{-\overline{Y}_s}^{+\overline{Y}_s} (1-\alpha_f)|y| f_y(y)dy + \int_{-\overline{Y}_s}^{+\overline{Y}_s} \alpha_f \overline{Y}_s f_y(y)dy \end{aligned}$$

Umformen und sortieren führt schließlich auf:

$$\overline{Y}_s = (1-\alpha_r)\left(\int_{-\infty}^{-\overline{Y}_s} |y|\, f_y(y)dy + \int_{+\overline{Y}_s}^{+\infty} |y|\, f_y(y)dy\right)$$

$$+ (1-\alpha_f) \int_{-\overline{Y}_s}^{+\overline{Y}_s} |y|\, f_y(y)dy$$

$$+ \overline{Y}_s \left(\alpha_r \left(\int_{-\infty}^{-\overline{Y}_s} f_y(y)dy + \int_{+\overline{Y}_s}^{+\infty} f_y(y)dy\right)\right.$$

$$\left. + \alpha_f \int_{-\overline{Y}_s}^{+\overline{Y}_s} f_y(y)dy\right)$$

Diese Formel stellt das allgemeine Ergebnis für den sich einstellenden mittleren Endwert des SAM-Schätzers dar. Um eine spezielle Lösung zu erhalten, muß diese Gleichung noch mit der entsprechenden Dichtefunktion gelöst werden.

Um einen Konflikt zwischen dem Zeit- und dem Endwert des SAM-Schätzers zu vermeiden, gilt im Folgenden auch:

$$\overline{Y}_s = Y_\infty$$

3.1.1 Lösung für Gleichverteilung

Die in Kapitel 3.1 hergeleitete Formel zur Berechnung des Endwerts eines SAM-Schätzers soll nun am Beispiel einiger spezieller Dichtefunktionen gelöst werden.

$$0 \leq \overline{Y}_s \leq a$$

Die Dichte einer Gleichverteilung läßt sich beschreiben durch:

$$f_y(y) = \left\{ \begin{array}{ll} \frac{1}{2a} & |y| \leq a \\ 0 & \text{SONST} \end{array} \right\}$$

Da der SAM-Wert laut Definition nicht negativ und auch nicht größer als der größtmögliche Eingangswert werden kann, gilt:
Eingesetzt in die Formel aus Kapitel 3.1 ergibt sich:

$$\begin{aligned} \overline{Y}_s = & (1-\alpha_r)\left(\int_{-a}^{-\overline{Y}_s} -\frac{y}{2a}\,dy + \int_{+\overline{Y}_s}^{+a} \frac{y}{2a}\,dy\right) \\ & + (1-\alpha_f)\left(\int_{-\overline{Y}_s}^{0} -\frac{y}{2a}\,dy + \int_{0}^{+\overline{Y}_s} \frac{y}{2a}\,dy\right) \\ & + \overline{Y}_s\left(\alpha_r\left(\int_{-a}^{-\overline{Y}_s} \frac{1}{2a}\,dy + \int_{+\overline{Y}_s}^{+a} \frac{1}{2a}\,dy\right) + \alpha_f \int_{-\overline{Y}_s}^{+\overline{Y}_s} \frac{1}{2a}\,dy\right) \end{aligned}$$

Integration des Ausdrucks und Einsetzen der Integrationsgrenzen führt auf:

$$\begin{aligned} \overline{Y}_s = & (1-\alpha_r)\left(-\frac{\overline{Y}_s^2}{4a} + \frac{a}{4} + \frac{a}{4} - \frac{\overline{Y}_s^2}{4a}\right) \\ & + (1-\alpha_f)\left(\frac{\overline{Y}_s^2}{4a} + \frac{\overline{Y}_s^2}{4a}\right) \\ & + \overline{Y}_s\left(\alpha_r\left(-\frac{\overline{Y}_s}{2a} + \frac{1}{2} + \frac{1}{2} - \frac{\overline{Y}_s}{2a}\right)\right. \\ & \left. + \alpha_f\left(\frac{\overline{Y}_s}{2a} + \frac{\overline{Y}_s}{2a}\right)\right) \end{aligned}$$

Kapitel 3.1.1: Lösung für Gleichverteilung

Ausrechnen und Zusammenfassen:

$$\overline{Y}_s = (1-\alpha_r)(\frac{a}{2} - \frac{\overline{Y}_s^2}{2a}) + (1-\alpha_f)\frac{\overline{Y}_s^2}{2a} +$$
$$\overline{Y}_s(\alpha_r(1-\frac{\overline{Y}_s}{a}) + \alpha_f\frac{\overline{Y}_s}{a})$$

Sortieren nach dem Argument führt auf:

$$\overline{Y}_s = (\frac{a}{2} - \frac{a}{2}\alpha_r) + \overline{Y}_s\alpha_r + \overline{Y}_s^2(-\frac{\alpha_r}{2a} + \frac{\alpha_f}{2a})$$

Die Lösungen dieser Gleichung sind gegeben durch:

$$\overline{Y}_{s_{1/2}} = a(\frac{1-\alpha_r \pm \sqrt{(1-\alpha_f)(1-\alpha_r)}}{\alpha_f - \alpha_r})$$

Sinnvolle SAM-Endwerte liefert nur die Lösung, bei der die Wurzel subtrahiert wird. Für den Fall, daß die Wurzel addiert wird, erhält man negative Werte wenn α_r größer ist als α_f, und Werte größer als *a* wenn α_r kleiner ist als α_f. Das Ergebnis für den sich einstellenden Endwert bei einem gleichverteilten Signal lautet damit:

$$Y_{\infty g} = a(\frac{1-\alpha_r - \sqrt{(1-\alpha_f)(1-\alpha_r)}}{\alpha_f - \alpha_r})$$

$$\alpha_r = \alpha_f \;_\; Y_{\infty g} = \frac{a}{2}$$

$$Y_{cg} = \frac{1}{Y_{\infty g}}\frac{a}{2} = \frac{1}{2}(\frac{\alpha_f - \alpha_r}{1-\alpha_r - \sqrt{(1-\alpha_f)(1-\alpha_r)}})$$

Für den Fall gleicher Alphas ist diese Gleichung nicht definiert, darum ist noch ein Sonderfall zu beachten:
Für den gesuchten Korrekturfaktor Y_{cg} ergibt sich damit:
Der Korrekturfaktor ist also unabhängig von dem Faktor *a*, der die Leistung des gleichverteilten Signals bestimmt. Zur späteren praktischen Verwendung soll hier noch die Beziehung zwischen *a* und der Standardabweichung eines gleichverteilten Prozesses genannt werden:

$$\sigma^2 = \int_{-a}^{+a}\frac{y^2}{2a}dy = \frac{y^3}{6a}\Big|_{-a}^{+a} = \frac{a^2}{3} \;_\; \sigma = a\sqrt{\frac{1}{3}}$$

Kapitel 3.1.1: Lösung für Gleichverteilung

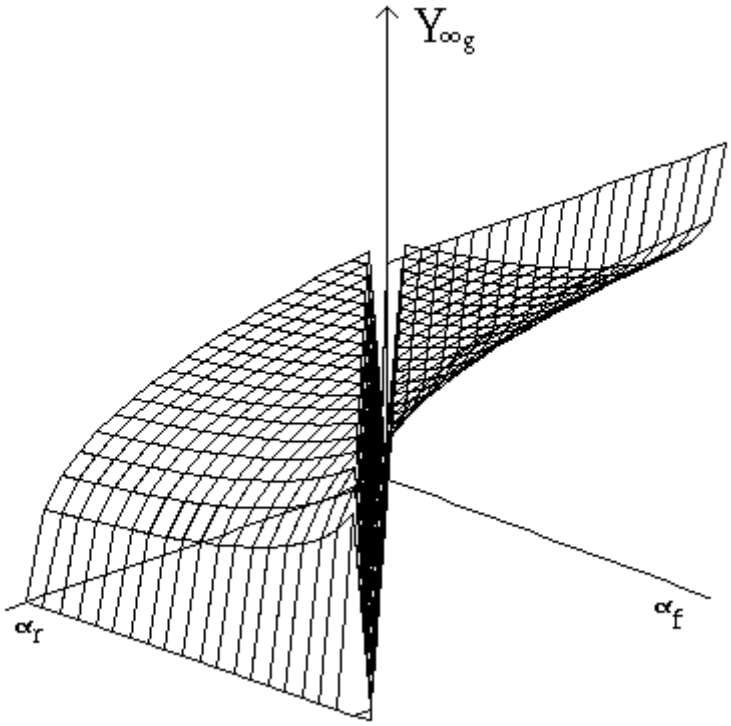

Abbildung 3.3: Schätzwert in Abhängigkeit von α_r und α_f

Der Einbruch der Werte in Abbildung 3.3 für $\alpha_r = \alpha_f$ folgt aus der Tatsache, daß die Lösungsgleichung für gleiche Alphas nicht definiert ist (Division durch Null).

3.1.2 Lösung für Laplaceverteilung

Die Dichtefunktion der Laplaceverteilung schreibt sich wie folgt:

$$f_y(y) = \frac{1}{\lambda} e^{-\lambda|y|}$$

$$\begin{aligned}
\overline{Y}_s = \frac{1}{\lambda}((1-\alpha_r)(&\int_{-\infty}^{-\overline{Y}_s} -y\, e^{\lambda y}\, dy + \int_{+\overline{Y}_s}^{+\infty} y\, e^{-\lambda y}\, dy) \\
+(1-\alpha_f)(&\int_{-\overline{Y}_s}^{0} -y\, e^{\lambda y}\, dy + \int_{0}^{+\overline{Y}_s} y\, e^{-\lambda y}\, dy) \\
+\overline{Y}_s(\alpha_r(&\int_{-\infty}^{-\overline{Y}_s} e^{\lambda y}\, dy + \int_{+\overline{Y}_s}^{+\infty} e^{-\lambda y}\, dy) \\
+\alpha_f(&\int_{-\overline{Y}_s}^{0} e^{\lambda y}\, dy + \int_{0}^{+\overline{Y}_s} e^{-\lambda y}\, dy)))
\end{aligned}$$

Einsetzen in das Ergebnis von Kapitel 3.1 und Berücksichtigung des Betrags im Exponenten der Dichtefunktion liefert:

Integration des Ausdrucks führt auf:

$$\begin{aligned}
\overline{Y}_s = \frac{1}{\lambda}((1-\alpha_r)(&-\frac{e^{\lambda y}}{2}(\lambda y-1)\Big|_{-\infty}^{-\overline{Y}_s} + \frac{e^{-\lambda y}}{2}(-\lambda y-1)\Big|_{+\overline{Y}_s}^{+\infty}) \\
+(1-\alpha_f)(&-\frac{e^{\lambda y}}{2}(\lambda y-1)\Big|_{-\overline{Y}_s}^{0} + \frac{e^{-\lambda y}}{2}(-\lambda y-1)\Big|_{0}^{+\overline{Y}_s}) \\
+\overline{Y}_s(\alpha_r(&\frac{1}{\lambda} e^{\lambda y}\Big|_{-\infty}^{-\overline{Y}_s} - \frac{1}{\lambda} e^{-\lambda y}\Big|_{+\overline{Y}_s}^{+\infty}) \\
+\alpha_f(&\frac{1}{\lambda} e^{\lambda y}\Big|_{-\overline{Y}_s}^{0} - \frac{1}{\lambda} e^{-\lambda y}\Big|_{0}^{+\overline{Y}_s})))
\end{aligned}$$

Einsetzen der Integrationsgrenzen und Zusammenfassen:

$$\lambda Y_{\infty l} = ((1-\alpha_f) + (\alpha_f - \alpha_r) e^{-\lambda Y_{\infty l}} + \alpha_f \lambda Y_{\infty l})$$

Die Gleichung ist vom Typ $Y_{\infty l} = g\,(Y_{\infty l})$, der sich nicht nach $Y_{\infty l}$ auflösen läßt. Zur Lösung wird darum das iterative Verfahren $Y_{\infty l\,k+1} = g\,(Y_{\infty l\,k})$ verwendet. Für $0 < \alpha_r < 1$ und $0 < \alpha_f < 1$ hat sich gezeigt, daß das Verfahren immer konvergiert.

3.1.2: Lösung für Laplaceverteilung

Eine weitere Zusammenfassung führt auf:

$$\lambda Y_{\infty l} = (1 + \frac{\alpha_f - \alpha_r}{1 - \alpha_f} e^{-\lambda Y_{\infty l}})$$

Diese vereinfachte Gleichung läßt sich jedoch nicht iterativ lösen, da das Verfahren hier nicht konvergiert.

Der schon in Kapitel 3.1.1 erwähnte Sonderfall gleicher Alphas ergibt für dieses Beispiel:

$$\alpha_r = \alpha_f \quad - \quad Y_{\infty l} = \frac{1}{\lambda}$$

Der Korrekturfaktor Y_{cl} für Laplaceverteilung lautet dann:

$$Y_{cl} = \frac{1}{\lambda Y_{\infty l}}$$

Der Korrekturfaktor für Laplaceverteilung läßt sich nur über den Endwert $Y_{\infty l}$ berechnen. Die Multiplikation mit λ macht jedoch auch diesen Korrekturfaktor unabhängig von der Leistung des Eingangssignals, die durch λ bestimmt wird.

Die Standardabweichung für diese Dichtefunktion berechnet sich aus λ zu:

$$\sigma = \frac{\sqrt{2}}{\lambda}$$

3.1.3 Lösung für Normalverteilung

Die Gaußdichte ist definiert durch:

$$f_y(y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{2\sigma^2}}$$

Analog wie in den vorangegangenen Beispielen wird diese Beziehung wieder in die Ergebnisformel aus Kapitel 3.1 eingesetzt. Aus Gründen der Übersichtlichkeit wird die Gleichung jedoch nun aufgeteilt und getrennt weiterbearbeitet. Es ergibt sich dann:

$$\overline{Y}_s = (1-\alpha_r)\left(\int_{-\infty}^{-\overline{Y}_s} |y| \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{2\sigma^2}} dy + \int_{+\overline{Y}_s}^{+\infty} |y| \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{2\sigma^2}} dy\right)$$
$$+ (1-\alpha_f) \int_{-\overline{Y}_s}^{+\overline{Y}_s} |y| \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{2\sigma^2}} dy$$

$$(\ _\ Teil\ 1\ )$$

$$+ \overline{Y}_s \left(\alpha_r \left(\int_{-\infty}^{-\overline{Y}_s} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{2\sigma^2}} dy + \int_{+\overline{Y}_s}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{2\sigma^2}} dy\right)\right.$$
$$\left. + \alpha_f \int_{-\overline{Y}_s}^{+\overline{Y}_s} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{2\sigma^2}} dy\right)$$

$$(\ _\ Teil\ 2\ )$$

Berücksichtigung der Betragsfunktion liefert:

$$(\ Teil\ 1\ _\ )$$

$$= (1-\alpha_r)\left(-\frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{-\overline{Y}_s} y\, e^{-\frac{y^2}{2\sigma^2}} dy + \frac{1}{\sqrt{2\pi}\sigma} \int_{+\overline{Y}_s}^{+\infty} y\, e^{-\frac{y^2}{2\sigma^2}} dy\right)$$
$$+ (1-\alpha_f)\left(-\frac{1}{\sqrt{2\pi}\sigma} \int_{-\overline{Y}_s}^{0} y\, e^{-\frac{y^2}{2\sigma^2}} dy + \frac{1}{\sqrt{2\pi}\sigma} \int_{0}^{+\overline{Y}_s} y\, e^{-\frac{y^2}{2\sigma^2}} dy\right)$$

Dieser Ausdruck läßt sich mit der Substitutionsregel integrieren:

$$\int f'(y)\, e^{f(y)}\, dy = e^{f(y)}$$

Kapitel 3.1.3: Lösung für Normalverteilung

Für den betrachteten Fall ergibt sich:

$$f(y) = -\frac{y^2}{2\sigma^2} _ f'(y) = -\frac{y}{\sigma^2}$$

$$_\int_a^b y\, e^{-\frac{y^2}{2\sigma^2}}\, dy = -\sigma^2 \int_a^b -\frac{y}{\sigma^2}\, e^{-\frac{y^2}{2\sigma^2}}\, dy = -\sigma^2\, e^{-\frac{y^2}{2\sigma^2}} \Big|_a^b$$

Integration des ersten Teils unter Ausnutzung der erhaltenen Beziehung ergibt:

$$= -\frac{\sigma^2}{\sqrt{2\pi}\sigma}((1-\alpha_r)(-e^{-\frac{y^2}{2\sigma^2}}\Big|_{-\infty}^{-\bar{Y}_s} + e^{-\frac{y^2}{2\sigma^2}}\Big|_{+\bar{Y}_s}^{+\infty})$$
$$+(1-\alpha_f)(-e^{-\frac{y^2}{2\sigma^2}}\Big|_{-\bar{Y}_s}^{0} + e^{-\frac{y^2}{2\sigma^2}}\Big|_{0}^{+\bar{Y}_s}))$$

Einsetzen der Integrationsgrenzen und Zusammenfassen:

$$= \frac{2\sigma}{\sqrt{2\pi}}((1-\alpha_r)(e^{-\frac{\bar{Y}_s^2}{2\sigma^2}}) + (1-\alpha_f)(1-e^{-\frac{\bar{Y}_s^2}{2\sigma^2}}))$$

Für die weitere Bearbeitung des zweiten Teils der Beziehung werden zuerst die Integrationsgrenzen umgeformt:

(Teil 2 _)

$$= \bar{Y}_s(\alpha_r(\int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{2\sigma^2}}\, dy - \int_{-\bar{Y}_s}^{+\bar{Y}_s} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{2\sigma^2}}\, dy)$$
$$+\alpha_f \int_{-\bar{Y}_s}^{+\bar{Y}_s} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{2\sigma^2}}\, dy)$$

Das erste Integral in diesem Ausdruck ergibt 1. Damit folgt:

$$= \bar{Y}_s(\alpha_r(1 - \int_{-\bar{Y}_s}^{+\bar{Y}_s} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{2\sigma^2}}\, dy) + \alpha_f \int_{-\bar{Y}_s}^{\bar{Y}_s} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{2\sigma^2}}\, dy)$$

Kapitel 3.1.3: Lösung für Normalverteilung

Die in diesem Ausdruck enthaltenen Integrale lassen sich nicht geschlossen lösen. Sie lassen sich jedoch durch die *errorfunction* beschreiben. Sie ist definiert durch:

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt$$

Um diese Beziehung nutzen zu können wird folgende Substitution gemacht:

$$\frac{y^2}{2\sigma^2} = t^2 _ dy = \sqrt{2}\sigma\, dt$$

Einsetzen und Korrektur der Integrationsgrenzen:

$$= \overline{Y}_s (\alpha_r (1 - \int_{\frac{-\overline{Y}_s}{\sqrt{2}\sigma}}^{\frac{\overline{Y}_s}{\sqrt{2}\sigma}} \frac{1}{\sqrt{\pi}} e^{-t^2}\, dt) + \alpha_f \int_{\frac{-\overline{Y}_s}{\sqrt{2}\sigma}}^{\frac{\overline{Y}_s}{\sqrt{2}\sigma}} \frac{1}{\sqrt{\pi}} e^{-t^2}\, dt)$$

Durch Ausnutzung der Symmetrie der Integrale erhält man:

$$= \overline{Y}_s (\alpha_r (1 - \frac{2}{\sqrt{\pi}} \int_0^{\frac{\overline{Y}_s}{\sqrt{2}\sigma}} e^{-t^2}\, dt) + \alpha_f \frac{2}{\sqrt{\pi}} \int_0^{\frac{\overline{Y}_s}{\sqrt{2}\sigma}} e^{-t^2}\, dt)$$

Mit der *errorfunction* läßt sich dies sofort schreiben als:

$$= \overline{Y}_s (\alpha_r (1 - erf(\frac{\overline{Y}_s}{\sqrt{2}\sigma})) + \alpha_f\, erf(\frac{\overline{Y}_s}{\sqrt{2}\sigma}))$$

Kapitel 3.1.3: Lösung für Normalverteilung

Der gesamte Ausdruck schreibt sich nun zu:

$$(\text{Teil 1} + \text{Teil 2})$$

$$\overline{Y}_s = \sqrt{\frac{2}{\pi}}\sigma((1-\alpha_r)e^{-\frac{\overline{Y}_s^2}{2\sigma^2}} + (1-\alpha_f)(1-e^{-\frac{\overline{Y}_s^2}{2\sigma^2}}))$$
$$+\overline{Y}_s(\alpha_r(1-erf(\frac{\overline{Y}_s}{\sqrt{2}\sigma})) + \alpha_f\, erf(\frac{\overline{Y}_s}{\sqrt{2}\sigma}))$$

Nach kurzer Umformung erhält man das Ergebnis:

$$\frac{Y_{\infty n}}{\sigma} = \sqrt{\frac{2}{\pi}}\frac{1-\alpha_f(1-e^{-\frac{Y_{\infty n}^2}{2\sigma^2}}) - \alpha_r e^{-\frac{Y_{\infty n}^2}{2\sigma^2}}}{1-\alpha_r(1-erf(\frac{Y_{\infty n}}{\sqrt{2}\sigma})) - \alpha_f\, erf(\frac{Y_{\infty n}}{\sqrt{2}\sigma})}$$

Auch diese Gleichung läßt sich, wie schon bei der Laplaceverteilung gesehen, nur iterativ lösen.

$$Y_{cn} = \frac{\sigma}{Y_{\infty n}}\sqrt{\frac{2}{\pi}}$$

Erschwerend wirkt sich noch aus, daß die *errorfunction* entweder über Tabelle oder durch numerische Integration ausgewertet werden muß.

Der Sonderfall für gleiche Alphas ergibt hier:

$$\alpha_r = \alpha_f \;-\; \frac{Y_{\infty n}}{\sigma} = \sqrt{\frac{2}{\pi}}$$

Der Korrekturfaktor Y_{cn} für die Normalverteilung berechnet sich somit zu:
Wie schon in den vorangegangenen Beispielen gesehen, ist auch der Korrekturfaktor für normalverteilte Signale unabhängig von deren Leistung.

3.2 Verifikation der Analyse für weiße Signale

Mit den in den drei vorangegangenen Kapiteln gewonnenen Formeln läßt sich nun prüfen, ob die der Herleitung aus Kapitel 3.1 zugrundeliegenden Vereinfachungen gerechtfertigt sind. Hierfür werden mittelwertfreie, weiße Signale mit konstanter Leistung auf den Eingang eines SAM-Schätzers gegeben. Aus dem Ausgangssignal des SAM-Schätzers wird der Mittelwert errechnet und mit den über die Formeln aus Kapitel 3.1.1 bis 3.1.3 errechneten Werten verglichen.

Die Zeitkonstanten des SAM-Schätzers sind für Wechselsignale nicht direkt bestimmbar, da sie abhängig von der Dichtefunktion des Signals sind. Als Anhaltspunkt werden deshalb Zeitkonstanten gewählt, die sich aus einer Sprungfunktion als Eingangssignal ergeben.

Die Zeitkonstante für den Anstieg von 10% auf 90% berechnet sich aus dem Parameter α zu:

$$t_{Anstieg}[ms] = \frac{\ln(0.1) - \ln(0.9)}{\ln(\alpha)} \frac{1000}{f_{Sample}}$$

Umgekehrt berechnet sich der Parameter α aus der Zeitkonstanten zu:

$$\alpha = e^{\frac{\ln(0.1) - \ln(0.9)}{t_{Anstieg}[ms]} \frac{1000}{f_{Sample}}}$$

Diese Formeln gelten gleichermaßen für den Anstieg von 10% auf 90%, wie für den Abfall von 90% auf 10%. Die Herleitung zu diesen Gleichungen befindet sich im Anhang B.

Kapitel 3.2: Verifikation der Analyse für weiße Signale

Die verwendeten Werte bei einer Abtastfrequenz von 24 kHz:

Zeitkonstanten:	Parameter α:
1 ms	0.91251475
2 ms	0.95525638
5 ms	0.98185641
10 ms	0.99088668
20 ms	0.99543291
50 ms	0.99817065
100 ms	0.99908491

Tabelle 3.1: SAM-Parameter

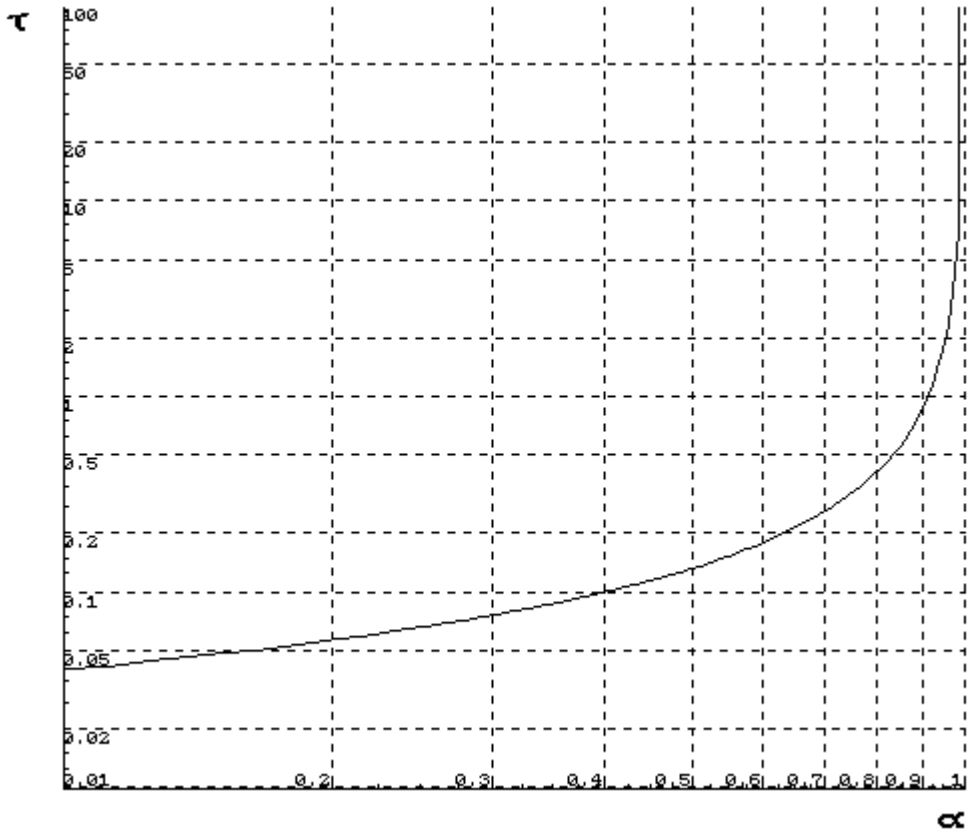

Abbildung 3.4: Zeitkonstante und Parameter

3.2.1 Verifikation für Gleichverteilung

Bei dem Testsignal für Gleichverteilung handelt es sich um eine Zufallsfolge von Fließkommazahlen, die zwischen -1.732 und +1.732 gleichverteilt sind. Dies entspricht einer Standardabweichung von 1. Die Länge des Testsignals umfaßt 65536 Werte.

Die Werte werden mit dem Zufallsgenerator des Entwicklungssystems der Sprache C erstellt. Der Generator wird vor dem ersten Aufruf mit der Systemzeit initialisiert, um einen zufälligen Startwert zu erhalten. Er arbeitet mit einem multiplikativen Kongruenzverfahren und liefert Zufallszahlen zwischen 0 und 2^{15}-1 mit einer Periodendauer von 2^{32} [4].

Zur Messung selbst werden der Reihe nach alle möglichen Parameterpaare, die sich mit den Werten aus Tabelle 3.1 bilden lassen, eingestellt. Mit jedem dieser Parameterpaare wird dann ein Schätzwertverlauf ermittelt. Da der Schätzwert in Abhängigkeit von den SAM-Parametern schwankt wird, zum Vergleich mit dem über die SAM-Endwertformel errechneten Wert, der Mittelwert über alle SAM-Werte gebildet. Der relative Fehler berechnet sich dann wie folgt:
Aus der Tatsache, daß der Korrekturfaktor Y_c direkt aus dem SAM-Endwert Y_∞ berechnet wird,

$$Relativer\ Fehler = \frac{Y_\infty - SAM - Mittelwert}{Y_\infty}$$

stellt der hier ermittelte relative Fehler einen guten Anhaltspunkt für die Richtigkeit der theoretisch errechneten Werte dar.

Relative Fehler in Abhängigkeit der SAM-Parameter:

$\tau_{\alpha r}$ / $\tau_{\alpha f}$	1 ms	2 ms	5 ms	10 ms	20 ms	50 ms	100 ms
1 ms	-7.6E-4	-2.0E-3	-1.6E-3	-9.0E-4	-2.2E-4	+4.4E-4	+6.0E-4
2 ms	+9.5E-4	-9.0E-4	-1.2E-3	-8.0E-4	-1.1E-4	+5.2E-4	+8.2E-4
5 ms	+2.0E-3	-2.0E-4	-1.3E-3	+1.2E-3	-7.3E-4	+3.1E-5	+5.2E-4
10 ms	+2.2E-3	-4.6E-4	-1.7E-3	-2.0E-3	-1.8E-3	-1.0E-3	-4.2E-4
20 ms	+1.6E-3	-1.3E-3	-2.8E-3	-3.3E-3	-3.4E-3	-3.0E-3	-2.3E-3
50 ms	+1.3E-3	-3.5E-3	-6.0E-3	-6.9E-3	-7.3E-3	-7.6E-3	-7.3E-3
100 ms	+5.8E-4	-5.3E-3	-1.0E-2	-1.2E-2	-1.3E-2	-1.4E-2	-1.5E-2

Tabelle 3.2: Relative Fehler für Gleichverteilung

Kapitel 3.2.1: Verifikation für Gleichverteilung

Der Tabelle 3.2 kann man entnehmen, daß der relative Fehler praktisch immer wesentlich kleiner als 1% (< 1.0E-2) ist. Für Anstiegszeitkonstanten, die kleiner als 10 ms sind, bleibt der Fehler sogar unter 0.15%.

Die Zunahme des Fehlers zu hohen Zeitkonstanten hin ($\tau_{\alpha r}$) läßt sich durch die daraus resultierenden langen Einschwingzeiten erklären. Tatsächlich erhält man bei der Messung mit längeren Meßsequenzen dementsprechend kleinere Fehler.

Die in der Analyse gemachte Annahme, daß der Ausgangswert des SAM-Schätzers ein konstanter Wert ist, wird für kleine SAM-Zeitkonstanten nur sehr schlecht erfüllt. Eine Tendenz zu größeren Fehlern ist, bedingt durch die Mittelung der SAM-Werte, für gleichverteilte Signale bei kleinen Zeitkonstanten jedoch trotzdem nicht ersichtlich.

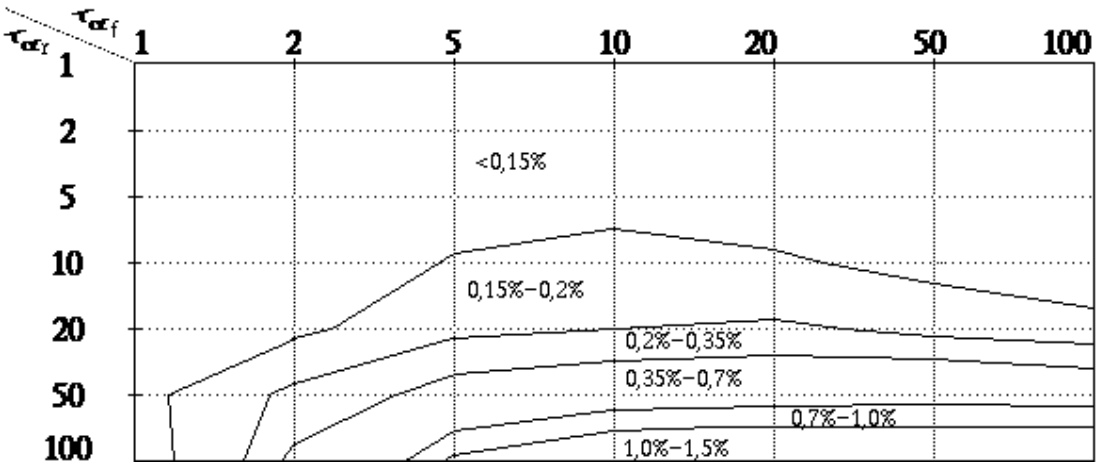

Abbildung 3.5: Konturendiagramm des relativen Fehlerbetrags für Gleichverteilung

3.2.2 Verifikation für Laplaceverteilung

Bei dem Testsignal für Laplaceverteilung handelt es sich ebenfalls um eine Zufallsfolge von Fließkommazahlen. Die Standardabweichung beträgt 1, was einem λ von 1.4142 entspricht. Die Länge des Testsignals umfaßt wiederum 65536 Werte. Die laplaceverteilte Meßsequenz wird aus einer Gleichverteilung durch die Dichtetransformation an der unten gezeigten Kennlinie erzeugt.

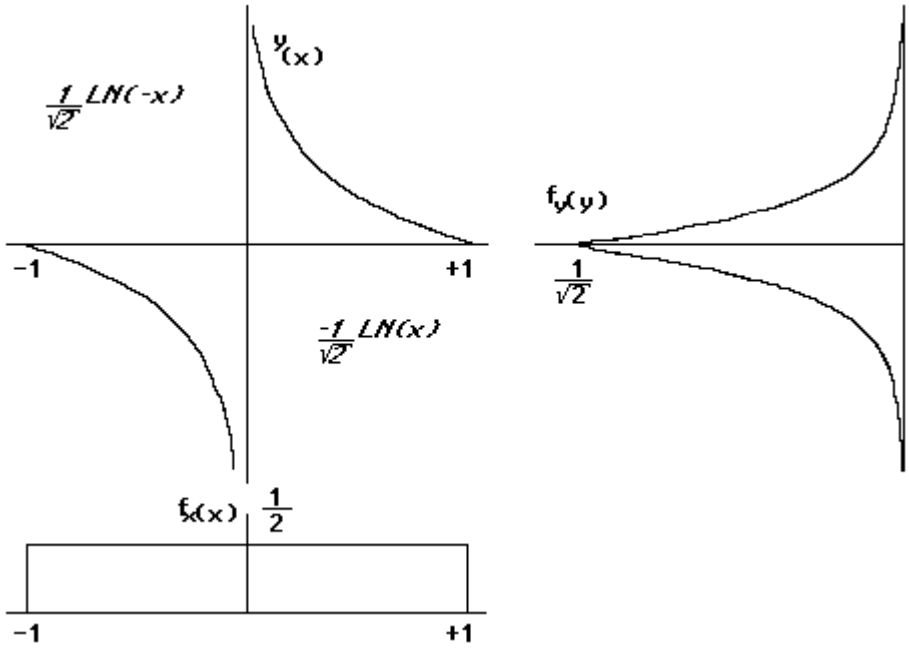

Abbildung 3.6: Dichtetransformation

Die Transformationskennlinie wird angesetzt als:

$$y = \left\{ \begin{array}{ll} \frac{-1}{\sqrt{2}} \ln(x) & x > 0 \\ \\ \frac{1}{\sqrt{2}} \ln(-x) & x < 0 \end{array} \right\}$$

Kapitel 3.2.2: Verifikation für Laplaceverteilung

Die Ableitung berechnet sich daraus zu:

$$\dot{y} = \begin{Bmatrix} \dfrac{-1}{\sqrt{2}\,x} & x > 0 \\ \\ \dfrac{1}{\sqrt{2}\,x} & x < 0 \end{Bmatrix}$$

y nach dem Argument auflösen ergibt:

$$x = \begin{Bmatrix} e^{-\sqrt{2}\,y} & y > 0 \\ \\ -e^{\sqrt{2}\,y} & y < 0 \end{Bmatrix}$$

Damit schreibt sich die transformierte Dichte:

$$f_y(x) = \frac{\frac{1}{2}}{|\,\dot{y}(x)\,|} = \frac{\frac{1}{2}}{\frac{1}{\sqrt{2}\,x}} = \frac{x}{\sqrt{2}}$$

Einsetzen von *x* ergibt:

$$f_y(y) = \begin{Bmatrix} \dfrac{1}{\sqrt{2}}\,e^{-\sqrt{2}\,y} & y > 0 \\ \\ \dfrac{1}{\sqrt{2}}\,e^{\sqrt{2}\,y} & y < 0 \end{Bmatrix}$$

Mit der Betragsfunktion lautet das Ergebnis schließlich:

$$f_y(y) = \frac{1}{\sqrt{2}}\,e^{-\sqrt{2}|y|}$$

Kapitel 3.2.2: Verifikation für Laplaceverteilung

Relative Fehler in Abhängigkeit der SAM-Paramter:

$\tau_{\alpha r}$ / $\tau_{\alpha f}$	1 ms	2 ms	5 ms	10 ms	20 ms	50 ms	100 ms
1 ms	+8.0E-6	-3.8E-3	-5.9E-3	-6.3E-3	-7.0E-3	-8.5E-3	-9.9E-3
2 ms	+3.9E-3	-6.8E-5	-2.2E-3	-2.6E-3	-2.5E-3	-3.3E-3	-4.4E-3
5 ms	+6.6E-3	+2.0E-3	-3.1E-4	-6.3E-4	-3.8E-4	+3.3E-4	+9.7E-4
10 ms	+7.0E-3	+2.7E-3	-1.3E-4	-6.4E-4	-2.6E-4	+9.9E-4	+2.5E-3
20 ms	+7.3E-3	+2.6E-3	-3.8E-4	-1.3E-3	-1.2E-3	+4.3E-4	+2.5E-3
50 ms	+5.9E-3	+1.8E-3	-1.9E-3	-3.4E-3	-3.9E-3	-2.7E-3	-3.2E-5
100 ms	+2.6E-3	-9.4E-4	-4.5E-3	-6.7E-3	-8.0E-3	-7.6E-3	-5.2E-3

Tabelle 3.3: Relative Fehler für Laplaceverteilung

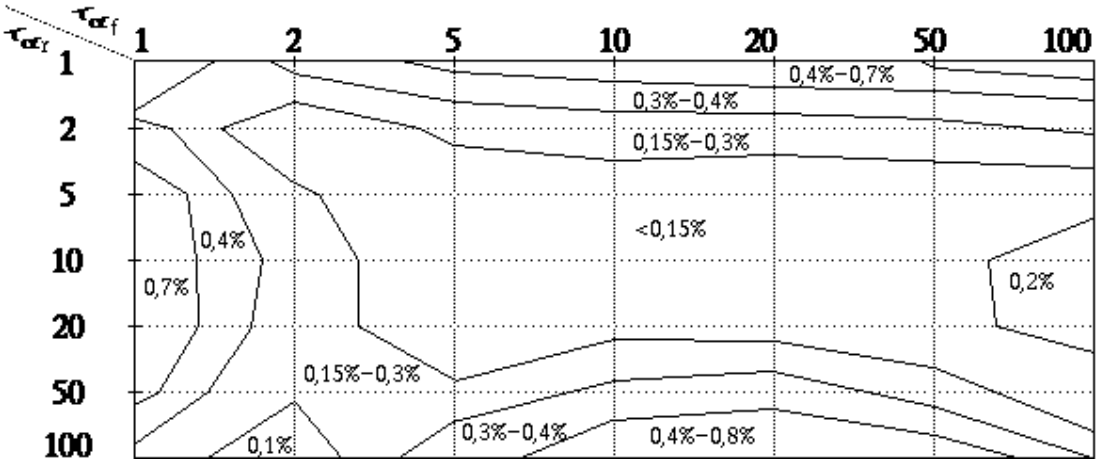

Abbildung 3.7: Konturendiagramm des relativen Fehlerbetrags für Laplaceverteilung

Abweichend vom Fehlerdiagramm für Gleichverteilung (Abbildung 3.5), läßt sich bei laplaceverteilten Meßsequenzen feststellen, daß bei langen Zeitkonstanten für $\tau_{\alpha r}$ der relative Fehler wesentlich kleiner ausfällt. Für $\tau_{\alpha r} < 20$ ms und $\tau_{\alpha f} > 5$ ms, kann man aus Abbildung 3.7 ersehen, daß der Fehler kleiner als 0.15% ist. Der Bereich des kleinsten Fehlers liegt somit, wie bei der Gleichverteilung, im üblichen Betriebsbereich des SAM-Schätzers.

3.2.3 Verifikation für Normalverteilung

Die Grundlage zur Erzeugung des Gaußprozesses bildet wiederum der schon erwähnte Zufallsgenerator. Dadurch, daß jeweils zwei Werte hintereinander vom Zufallsgenerator abgerufen werden, liefert er zwei voneinander statistisch unabhängige, zwischen 0 und 1 gleichverteilte Prozesse $g_1(k)$ und $g_2(k)$. Hieraus wird analytisch exakt durch die nichtlineare Transformation

$$n(k)=\sqrt{-2\ln g_1(k)}*\cos(\pi(1-2g_2(k)))$$

der weiße Gaußprozeß $n(k)$ erzeugt [2]. Die Länge der Meßsequenz wird auch hier auf 65536 Werte begrenzt und die Standardabweichung beträgt 1.

Relative Fehler in Abhängigkeit der SAM-Paramter:

$\tau_{\alpha r}$ / $\tau_{\alpha f}$	1 ms	2 ms	5 ms	10 ms	20 ms	50 ms	100 ms
1 ms	+5.8E-4	-2.6E-3	-3.8E-3	-3.7E-3	-3.3E-3	-1.5E-3	+6.1E-4
2 ms	+4.9E-3	+6.8E-4	-1.3E-3	-1.4E-3	-1.1E-3	-2.7E-5	+1.6E-3
5 ms	+8.9E-3	+4.3E-3	+1.0E-3	+3.1E-4	+5.7E-4	+1.5E-3	+2.6E-3
10 ms	+1.1E-2	+6.4E-3	+3.0E-3	+1.6E-3	+1.5E-3	+2.7E-3	+4.0E-3
20 ms	+1.3E-2	+8.0E-3	+4.7E-3	+3.3E-3	+2.7E-3	+3.9E-3	+5.8E-3
50 ms	+1.7E-2	+1.0E-2	+6.1E-3	+5.2E-3	+5.1E-3	+6.2E-3	+8.7E-3
100 ms	+2.0E-2	+1.3E-2	+7.1E-3	+5.9E-3	+6.3E-3	+8.6E-3	+1.2E-2

Tabelle 3.4: Relative Fehler für Normalverteilung

Kapitel 3.2.3: Verifikation für Normalverteilung

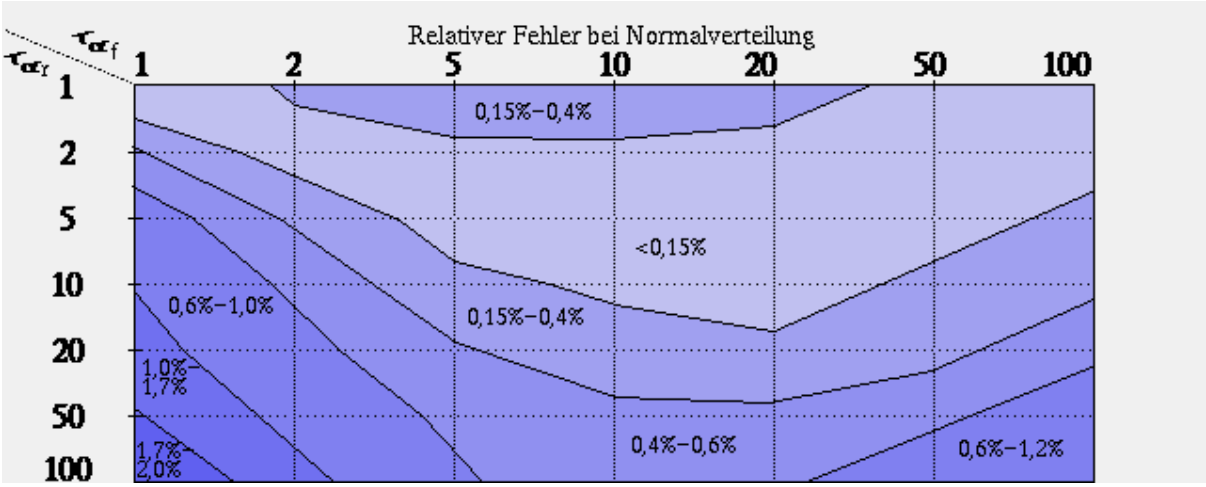

Abbildung 3.8: Konturendiagramm des relativen Fehlerbetrags für Normalverteilung

Der relative Fehler für normalverteilte Signale nimmt im Betriebsbereich des SAM-Schätzers Werte an, die kleiner als 0.15% sind. Für lange Anstiegs- und kurze Abfallzeitkonstanten wird der SAM-Endwert (mittlerer SAM-Wert) sehr klein. Da er bei der Berechnung des relativen Fehlers im Nenner steht, steigt dieser entsprechend an.

3.3 Zusammenfassung der Verifikation für weiße Signale

Die Meßwerte aus Tabelle 3.2 bis 3.4 belegen, daß die durch die Analyse des SAM-Schätzers gewonnene Formel gute Ergebnisse liefert, sofern das Eingangssignal wie hier nicht tiefpaßgefiltert ist. Für solche weißen Signale (Bsp.: Abbildung 3.9 & 3.10) kann man von einem relativen Fehler zwischen 10^{-4} und 10^{-2} (< 1.0%) ausgehen. Die Tatsache, daß die Meßergebnisse mit Musterfolgen ermittelt wurden, die auf 65536 Werte begrenzt waren, das entspricht bei 24 kHz einer Dauer von etwa 2.7 Sekunden, zeigt wie gut die errechneten SAM-Endwerte mit den im Versuch ermittelten Werten übereinstimmen. Für einzelne Schätzwerte wurden die Messungen mit Meßsequenzen doppelter Länge wiederholt, wodurch sich der relative Fehler noch weiter leicht verringerte. Die besten Ergebnisse im Versuch wurden mit den SAM-Zeitkonstanten 2 ms < $\tau_{\alpha r}$ < 10 ms und $\tau_{\alpha f}$ > 5 ms ermittelt. Dieser Bereich entspricht auch dem Betriebsbereich des SAM-Schätzers in der Praxis ($\tau_{\alpha f}$ > 60 ms). Der relative Fehler liegt dort bei allen überprüften Verteilungsdichten (Gleich-, Laplace- & Normalverteilung) unter 0.15%.

Kapitel 3.3: Zusammenfassung der Verifikation für weiße Signale

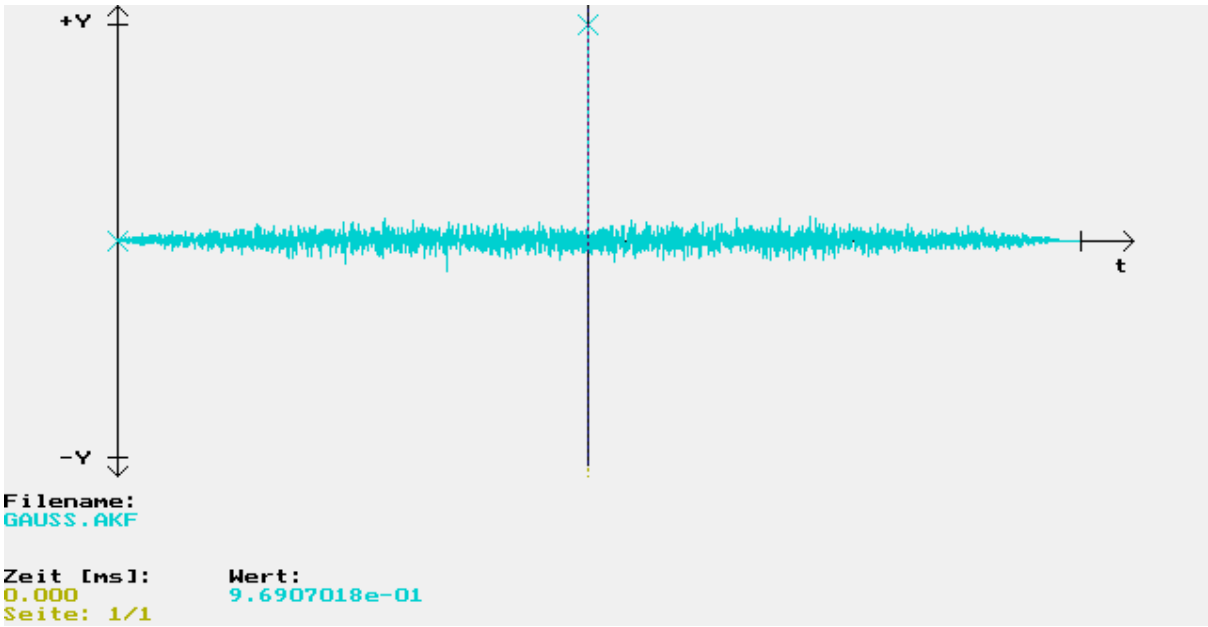

Abbildung 3.9: Autokorrelation von weißem gaußförmigem Rauschen

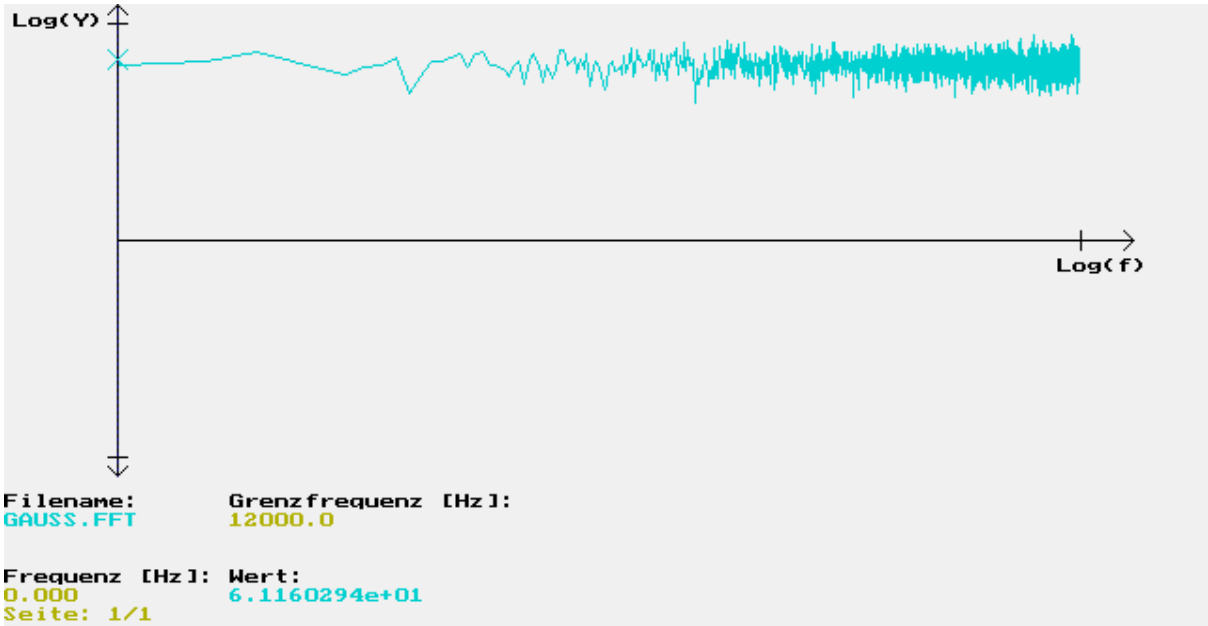

Abbildung 3.10: Spektrum von weißem gaußförmigem Rauschen

3.4 Verifikation der Analyse für farbige gaußverteilte Signale

Der Analyse des SAM-Schätzers liegt die Annahme zugrunde, daß aufeinanderfolgende Werte des Eingangssignals statistisch unabhängig sind. Diese Annahme setzt voraus, daß es sich bei den Eingangssignalen des SAM-Schätzers um weiße Rauschsignale handelt. Realistische Signale erfüllen diese Forderung jedoch nicht, vielmehr zeigen sie einen Frequenzgang, ähnlich dem eines tiefpaßgefilterten Rauschsignals. Zum Vergleich zeigen Abbildung 3.21 & 3.23 je ein Spektrum eines realistischen Signals, Abbildung 3.25 das Spektrum eines mit τ_e = 5 ms tiefpaßgefilterten normalverteilten Rauschsignals. Durch die Tiefpaßfilterung der Eingangssignale für den SAM-Schätzer, bei der aufeinanderfolgende Werte ihre statistische Unabhängigkeit verlieren (Abbildung 3.20, 3.22 & 3.24), soll nun überprüft werden, inwieweit die errechneten Schätzwerte trotzdem noch mit den praktisch ermittelten Werten übereinstimmen.

Beim Durchgang eines Gaußprozesses durch ein lineares System, liegt am Ausgang wiederum ein Gaußprozeß an [3]. Ein rekursives Filter erster Ordnung stellt ein solches lineares System dar.

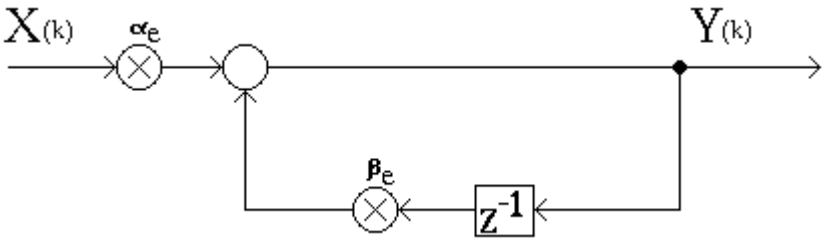

Abbildung 3.11: Filter erster Ordnung

Mit diesem Filter läßt sich bei geeigneter Wahl der Parameter ($0 \leq \alpha_e, \beta_e \leq 1$) ein Tiefpaß realisieren. Um beim Durchgang des Signals durch den Tiefpaß keine Signalleistung zu verlieren, wird nach dem Filter noch ein Leistungskorrekturfaktor, dessen Herleitung in Anhang A zu finden ist, mit dem Signal multipliziert.

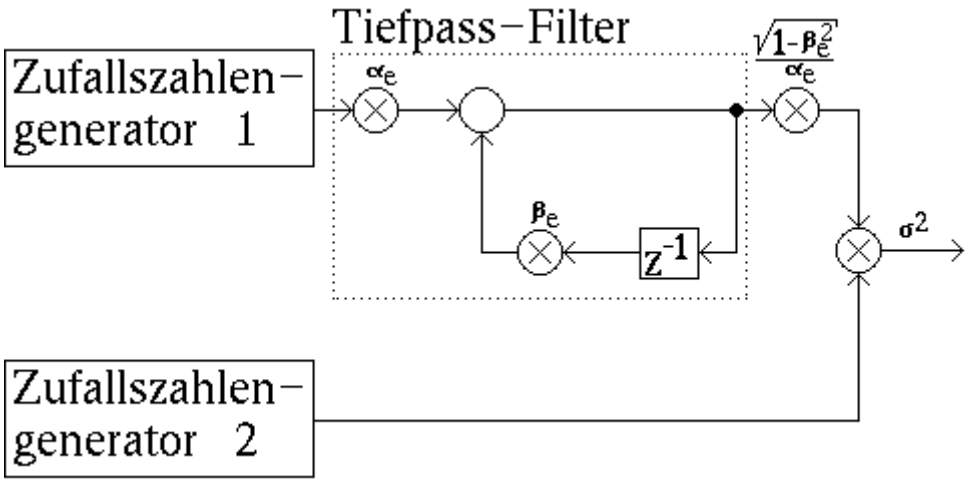

Abbildung 3.12: Leistungskorrekturfaktor

Zur Messung werden einige weiße und damit auch mittelwertfreie Musterfolgen eines Gaußprozesses mit jeweils 65536 Werten und einer Standardabweichung von 1 durch das schon vorgestellte rekursive Filter erster Ordnung gefiltert. Die Filterparameter werden nach der Formel aus Kapitel 3.2 so eingestellt, daß sich Zeitkonstanten zwischen 1 ms und 100 ms ergeben. Durch den Leistungskorrekturfaktor wird eine gleichbleibende Leistung der Prozesse, unabhängig vom Grad der Tiefpaßfilterung, erreicht. Die so erhaltenen Musterfolgen, die noch immer Normalverteilung aufweisen, dienen als Eingangssignale für den SAM-Schätzer.

Auf den folgenden Diagrammen ist der Betrag des relativen Fehlers, in Abhängigkeit von der Zeitkonstanten der Tiefpaßfilterung (τ_e) der Eingangssignale, aufgetragen.

Anmerkung:

Die relativen Fehlerbeträge, die an der Y-Achse der Diagramme 3.13 bis 3.18 abgelesen werden können, sind nicht in Prozent angegeben.

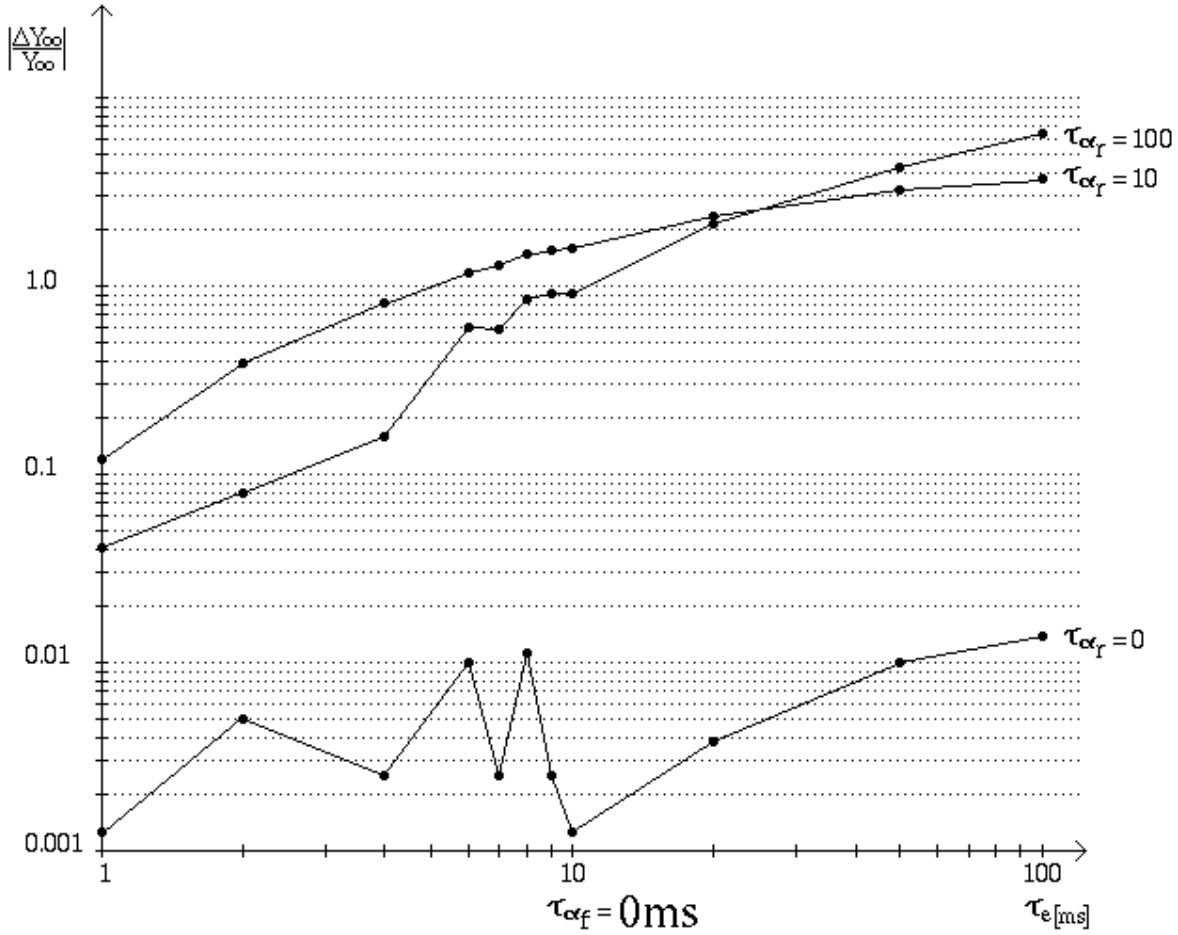

Abbildung 3.13: Relativer Fehler bei $\tau_{\alpha f}$ = 0 ms

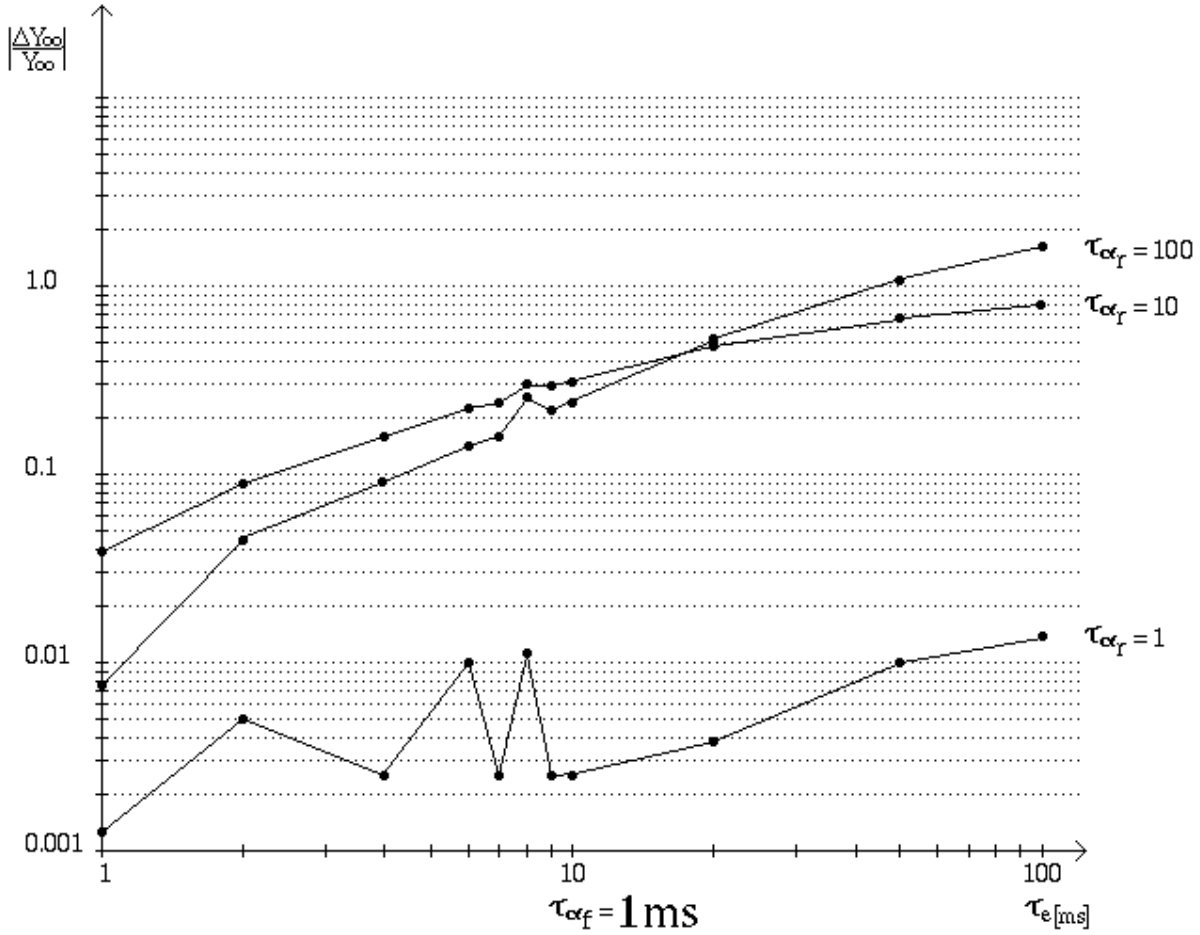

Abbildung 3.14: Relativer Fehler bei $\tau_{\alpha f} = 1$ ms

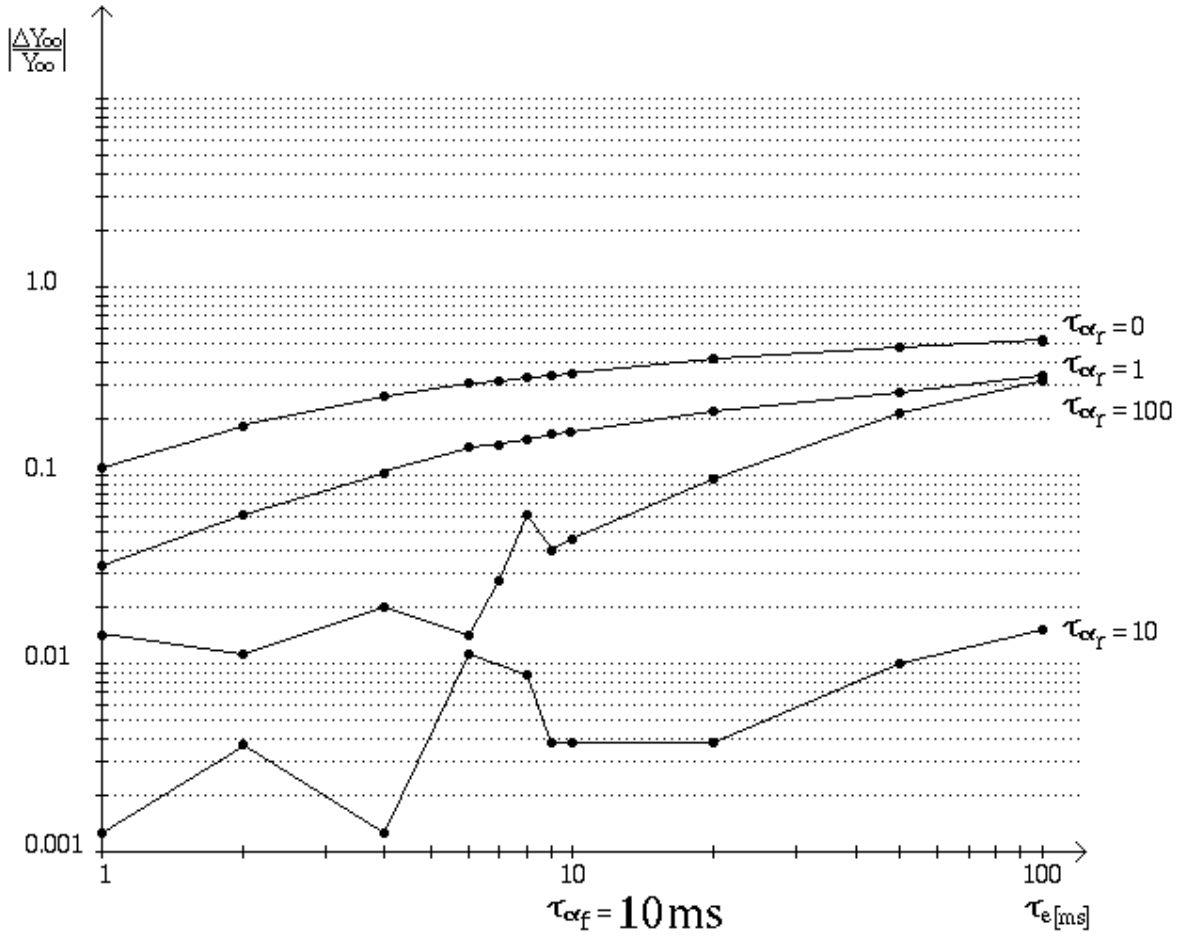

Abbildung 3.15: Relativer Fehler bei $\tau_{\alpha f} = 10$ ms

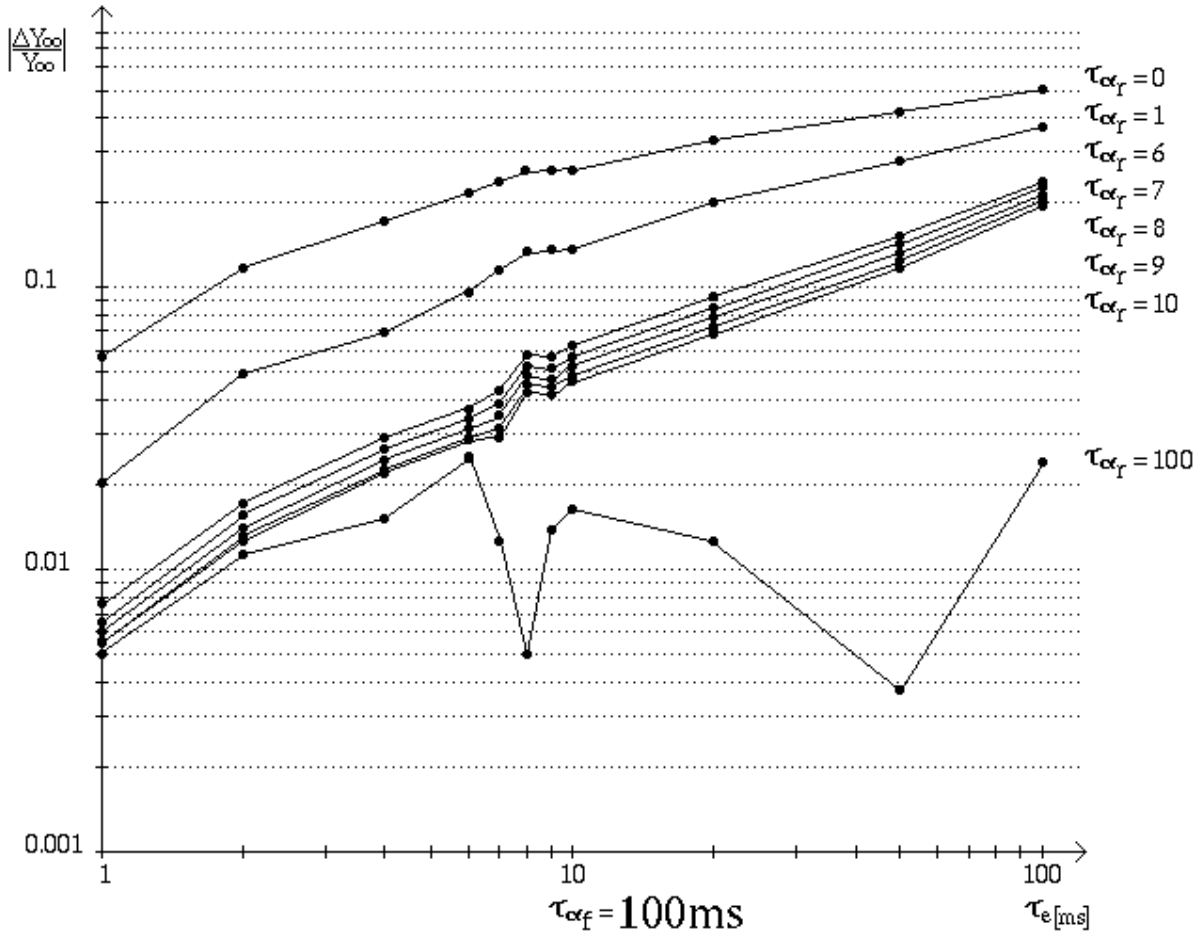

Abbildung 3.16: Relativer Fehler bei $\tau_{\alpha f}$ = 100 ms

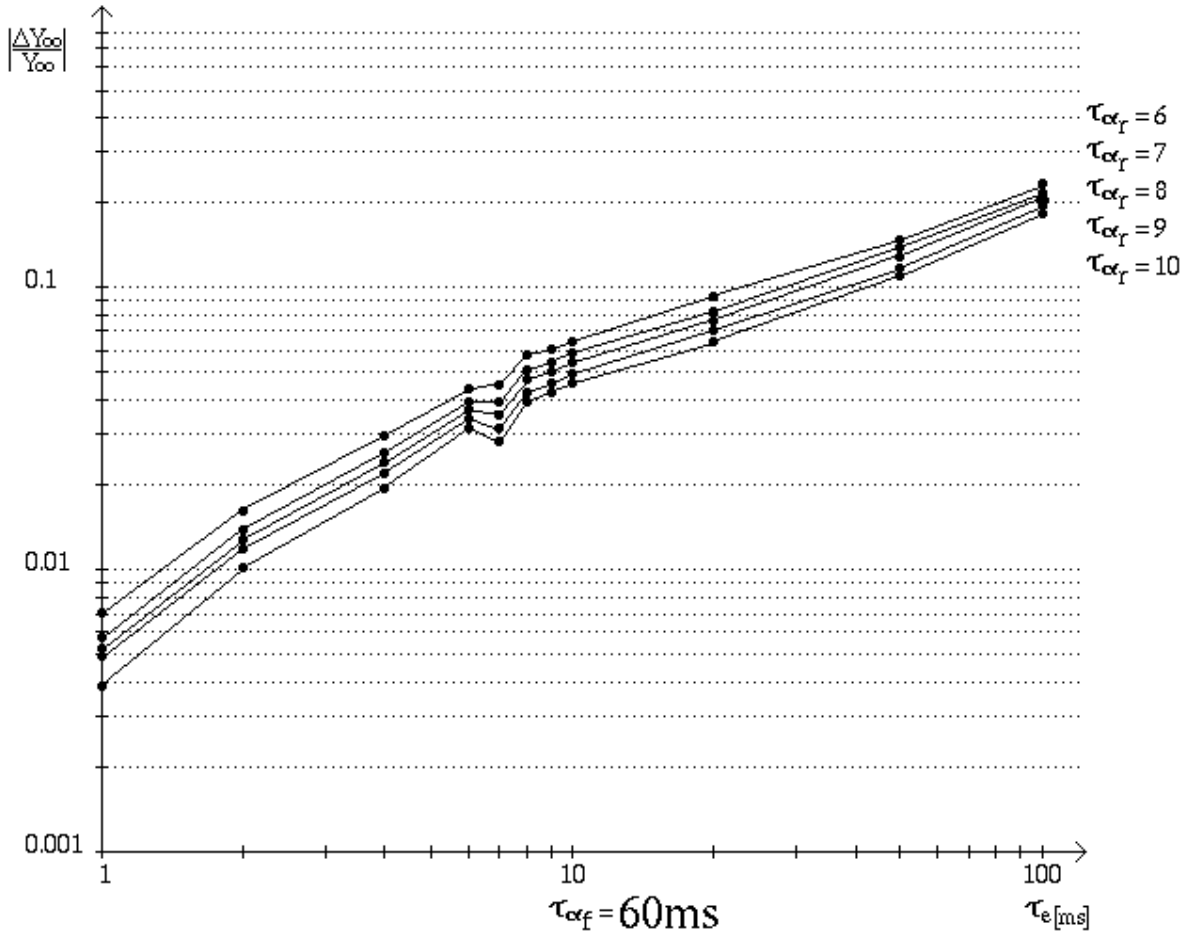

Abbildung 3.17: Relativer Fehler bei $\tau_{\alpha f}$ = 60 ms

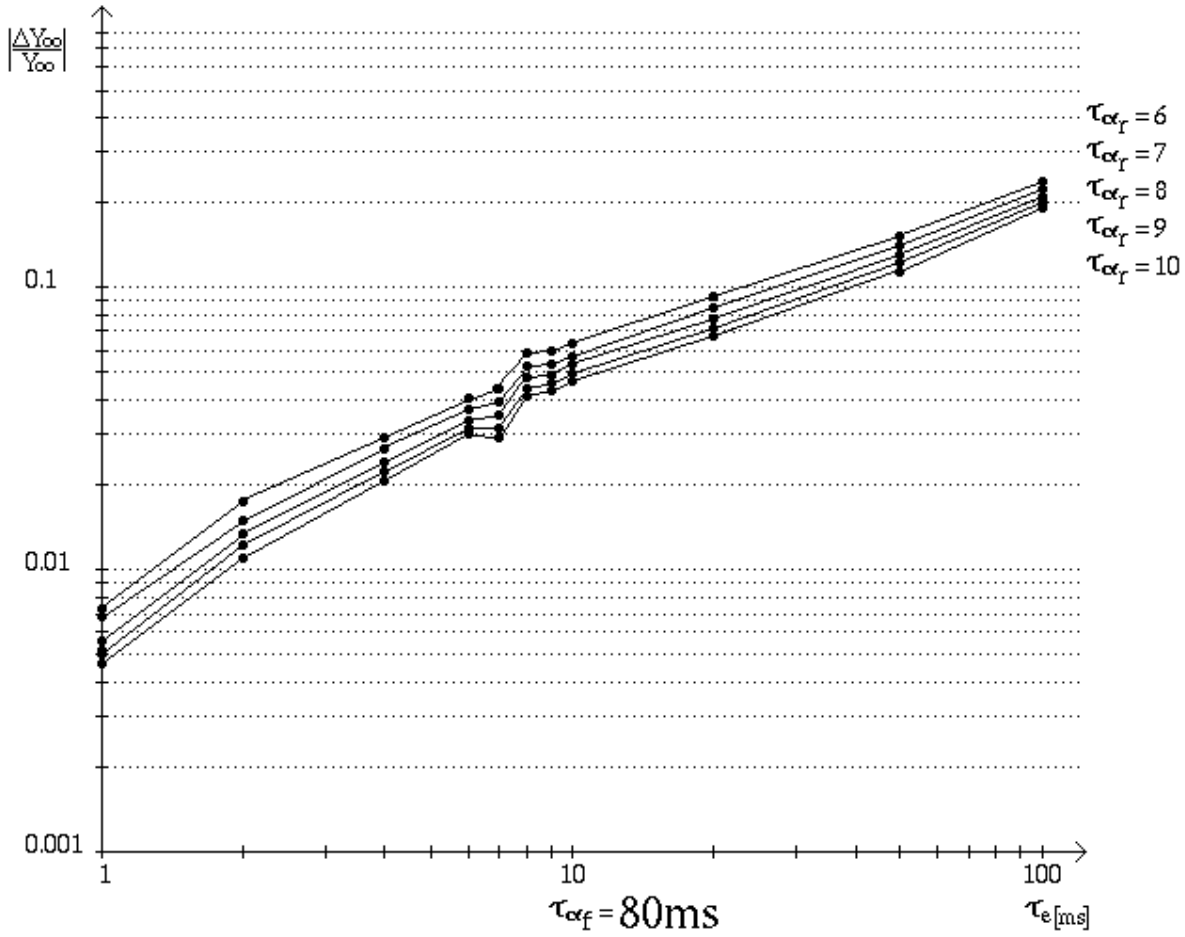

Abbildung 3.18: Relativer Fehler bei $\tau_{\alpha f}$ = 80 ms

Kapitel 3.4: Verifikation der Analyse für farbige gaußverteilte Signale

Bei der Auswertung der Ergebnisse für die tiefpaßgefilterten, normalverteilten Musterfolgen zeigt sich eine Zunahme des relativen Fehlers mit steigendem Grad der Tiefpaßfilterung der Eingangssignale. Diese Erkenntnis steht im Einklang mit der gemachten Annahme bei der Analyse des SAM-Schätzers. Danach sollten zwei aufeinanderfolgende Werte des Eingangssignals statistisch unabhängig sein. Mit steigenden Zeitkonstanten bei der Filterung erhöht sich jedoch auch das Maß der Korrelation zwischen den einzelenen Abtastwerten.

Für in der Praxis anzutreffende typische Hintergrundgeräusche, wie das Rauschen eines PC-Lüfters oder Straßenverkehrs-Lärm, können die analytisch ermittelten Werte aber eine gute Näherung darstellen. Für diese Signale, die sich additiv aus vielen unabhängigen Anteilen zusammensetzen, ist der Gaußprozeß ein wirklichkeitsnahes Modell [3].

Histogramme für einige Musterfolgen (Abbildung 3.19) zeigen, daß die hier verwendeten natürlichen Störsignale normalverteilt sind. Zwar besteht wegen des abfallenden Frequenzgangs (vgl. Abbildung 3.21, 3.23 & 3.25) eine statistische Abhängigkeit zwischen aufeinanderfolgenden Werten (Abbildung 3.20 & 3.22), doch liegt der relative Fehler bei SAM-Zeitkonstanten im Bereich von 6 ms < $\tau_{\alpha r}$ < 10 ms und $\tau_{\alpha f}$ > 60 ms deutlich unter 4% (Tabelle 3.5). Da der Korrekturfaktor Y_{cn}, im Gegensatz zum SAM-Endwert $Y_{\infty n}$, unabhängig von der Leistung des Eingangssignals ist, läßt er sich mit der SAM-Endwertformel aus Kapitel 3.1.3 für jeden SAM-Parametersatz direkt ermitteln.

$\tau_{\alpha r}$ / $\tau_{\alpha f}$	60 ms	80 ms	100 ms
6 ms	2.63 / 3.35	2.82 / 3.48	2.95 / 3.57
8 ms	2.16 / 2.88	2.38 / 3.06	2.55 / 3.18
10 ms	1.83 / 2.55	2.06 / 2.75	2.17 / 2.90

Tabelle 3.5: Relativer Fehler % bei PC-Lüftersignal / Straßenlärmsignal (Annahme: Gauß)

Kapitel 3.4: Verifikation der Analyse für farbige gaußverteilte Signale

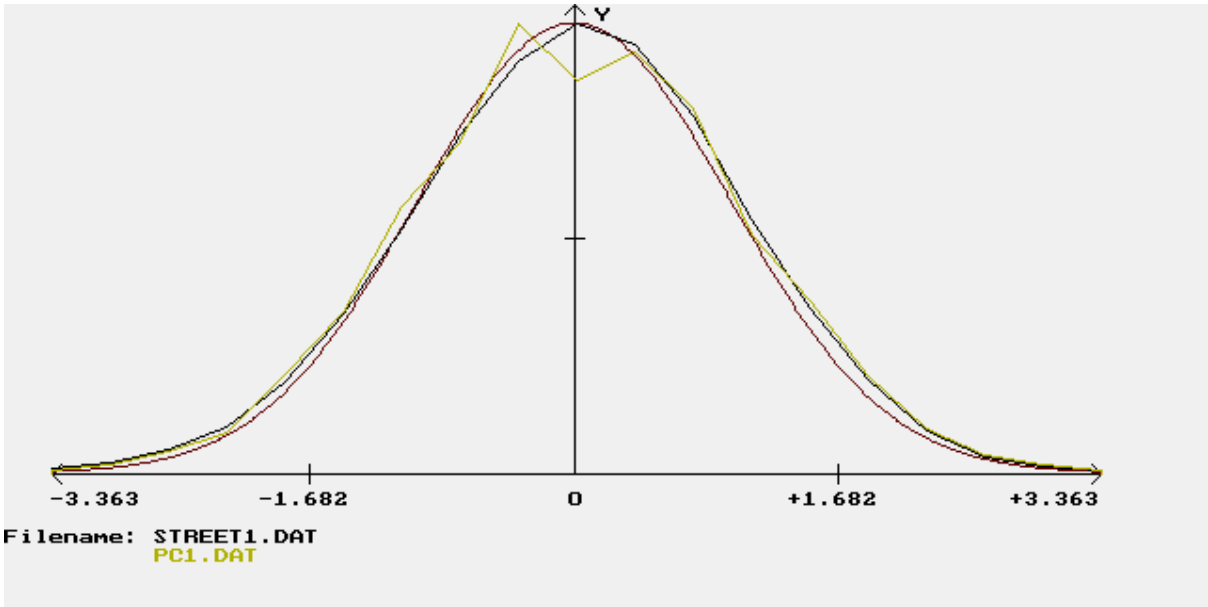

Abbildung 3.19: Histogramm eines PC-Lüftersignals und eines Straßenlärmsignals

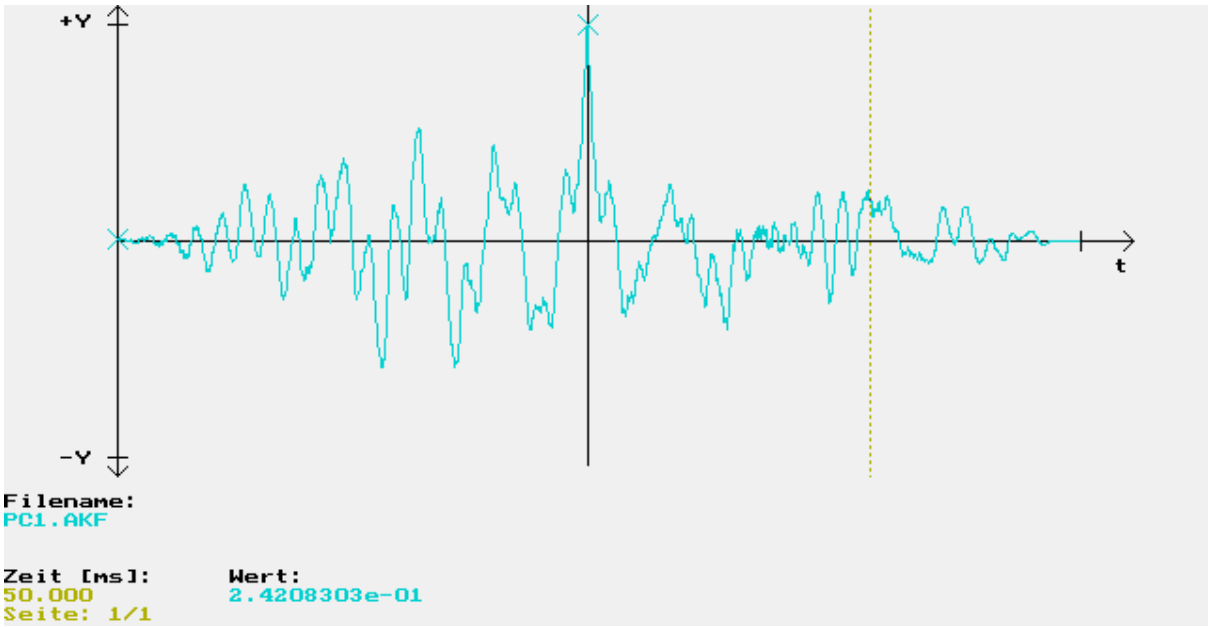

Abbildung 3.20: Autokorrelation des PC-Lüftersignals

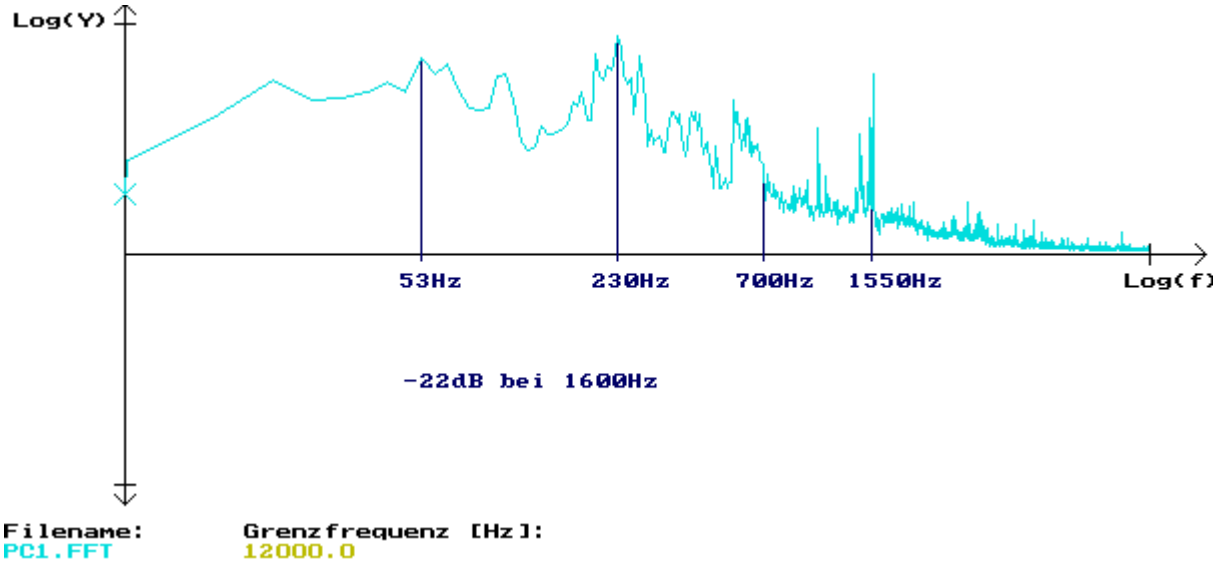

Abbildung 3.21: Spektrum des PC-Lüftersignals

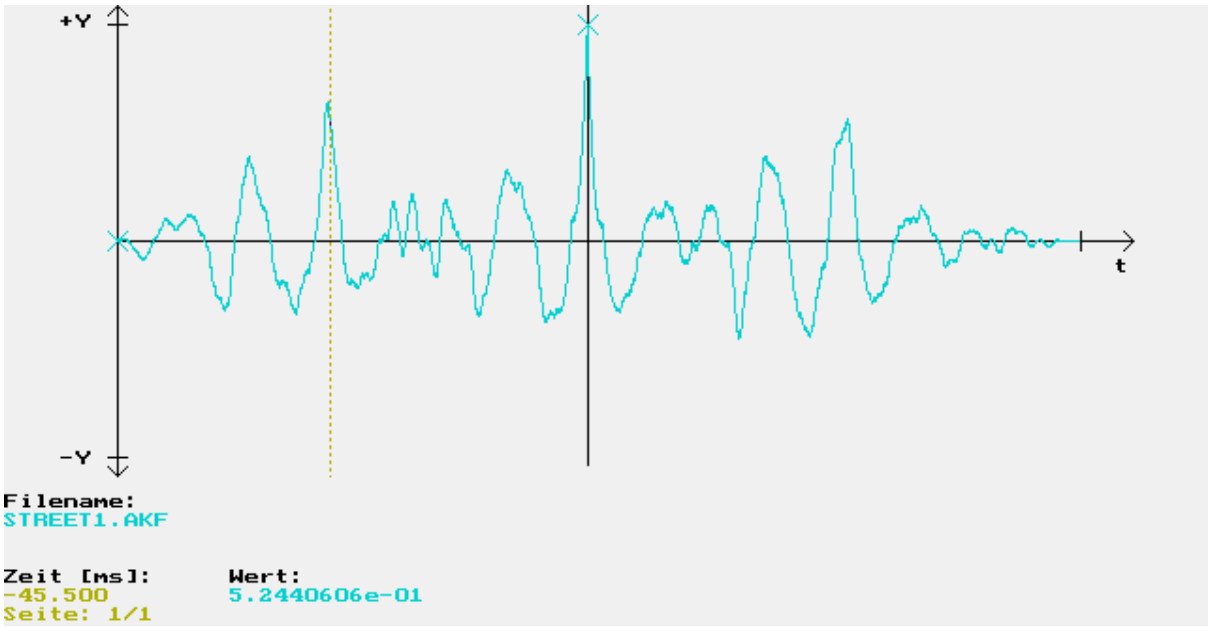

Abbildung 3.22: Autokorrelation des Straßenlärmsignals

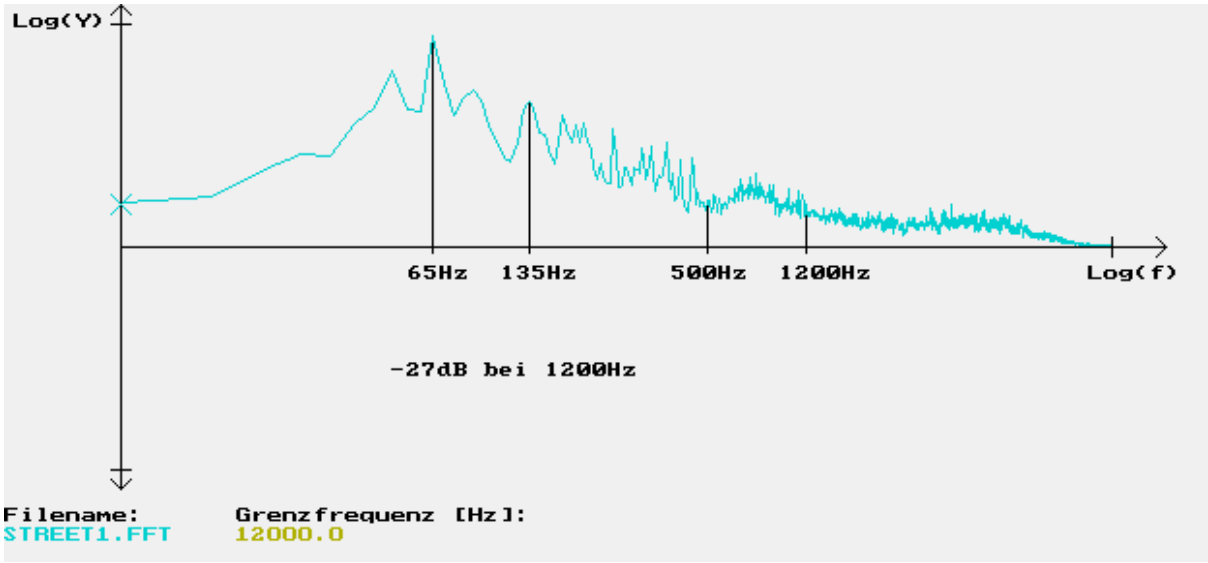

Abbildung 3.23: Spektrum des Straßenlärmsignals

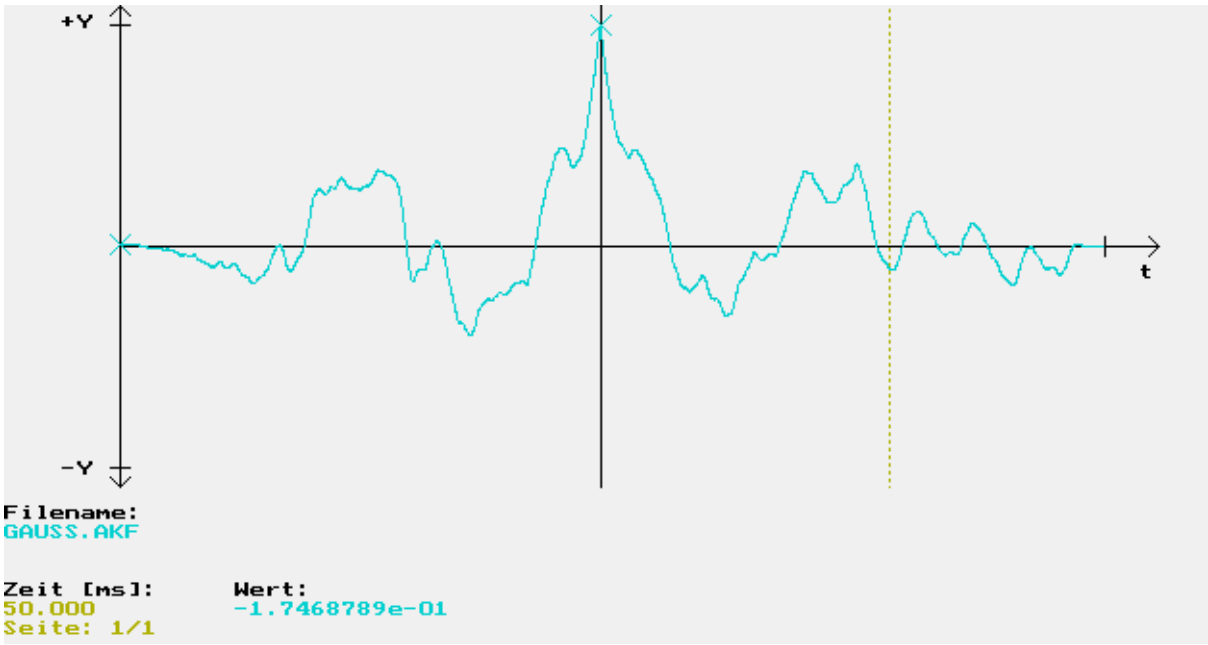

Abbildung 3.24: Autokorrelation eines gefilterten (τ_e = 5 ms) Gaußsignals

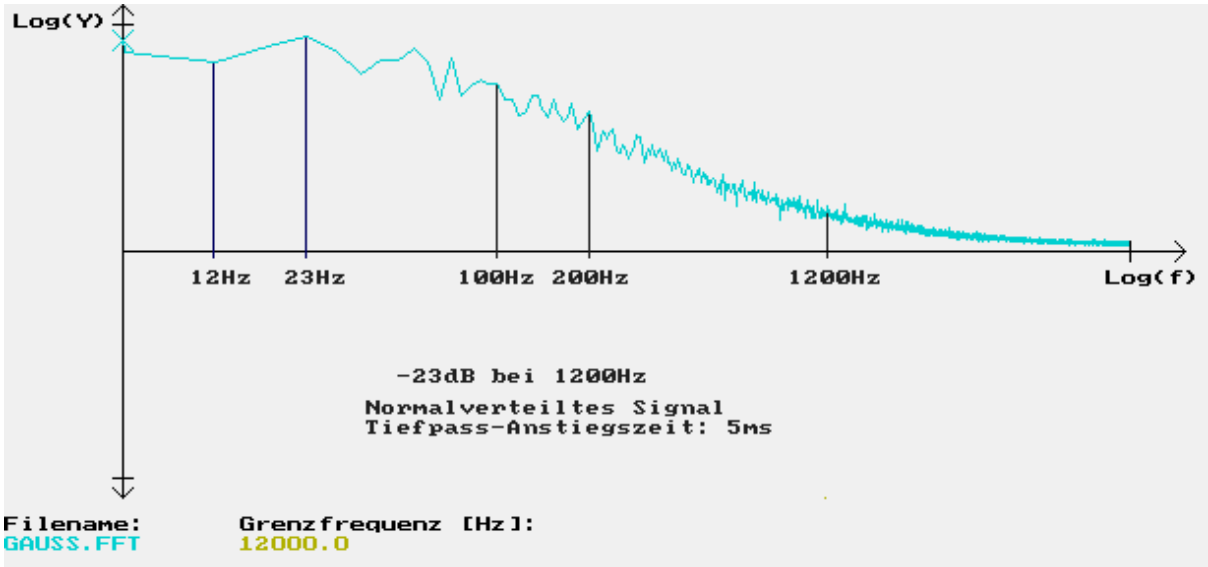

Abbildung 3.25: Spektrum eines gefilterten (τ_e = 5 ms) Gaußsignals

3.5 Abweichung zwischen Normal- und Laplaceverteilung

In der Praxis muß man sich entscheiden, für welche Dichtefunktion man den Korrekturfaktor des SAM-Schätzers einstellt. Während sich Störgeräusche wie Straßenlärm oder das Rauschen eines PC-Lüfters mit der Gaußverteilung modellieren lassen, stellt die Laplaceverteilung für bandbegrenzte (300 Hz ..... 3400 Hz) Sprachsignale eine recht gute Näherung dar [6].

Nachfolgende Tabelle veranschaulicht welcher relative Fehler durch die Annahme verursacht wird, es sei ein normalverteiltes Signal am Eingang des SAM-Schätzers, tatsächlich liegt aber ein laplaceverteiltes Signal an.

$$Relativer\ Fehler\ \% = 100\% \left| \frac{Y_{cn} - Y_{cl}}{Y_{cn}} \right|$$

$\tau_{\alpha r}$ / $\tau_{\alpha f}$	1 ms	2 ms	5 ms	10 ms	20 ms	50 ms	100 ms
1 ms	0.0	4.44	10.42	14.81	18.95	23.98	27.42
2 ms	4.28	0.0	5.99	10.54	14.93	20.35	24.10
5 ms	9.52	5.69	0.0	4.56	9.15	15.01	19.15
10 ms	12.96	9.62	4.39	0.0	4.58	10.63	15.03
20 ms	15.86	13.06	8.46	4.41	0.0	6.07	10.65
50 ms	18.87	16.78	13.11	9.70	5.77	0.0	4.59
100 ms	20.57	18.93	15.99	13.13	9.71	4.42	0.0

Tabelle 3.6: Abweichung von Y_c zwischen Normalverteilung und Laplaceverteilung

Kapitel 3.5: Abweichung zwischen Normal- und Laplaceverteilung

Die Korrekturfaktoren wurden mit dem Programm SPV ermittelt. Eine Tabelle aller Korrekturfaktoren für diese SAM-Zeitkonstanten findet sich im Anhang F. Es sei an dieser Stelle nochmals darauf hingewiesen, daß die genannten SAM-Zeitkonstanten nicht den tatsächlichen Zeitkonstanten für Wechselsignale entsprechen (siehe Kapitel 3.2).

Aus Tabelle 3.6 ist ersichtlich, daß der bei der obigen Annahme gemachte Fehler umso größer wird, je weiter die SAM-Zeitkonstanten $\tau_{\alpha r}$ und $\tau_{\alpha f}$ auseinanderliegen. Für in der Praxis gebräuchliche SAM-Parameter muß man mit einem Fehler von ca. 10% - 15% rechnen. Der Fehler in der Diagonalen ($\tau_{\alpha r} = \tau_{\alpha f}$) beträgt immer 0%, da der Korrekturfaktor für alle Dichtefunktionen bei gleichen Zeitkonstanten 1 beträgt. Dies ist aus der Definition des Korrekturfaktors ersichtlich, da er die Abweichung zwischen SAM-Endwert und AMN-Endwert korrigieren soll. Für gleiche Zeitkonstanten ($\tau_{\alpha r} = \tau_{\alpha f}$) verhält sich der SAM-Schätzer aber gerade wie ein AMN-Schätzer.

Tabelle 3.7 veranschaulicht den umgekehrten Fall. Es wird angenommen es sei ein laplaceverteiltes Signal am Eingang des SAM-Schätzers, tatsächlich liegt aber ein normalverteiltes Signal an.

$$Relativer\ Fehler\ \% = 100\% \left| \frac{Y_{cl} - Y_{cn}}{Y_{cl}} \right|$$

$\tau_{\alpha r}$ / $\tau_{\alpha f}$	1 ms	2 ms	5 ms	10 ms	20 ms	50 ms	100 ms
1 ms	0.0	4.65	11.63	17.38	23.39	31.54	37.77
2 ms	4.10	0.0	6.37	11.78	17.55	25.55	31.74
5 ms	8.69	5.39	0.0	4.78	10.07	17.66	23.69
10 ms	11.48	8.77	4.21	0.0	4.80	11.90	17.69
20 ms	13.69	11.55	7.80	4.22	0.0	6.47	11.91
50 ms	15.87	14.35	11.59	8.84	5.45	0.0	4.81
100 ms	17.06	15.91	13.79	11.61	8.85	4.23	0.0

Tabelle 3.7: Abweichung von Y_c zwischen Laplaceverteilung und Normalverteilung

In diesem Fall ist der relative Fehler noch etwas höher als im ersten Fall. Es ist demnach günstiger den Korrekturfaktor für normalverteilte Signale zu wählen oder aber einen günstigen Zwischenwert zu berechnen.

4 Kennlinie der Verstärkungssteuerung

In Kapitel 2 wurde erwähnt, daß sich die Kennlinie der Verstärkungssteuerung aus drei Bereichen zusammensetzt. Für kleine SAM-Werte nimmt die Verstärkung im Linearitätsbereich einen kleinen konstanten Faktor an. Bei etwas steigenden SAM-Werten soll der Verstärkungsverlauf im Expansionsbereich überproportional steigen. Für SAM-Werte, die einen Schwellwert Y_0 überschreiten, sollen die Verstärkungswerte im Kompressionsbereich mit steigenden SAM-Werten abnehmen. Den aus diesen Forderungen resultierenden Knick in der Kennlinie zwischen Expansionsbereich und Kompressionsbereich, der zu starken Verstärkungsschwankungen im Bereich des Schwellwerts Y_0 führen würde, eliminiert man durch den Einsatz eines Übergangspolynoms dritter Ordnung. Dieses Polynom wird so gewählt, daß es an seinem Anfangs- und Endpunkt stetig differenzierbar mit der Kennlinie ist (Abbildung 2.2).

Die 4 Bereiche der Kennlinie sind damit wie folgt definiert [1]:

$$V(\overline{Y}_s) = \begin{cases} \dfrac{Y_a\, d^{p-1}}{Y_0} & \overline{Y}_s \leq d\, Y_0 \\[2ex] \dfrac{Y_a}{Y_0}\left(\dfrac{\overline{Y}_s}{Y_0}\right)^{p-1} & d\, Y_0 < \overline{Y}_s < q\, Y_0 \\[2ex] P_{ueb}(\overline{Y}_s) & q\, Y_0 < \overline{Y}_s < r\, Y_0 \\[2ex] \dfrac{Y_a}{\overline{Y}_s} & r\, Y_0 < \overline{Y}_s \end{cases}$$

Zusammenfassung der Kennlinienparameter:

Y_0:	Übergangspunkt zwischen Expansionsbereich und Kompressionsbereich.
Y_a:	$(Y_a)^2$ entspricht der mittleren Ausgangsleistung.
d:	Relative Breite des Expansionsbereichs bezüglich Y_0.
p:	$(p\text{-}1)$ Ordnung des Polynoms im Expansionsbereich.
q:	Anfangspunkt des Übergangspolynoms in der Kennlinie bezüglich Y_0.
r:	Endpunkt des Übergangspolynoms in der Kennlinie bezüglich Y_0.

Für den Sonderfall $d = q$, fällt der zweite Bereich der Kennlinie weg. Das Übergangspolynom beginnt dann schon nach dem Linearitätsbereich.

Das Übergangspolynom schreibt sich wie folgt:

$$P_{ueb}(\overline{Y}_s) = a_0 + a_1\left(\frac{\overline{Y}_s}{Y_0} - 1\right) + a_2\left(\frac{\overline{Y}_s}{Y_0} - 1\right)^2 + a_3\left(\frac{\overline{Y}_s}{Y_0} - 1\right)^3$$

Die Forderung nach stetiger Differenzierbarkeit liefert die Parameter a_0 bis a_3:

$$\begin{pmatrix} 1 & q-1 & (q-1)^2 & (q-1)^3 \\ 1 & r-1 & (r-1)^2 & (r-1)^3 \\ 0 & 1 & 2(q-1) & 3(q-1)^2 \\ 0 & 1 & 2(r-1) & 3(r-1)^2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \frac{Y_a}{Y_0} \begin{pmatrix} q^{p-1} \\ \frac{1}{r} \\ (p-1)q^{p-2} \\ -\frac{1}{r^2} \end{pmatrix}$$

Kapitel 4: Kennlinie der Verstärkungssteuerung

Ein Beispiel für die Kennlinie mit Übergangspolynom ($d = q$) ist in der folgenden Abbildung gegeben.

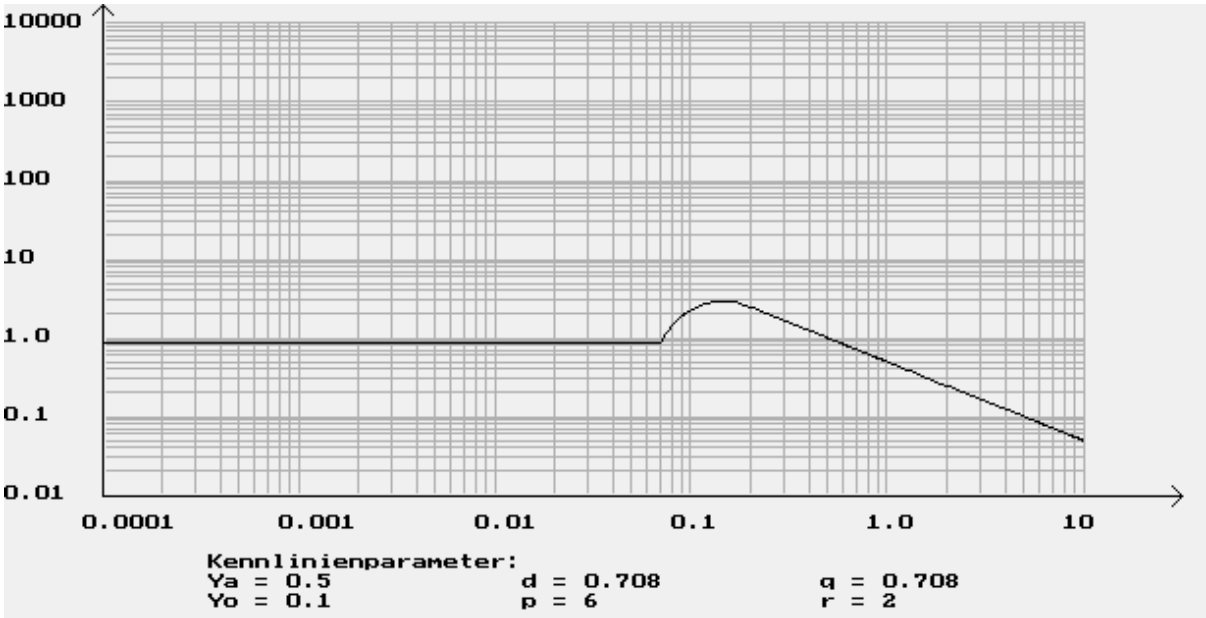

Abbildung 4.1: Kennlinie der Verstärkungssteuerung

4.1 Auswirkung der SAM-Parameter auf den Klirrfaktor

In diesem Kapitel soll überprüft werden, welchen Einfluß die Wahl der SAM-Parameter auf den Klirrfaktor des Ausgangssignals hat.

Die SAM-Parameter sind so einzustellen, daß ein Sinussignal der Frequenz 100 Hz am Ausgang der Verstärkungssteuerung weniger als 1% Klirrfaktor aufweist. Die Zeitkonstanten des SAM-Schätzers müssen demnach auf genügend lange Werte eingestellt werden um keine großen Verstärkungsschwankungen, die für den Klirrfaktor verantwortlich sind, hervorzurufen.

Der Klirrfaktor ist definiert durch:

$$Klirrfaktor\,\% = 100\% \sqrt{\frac{S_{II}^2 + S_{III}^2 + S_{IV}^2 + ...}{S_I^2 + S_{II}^2 + S_{III}^2 + S_{IV}^2 + ...}}$$

S_I ist in dieser Formel der spektrale Betrag der Grundwelle und S_{II} , S_{III} .... sind die spektralen Beträge der durch die Verzerrung entstehenden Oberwellen.

Für die Messungen wird ein 100 Hz-Sinussignal mit einer Amplitude von 1 verwendet. Das Signal umfaßt 8192 Werte, was bei 24 kHz etwa 340 ms entspricht. Die Kennlinienparameter sind identisch mit denen aus Abbildung 4.1.

Zur Bestimmung des Klirrfaktors wird das verstärkte Ausgangssignal mit einer FFT (Fast-Fourier-Transformation) in den Spektralbereich transformiert (Rechteckfenster). Die Rahmenlänge wird, um eine gute spektrale Auflösung zu erhalten, sehr hoch gewählt (in diesem Fall 4096 Werte). Von den beiden Rahmen im Spektralbereich, die aus dem Ausgangssignal mit 8192 Werten ermittelt werden, wird nur der Zweite betrachtet. Damit ist sichergestellt, daß keine Frequenzanteile, die beim Einschwingen des Systems entstehen, das Ergebnis verfälschen. Zur Berechnung des Klirrfaktors findet also nur der zweite Rahmen (Frame) Beachtung. Die absoluten Werte der Grundfrequenz bei 100 Hz und ihrer Oberwellen bei 200 Hz, 300 Hz usw. werden gemessen und mit obiger Formel für den Klirrfaktor ausgewertet.

Kapitel 4.1: Auswirkung der SAM-Parameter auf den Klirrfaktor

Klirrfaktor in Abhängigkeit der SAM-Paramter:

$\tau_{\alpha r}$ / $\tau_{\alpha f}$	1 ms	2 ms	5 ms	10 ms	20 ms	50 ms	100 ms
1 ms	29.9	22.1	13.2	8.1	4.6	1.9	0.9
2 ms	28.3	20.5	12.0	7.4	4.3	1.9	0.9
5 ms	22.6	16.0	9.4	5.9	3.6	1.6	0.7
10 ms	17.3	12.1	7.3	4.7	2.9	1.4	0.7
20 ms	12.5	8.9	5.4	3.6	2.3	1.1	0.6
50 ms	5.4	5.7	3.4	2.3	1.5	0.8	0.4
100 ms	3.8	2.3	2.4	1.6	1.0	0.6	0.3

Tabelle 4.1: Klirrfaktor % für ein 100 Hz-Sinussignal

Es zeigt sich, daß der Klirrfaktor mit steigenden SAM-Zeitkonstanten abnimmt. Für die in der Praxis relevanten SAM-Zeitkonstanten ist der Klirrfaktor stets kleiner als 2%.

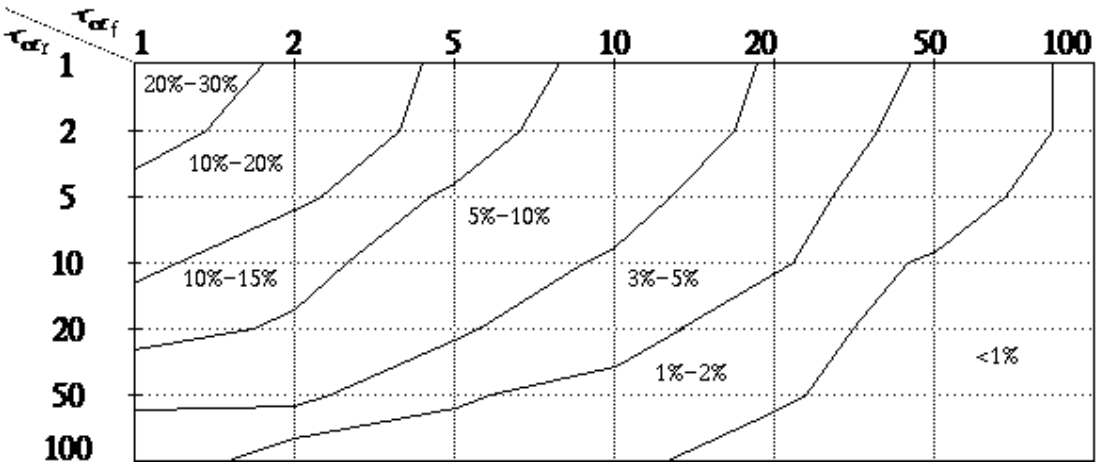

Abbildung 4.2: Konturendiagramm des Klirrfaktors

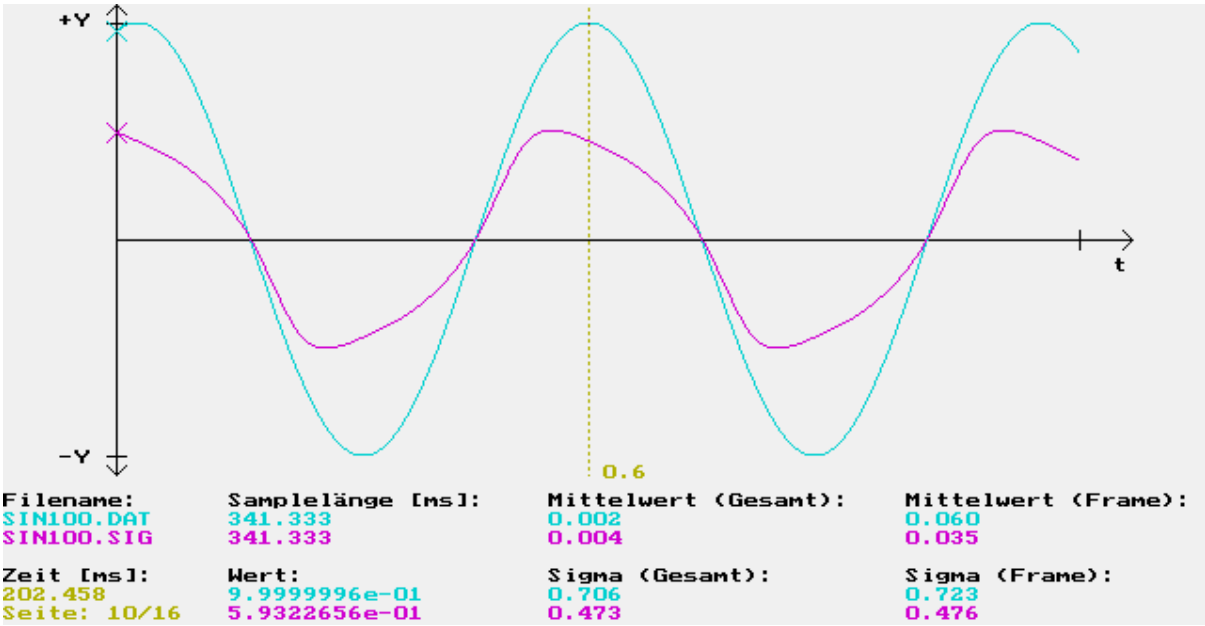

Abbildung 4.3: Verzerrtes Ausgangssignal mit $\tau_{\alpha r}$ = 2 ms und $\tau_{\alpha f}$ = 5 ms

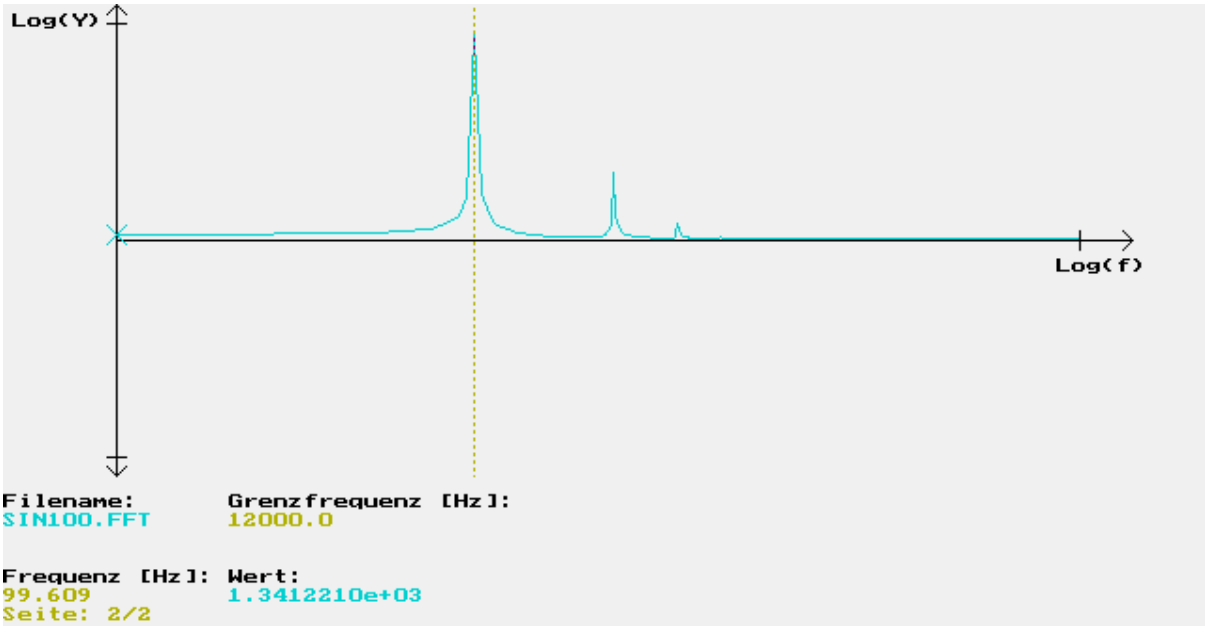

Abbildung 4.4: Spektrum des Ausgangssignals (Klirrfaktor 12%)

4.2 Das Glättungsfilter

Durch den Verlauf der nichtlinearen Kennlinie können die Verstärkungswerte, besonders im Bereich des Schwellwerts Y_0, stark schwanken. Der Einsatz des Übergangspolynoms war ein erster Schritt diese Schwankungen und die daraus resultierenden Verzerrungen zu verringern. Eine zusätzliche Möglichkeit besteht darin, die nach der Kennlinie anliegenden Verstärkungswerte zur Glättung mit einem Tiefpaß zu filtern.

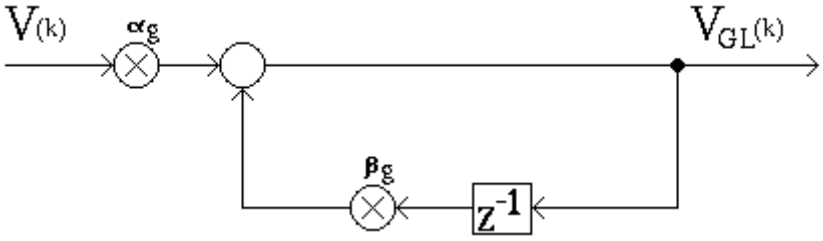

Abbildung 4.5: Glättungsfilter für Verstärkungswerte

Zur Überprüfung welche Auswirkungen das Glättungsfilter auf den Klirrfaktor ausübt, werden einige Messungen mit dem SAM-Parametersatz $\tau_{\alpha r}$ = 6 ms und $\tau_{\alpha f}$ = 60 ms durchgeführt. Folgende Tabelle zeigt die resultierenden Klirrfaktoren in Abhängigkeit von der Zeitkonstanten des Glättungsfilters. Um die Gesamtdynamik des Systems nicht merklich zu verändern, werden die Zeitkonstanten kleiner als die kleinste eingestellte SAM-Zeitkonstante gewählt. Der Klirrfaktor wird bei diesem SAM-Parametersatz für noch größere Zeitkonstanten des Glättungsfilters so klein, daß er mit der beschriebenen Meßmethode (FFT mit Rechteckfenster) nicht meht bestimmt werden kann.

Zeitkonstanten (τ_{gl}):	Klirrfaktor %:
0 ms	1.3
1 ms	1.1
2 ms	0.8
3 ms	0.6
4 ms	0.5
5 ms	0.4

Tabelle 4.2: Klirrfaktoren mit Glättungsfilter

Beim Einsatz des Glättungsfilters zeigt sich, daß schon für kleine Zeitkonstanten eine deutliche Verbesserung des Klirrfaktors erreicht werden kann.

Kapitel 4.2: Das Glättungsfilter

Bei einem Leistungssprung des Eingangssignals, wie er etwa beim Übergang von Hintergrundrauschen zu lauter Sprache auftritt, steigen die SAM-Parameter derart an, daß die Kennlinie vom Linearitätsbereich bis zum Kompressionsbereich durchlaufen wird. Die Anstiegszeitkonstante des SAM-Schätzers wird in der Regel größer als 5 ms gewählt, um kurzzeitige Störspitzen zu unterdrücken. Das führt dazu, daß der SAM-Wert nur relativ langsam steigt und somit die Verstärkung bei einem Leistungssprung erst im Expansionsbereich anschwillt, um dann wieder im Kompressionsbereich abgesenkt zu werden. Das resultierende Übersteuern des Signals läßt sich durch eine Verlängerung der Glättungsfilterzeitkonstanten vermeiden. Man erreicht dadurch quasi eine "Durchtunnelung" des Expansionsbereichs und gelangt "direkt" vom Linearitätsbereich in den Kompressionsbereich.

Abbildung 4.6 zeigt ein normalverteiltes Signal mit einem Leistungssprung von $\sigma = 0.05$ auf $\sigma = 1$ (Standardabweichung) und den mit Y_c korrigierten SAM-Schätzwertverlauf. Mit diesem Signal werden die Verstärkungsverläufe bei unterschiedlichen Glättungsfilterzeitkonstanten ermittelt (Abbildung 4.7). Die Kennlinienparameter werden, im Vergleich zu Abbildung 4.1, nicht verändert.

Zeitkonstanten (τ_{gl}):	Überschwinger %:
0 ms	231.4
5 ms	101.9
10 ms	68.3
15 ms	52.1
20 ms	42.4
25 ms	36.0
30 ms	31.3
35 ms	27.8
40 ms	25.1
45 ms	23.0
50 ms	21.3

Tabelle 4.3: Maximale Überschwinger %

Die kurzzeitige Verstärkungsüberhöhung, die sich sehr störend für den Hörer bemerkbar macht, nimmt wie erwartet mit steigenden Zeitkonstanten für das Glättungsfilter ab. Der Nachteil dieses Verfahrens liegt darin, daß es mit zunehmenden Zeitkonstanten auch immer länger dauert bis der erwünschte Verstärkungswert erreicht wird. Als guter Kompromiß erweisen sich Zeitkonstanten zwischen 25 ms und 35 ms.

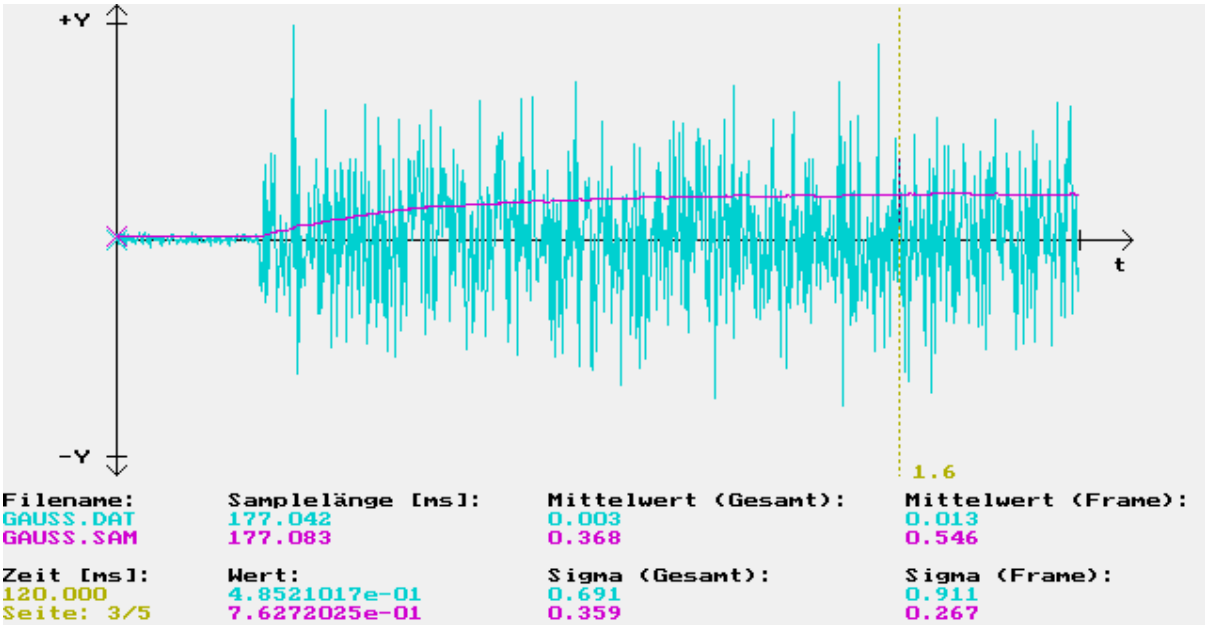

Abbildung 4.6: Leistungssprung und zugehöriger SAM-Verlauf

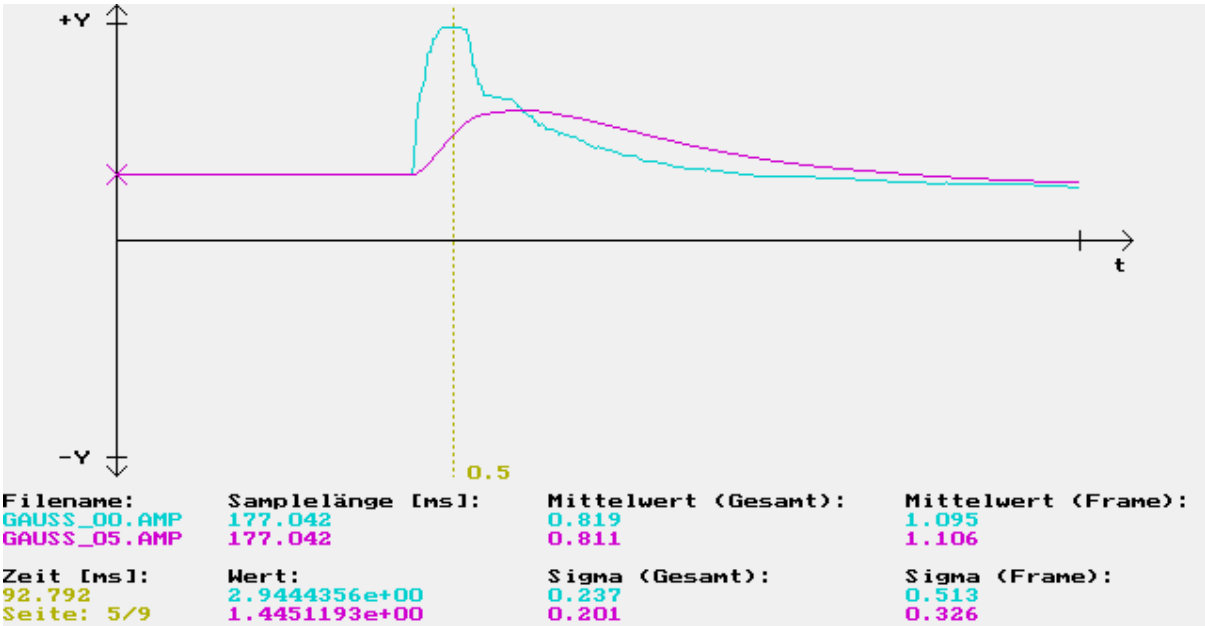

Abbildung 4.7: Verstärkungsverlauf τ_{gl} = 0 ms & τ_{gl} = 5 ms

5 Programmbeschreibung (SPV)

Zur Durchführung aller Messungen in dieser Arbeit wurde ein Programm in der Programmiersprache C entwickelt, das viele Standardfunktionen zur Signalbearbeitung und Signalanalyse, sowie alle für die Berechnungen in dieser Arbeit notwendigen Spezialfunktionen unter einer komfortablen Benutzeroberfläche integriert.

Das Programm besteht aus zwei Dateien:

- SPV.EXE Programmdatei.
- SPV.CFG Datei für die Programmparameter.

Die Datei SPV.CFG wird automatisch von SPV.EXE erzeugt und initialisiert wenn sie beim Programmstart nicht gefunden wird.

Zum Betrieb des Programms ist ein MS-DOS kompatibler PC mit Festplatte und einer EGA/VGA-Grafikkarte erforderlich.

5.1 Start des Programms und Kommandozeilenoptionen

Die Kommandozeile zum Start des Programms kann mehrere Parameter enthalten. Wenn als Parameter **?** oder eine ungültige Option angegeben wird, so wird eine Übersicht über die möglichen Optionen gegeben. Die Eingabe von SPV **?** führt zur Ausgabe von:

SPV V1.6 - Autor: Harald Zottmann - Datum: 1993

SPV [QCALC] [QDISP] [NOBEEP]

QCALC	**Nur Mittelwert und Sigma für**	**" Schätzwert**	**(SAM)**	**"**
		" Verstärkung	**(AMP)**	**"**
		" Ausgangssignal	**(S I G)**	**"**
QDISP	**Schnellmodus für**	**" Zeitfunktion zeigen**		**"**
		" Korrelation zeigen		**"**
		" Spektrum zeigen		**"**
NOBEEP	**Sound aus**			

Alle Parameter sind optional, d.h. sie können uneingeschränkt benutzt oder weggelassen werden. Zur Erklärung der Parameter ist es notwendig, sich über eine grundlegende Eigenart des Programms klar zu werden. Um möglichst wenig Arbeitsspeicher zu verbrauchen, wurde das Programm dateiorientiert aufgebaut. Alle Signaldateien werden auf der Festplatte zwischengespeichert, bevor sie zur Anzeige oder Weiterbearbeitung gebracht werden. Dieses Konzept ermöglicht eine sehr flexible Arbeitsweise, aber es führt auch zu einem Geschwindigkeitsverlust durch plattenintensives Arbeiten.

QCALC Bei den Funktionen Schätzwert, Verstärkung und Ausgangssignal wird keine Datei geöffnet um den Verlauf der Funktion abzuspeichern. Statt dessen werden nur Mittelwert und Standardabweichung der Signale berechnet. Diese Option beschleunigt drastisch die Ausgabe der relevanten Werte, wenn der genaue Verlauf der Funktion nicht gebraucht wird.

QDISP Bevor eine Datei angezeigt wird, werden vom Programm Mittelwert und Standardabweichung sowie die maximale Amplitude zur exakten Normierung ermittelt. Hierfür muß zuerst die gesamte Datei gelesen werden, was sehr zeitaufwendig ist. Mit der Option **QDISP** wird dieser Programmschritt übersprungen und nur noch der aktuelle Ausschnitt aus der Funktion normiert.

NOBEEP Schaltet das Tonsignal, das bei Fehlermeldungen und am Ende langer Operationen ertönt, ab.

5.2 Das Hauptmenü

Nach dem Start des Programms erscheint das Hauptmenü, aus dem auf alle Funktionen zugegriffen werden kann. Das Programm kann mit der ESC-Taste und einer anschließenden Sicherheitsabfrage verlassen werden.

Abbildung 5.1: SPV-Hauptmenü

Über die Tasten CURSOR UP und CURSOR DOWN kann die gewünschte Funktion angewählt und mit einem Druck auf die RETURN-Taste aktiviert werden.

5.3 Verzeichnis wechseln

Es erscheint ein Dateiauswahlfenster (Filerequester) über das alle Operationen zum Selektieren von Dateien (Files) und Verzeichnissen (Directories) ablaufen. Die Bedienung erfolgt mit den Tasten CURSOR UP und CURSOR DOWN sowie der TAB-Taste zum Wechseln zwischen Files und Directories. Mit der RETURN-Taste wählt man ein File an oder wechselt das Verzeichnis. Die ESC-Taste führt zum Verlassen des Requesters.

Abbildung 5.2: SPV-Filerequester

5.4 Parameter einstellen und SAM-Korrekturfaktor berechnen

Mit dieser Funktion gelangt man ins Parametermenü zur Einstellung aller globaler Variablen und zur Berechnung der Korrekturfaktoren Y_{cg}, Y_{cl} und Y_{cn}.

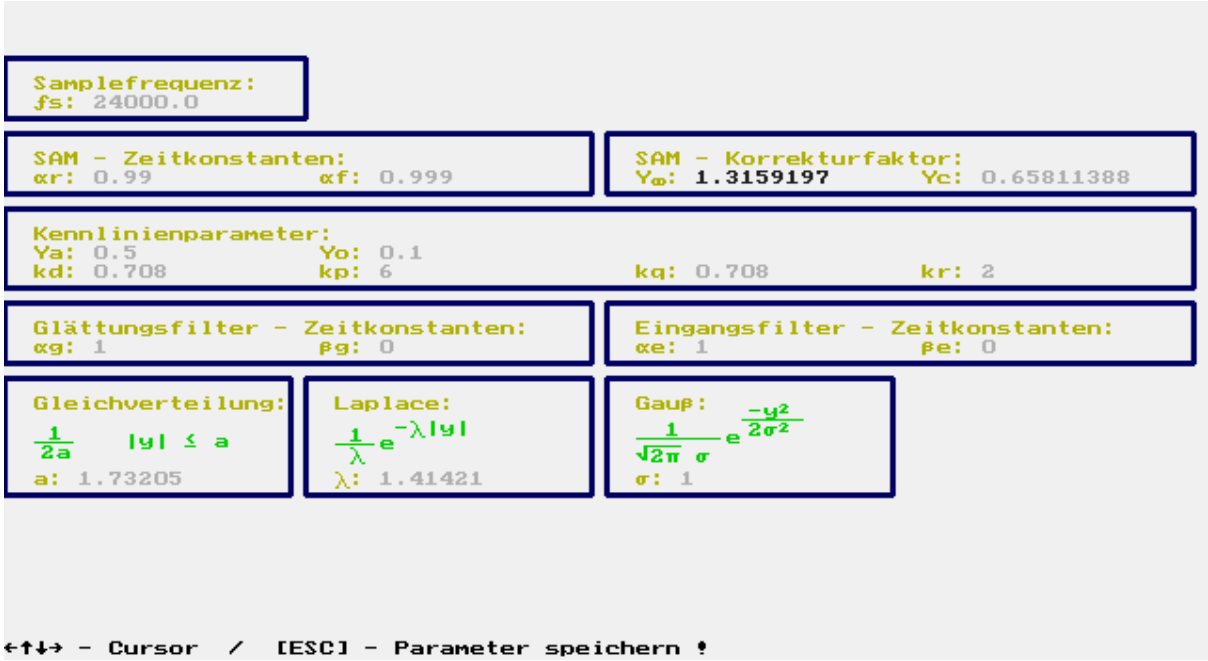

Abbildung 5.3: Parametermenü

Wenn bei der Eingabe eines neuen Parameters der zulässige Wertebereich für diesen überschritten wird, so erscheint eine Fehlermeldung. Sicherheitshalber kann das Parametermenü erst dann verlassen werden, wenn der entsprechende Wert korrigiert wird.

Zur Berechnung des SAM-Korrekturfaktors Y_c, wird der Eingabecursor mit den Cursortasten auf das entsprechende Feld gestellt und die dort angezeigte Zahl mit DELETE oder BACKSPACE gelöscht. Die Returntaste öffnet ein weiteres Menü, in dem die gewünschte Dichtefunktion ausgewählt werden kann. Der Berechnung liegen die gerade eingestellten SAM-Zeitkonstanten sowie der Wert, der die gewählte Dichtefunktion beschreibt (Bsp.: *a* für Gleichverteilung), zugrunde.

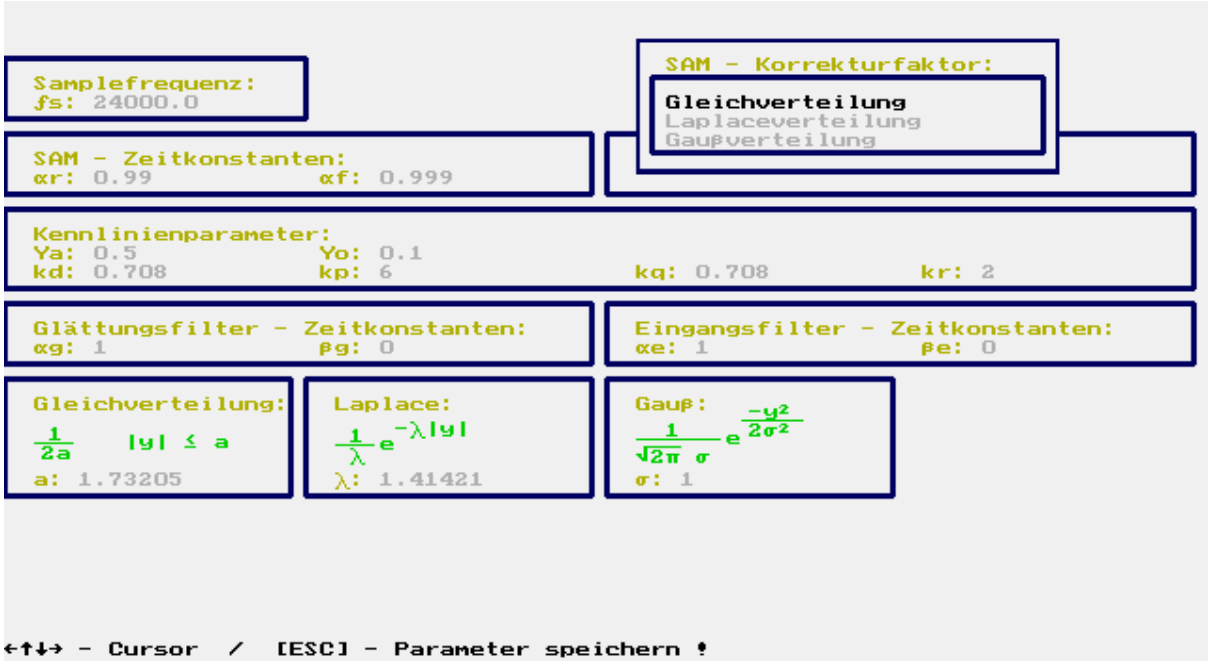

Abbildung 5.4: Parametermenü mit Auswahlmenü für die Dichtefunktion

Die anderen Parameter in diesem Menü werden an der Stelle der Anleitung beschrieben, an der sie relevant werden. Das Menü kann über die ESC-Taste verlassen werden, was gleichzeitig zu einer Sicherung der Parameter in die Konfigurationsdatei SPV.CFG führt. Die Programmparameter sind so bei einem Neustart von SPV immer aktuell.

5.5 Eingang erzeugen

Diese Funktion dient dem Erstellen von Musterfolgen mit bestimmten Dichtefunktionen.

- Gleichverteilung (FIL): Es wird eine gleichverteilte Zufallsfolge, wie in Kapitel 3.1.1 beschrieben, erzeugt. Der Parameter *a* wird im Parametermenü eingestellt. Die Folge wird durch das Eingangsfilter tiefpaß gefiltert und anschließend mit dem schon erwähnten Leistungs korrekturfaktor multipliziert. Durch das Filter geht die Gleich

verteilung, abhängig von der gewählten Zeitkonstanten, immer mehr in eine Normalverteilung über. Die Parameter für das Eingangsfilter werden ebenfalls im Parametermenü eingestellt

- Gleichverteilung: Wie vor, jedoch ohne Filter.

- Laplaceverteilung (FIL): Es wird eine laplaceverteilte Zufallsfolge, wie in Kapitel 3.1.2 beschrieben, erzeugt. Der Parameter λ wird wiederum im Parametermenü eingestellt. Auch diese Folge wird gefiltert, und es gilt für die Filterung das gleiche wie bei der Gleichverteilung mit Filter.

- Laplaceverteilung: Wie vor, jedoch ohne Filter.

- Gaußverteilung (FIL): Es wird eine normalverteilte Zufallsfolge, wie in Kapitel 3.1.3 beschrieben, erzeugt. Die Standardabweichung σ wird im Parametermenü eingestellt. Die Folge wird gefiltert, verliert dadurch aber nicht ihre Dichtefunktion.

- Gaußverteilung: Wie vor, jedoch ohne Filter.

- Gaußverteilung (SIRP): Es wird eine normalverteilte Zufallsfolge mit Hilfe von Include dateien zur Sirpgenerierung [5] erzeugt.

- K_0-Verteilung (SIRP): Es wird eine K_0-verteilte Zufallsfolge mit Hilfe von Include-dateien zur Sirpgenerierung [5] erzeugt.

Nach der Auswahl der gewünschten Verteilung fordert das Programm zur Eingabe der Dateilänge (Anzahl der Werte) auf. Schließlich muß noch ein Name für die zu erzeugende Datei angegeben oder der Vorschlag des Programms bestätigt werden. Das Programm startet dann die Erzeugung der Datei und informiert dabei den Benutzer mit einer Prozentanzeige über das Fortschreiten der Aktion. Sobald die Datei geschrieben ist, wird sie an die Histogrammfunktion weitergereicht. Das nun erscheinende Auswahlmenü zur Histogrammfunktion kann, wie alle anderen Menüs (Requester) des Programms auch, mit der ESC-Taste abgebrochen werden. Die Funktion "Histogramm zeigen" wird auf der folgenden Seite expliziert.

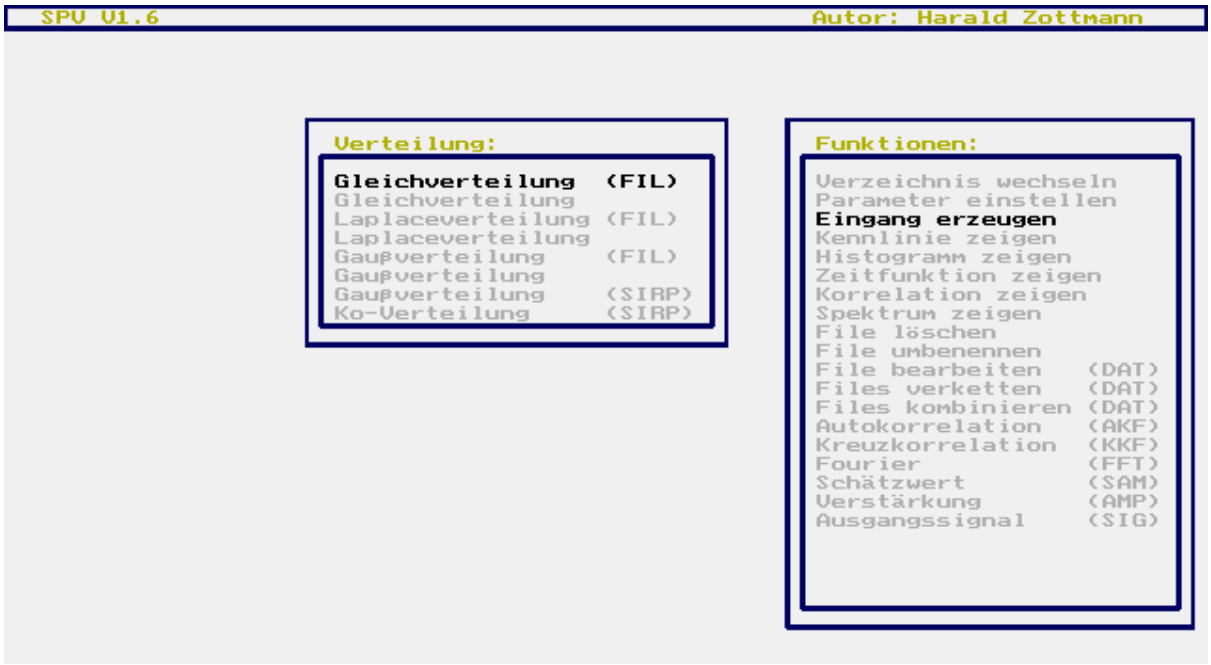

Abbildung 5.5: SPV-Eingang erzeugen

5.6 Kennlinie der Verstärkungssteuerung zeigen

Diese Funktion zeichnet die Verstärkungskennlinie mit den im Parametermenü eingestellten Kennlinienparametern (siehe Abbildung 4.1). Die Darstellung erfolgt doppeltlogarithmisch. Zur Weiterverwendung läßt sich die Grafik in ein codiertes PCX-Grafikfile mit 16 Farben sichern (Druck auf Taste G). Die Grafikdateien werden mit den Dateinamen SPVPIC0.PCX bis SPVPIC99.PCX durchnumeriert und im Verzeichnis von SPV abgelegt.

5.7 Histogramm zeigen

Die Histogrammfunktion erlaubt die lineare oder logarithmische Darstellung der Häufigkeit der Werte eines Datenfiles. Bei ihrer Aktivierung erscheint das bereits bekannte Dateiauswahlfenster zur Selektion eines Datenfiles. Der folgende Requester erfordert die Angabe: "lineare oder logarithmische Darstellung". Die Auflösung des Histogramms bestimmt die Anzahl der Unterteilungen des noch zu wählenden Wertebereichs. Danach kann man eine Vergleichsfunktion anwählen, welche die entsprechenden Parameter aus dem Parametermenü verwendet. Der Wertebereich des Histogramms kann entweder von Hand eingestellt oder durch eine Leereingabe vom Programm so bestimmt werden, daß eine eventuell gewählte Vergleichsfunktion automatisch optimal skaliert wird. Das Programm zeichnet im Anschluß das Koordinatensystem mit der Vergleichsfunktion und beginnt die Histogrammwerte einzulesen. Sind alle Werte gelesen, wird das Histogramm mit den Daten über Mittelwert und Standardabweichung zur Anzeige gebracht. Die Taste G bewirkt auch hier das Sichern des Grafikbildschirms in eine PCX-Datei. Über die RETURN-Taste wechselt das Programm den Bildschirm und es kann eine weitere Datei angewählt werden. Die gleichzeitige Darstellung von max. 15 Histogrammen ist möglich. Wertebereich und Darstellungsform sind nicht mehr neu einstellbar.

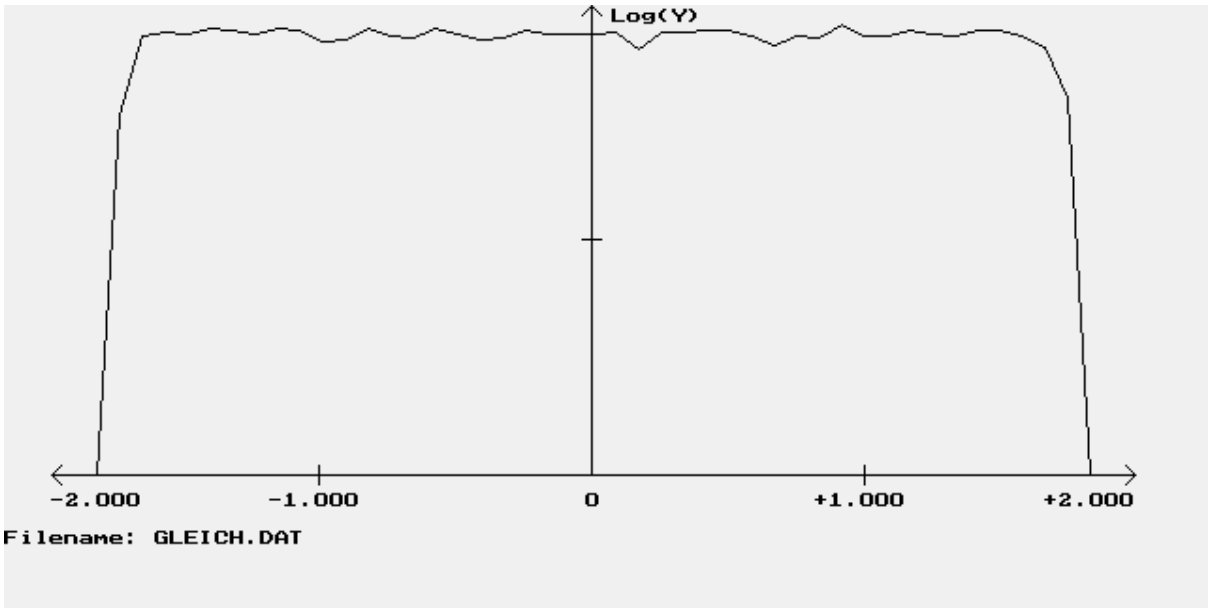

Abbildung 5.6: Logarithmische Darstellung einer Gleichverteilung

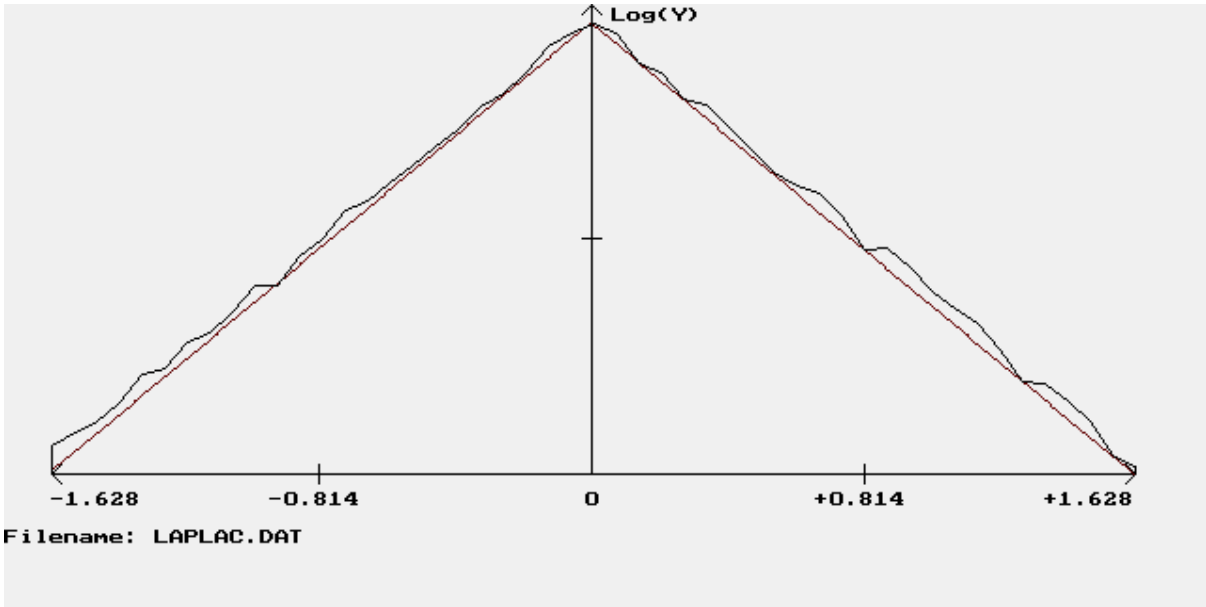

Abbildung 5.7: Logarithmische Darstellung einer Laplaceverteilung mit Vergleichsfunktion

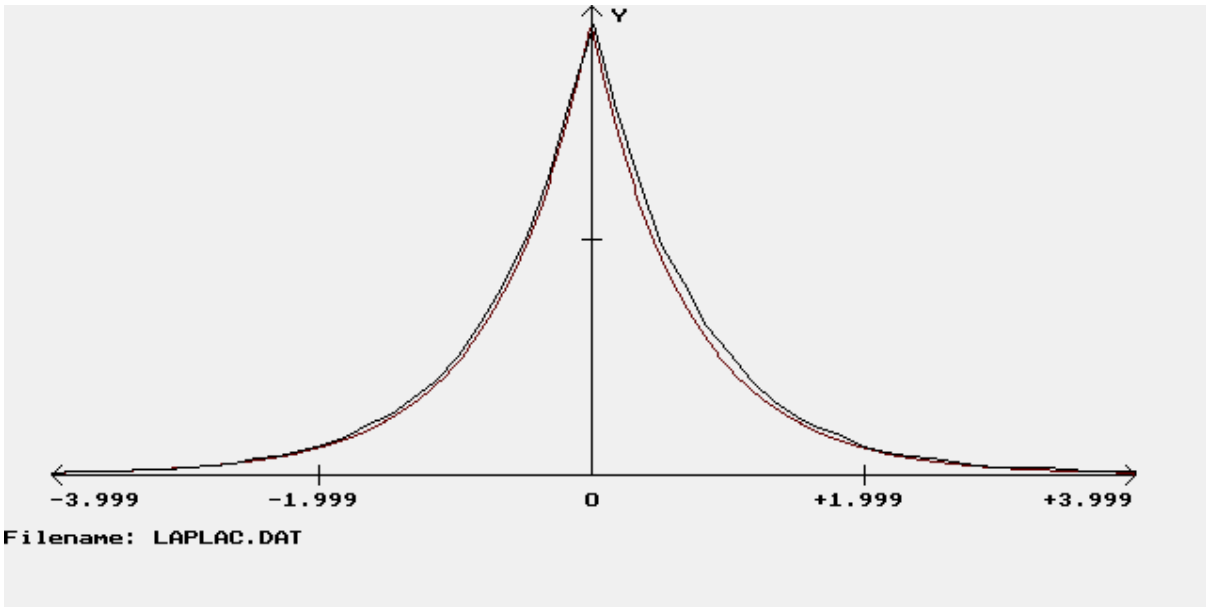

Abbildung 5.8: Lineare Darstellung einer Laplaceverteilung mit Vergleichsfunktion

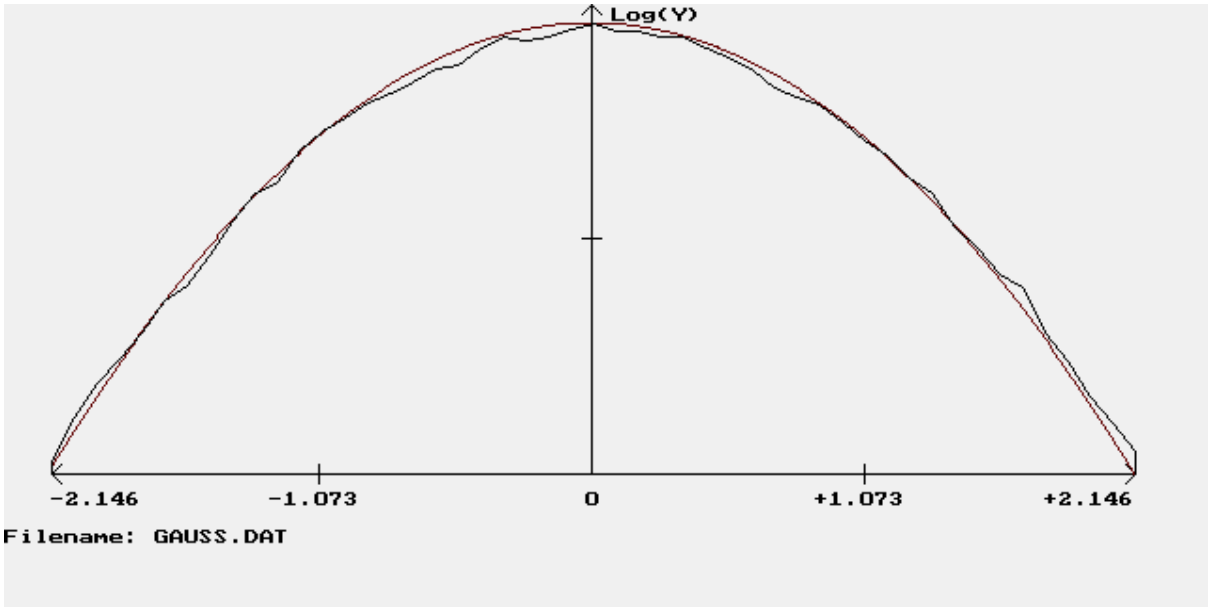

Abbildung 5.9: Logarithmische Darstellung einer Gaußverteilung mit Vergleichsfunktion

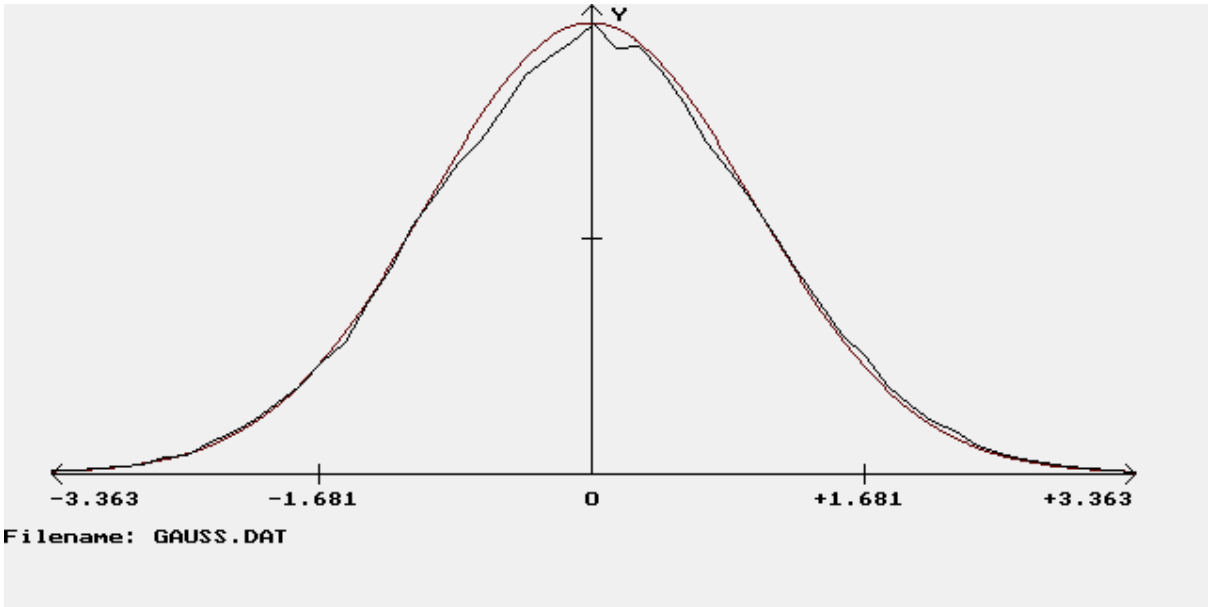

Abbildung 5.10: Lineare Darstellung einer Gaußverteilung mit Vergleichsfunktion

Die Histogramme aus Abbildung 5.6 bis 5.10 zeigen Dateien die mit SPV erstellt wurden. Jede Datei umfaßt 65536 Werte und wurde bei einer mittleren Auflösung von SPV automatisch skaliert.

5.8 Zeitfunktion zeigen

Diese Funktion dient der gleichzeitigen grafischen Anzeige von ein bis zwei Zeitsignalen. Über das Dateiauswahlfenster werden die Dateien selektiert. Die Framelänge gibt die Anzahl der Werte an, die auf einer Bildschirmseite untergebracht werden sollen. Nach einer kurzen Wartezeit erscheint die Datei auf dem Bildschirm. Über die Tasten CURSOR LEFT und CURSOR RIGHT kann der gelbe Markierungsstrich auf dem Bildschirm um jeweils einen Wert vor oder zurück verschoben werden. Dabei können der Zeitpunkt in der Datei, an dem sich der Strich momentan befindet, und die aktuellen Werte des Signals zu diesem Zeitpunkt abgelesen werden. Wenn zwei Dateien dargestellt werden, so befindet sich unten rechts an dem Markierungsstrich noch eine Faktorangabe, die beim Vergleich zwischen einer Eingangsdatei und der verstärkten Ausgangsdatei sehr hilfreich ist, um den jeweiligen Verstärkungsfaktor sofort ablesen zu können. Die Tasten CURSOR UP und CURSOR DOWN verschieben den Markierungsstrich um mehrere Werte. Damit ist ein schnelles Erreichen einer weiter entfernten Bildschirmposition möglich. Mit PAGE UP und PAGE DOWN kann eine Bildschirmseite vor bzw. zurückgeblättert werden. Die Anzeige Seite x/y gibt an, auf welcher Seite man sich gerade befindet. Mit POS1 erreicht man den Anfang der Datei und mit END das Ende. Zusätzlich werden am unteren Bildschirmrand noch einige statistische Daten über die angezeigten Datenfiles eingeblendet. Die Anzeige kann mit der ESC-Taste verlassen oder mit der G-Taste abgespeichert werden.

5.9 Korrelation zeigen

Diese Funktion stellt lediglich einen anderen Modus der Funktion "Zeitfunktion zeigen" dar. Die Bedienung erfolgt hier analog, jedoch mit dem Unterschied, daß nur AKF- und KKF-Files angewählt werden dürfen. Der Verschiebungszeitpunkt 0 ms der Korrelation wird hier noch mit einer senkrechten Linie markiert. Wurden zwei Korrelationen selektiert, werden ihre zeitlichen Nullpunkte in der grafischen Darstellung zusammengelegt.

5.10 Spektrum zeigen

Diese Funktion dient der logarithmischen Anzeige von FFT-Dateien. Es ist zu entscheiden zwischen rahmenweiser Ansicht eines Spektrums und der Mittelung aller verfügbarer Rahmen. Wenn zwei Spektren mit unterschiedlicher Rahmenlänge angewählt sind, wird automatisch gemittelt und auf die Rahmenlänge des kürzeren Rahmens normiert. Die Frequenzanzeige basiert auf der im Parametermenü eingestellten Samplefrequenz.

5.11 File löschen

Diese Funktion wurde implementiert, um nicht mehr benötigte Dateien von der Festplatte löschen zu können, ohne SPV zwischenzeitlich beenden zu müssen. Bei ihrem Aufruf erscheint der Filerequester, um die zu löschende Datei auszuwählen. Eine versehentliche Aktivierung dieser Funktion kann mit der ESC-Taste abgebrochen werden. Eine letzte Sicherheitsabfrage vor dem Löschvorgang schützt zusätzlich vor dem Verlust wichtiger Dateien.

5.12 File umbenennen

Diese Funktion gestattet das Umbenennen von beliebigen Dateien. Über den Filerequester wird die Datei, die einen neuen Namen bekommen soll, angewählt. Daraufhin kann der neue Name eingegeben werden. Existiert der neue Filename schon oder entspricht er nicht den DOS-Regeln, erscheint eine Fehlermeldung und der alte Filename bleibt erhalten.

Abbildung 5.11: Warnung

5.13 File bearbeiten

Unter diesem Menüpunkt sind alle Funktionen zusammengefaßt, die eine Bearbeitung der Datenfiles ermöglichen. Nach der Auswahl des Files, auf das sich die Operation beziehen soll, kann man wählen zwischen:

- Mittelwert einstellen: Dient der Addition oder Subtraktion eines noch anzugebenden Mittelwerts. Erfolgt eine Leereingabe (Zahl löschen und RETURN-Taste drücken), so wird die Datei automatisch mittelwertfrei.
- Sigma einstellen: Wählen einer beliebigen Standardabweichung für das Datenfile. Negative Werte werden ignoriert.
- Amplitude normieren: Das Datenfile wird so umnormiert, daß sein Betragsmaximum (Peak) dem eingestellten Wert entspricht.
- Multiplizieren: Multiplikation mit einem festen Wert. Der Wert kann auch negativ sein. Die Eingabe von -1 hätte zum Beispiel eine Phasendrehung des Signals um 180° zur Folge
- Filter: Die Datei wird durch das Eingangsfilter gefiltert.
- Filter (Sigma konstant): Wie vor, jedoch zusätzliche Multiplikation mit dem Leistungs korrekturfaktor (Anhang A).

Zum Start der Operation muß noch ein Name für die Zieldatei eingegeben werden. Wird der Namensvorschlag akzeptiert, so wird die alte Datei mit den neuen Werten überschrieben.

5.14 Files verketten

Mit Hilfe dieser Funktion ist man in der Lage beliebige Teile zweier Datenfiles (DAT, SAM, AMP oder SIG) zu einer neuen Datei aneinander anzufügen. Zuerst wählt man nacheinander die beiden Datenfiles aus. Es öffnet sich nun ein Requester (Abbildung 5.12) wie er noch an anderen Stellen des Programms (AKF, KKF & FFT) Verwendung findet.

Um die beiden Dateiteile flexibel und übersichtlich bestimmen zu können benötigt man die beiden Start- und Endpunkte in den Dateien. Diese Informationen können hier mittels der Cursortasten grafisch sichtbar eingegeben werden. Die TAB-Taste dient der Auswahl des Punktes, der verändert werden soll. Der angewählte Markierungsstrich wird dann weiß dargestellt, die drei anderen sind in Grau gehalten. Mit den Tasten CURSOR RIGHT und CURSOR LEFT läßt sich die Markierung um je einen Wert verschieben. Gleichzeitig wird man über die genaue Position mit einer Zeit- und einer Wertangabe informiert. Die Tasten CURSOR UP und CURSOR DOWN verschieben die Markierung um je 100 Schritte. Mit der RETURN-Taste bestätigt man die Eingabe, ESC führt zum Abbruch. Schließlich muß noch ein neuer

Dateiname eingegeben werden, wobei das Suffix vorgeschlagen wird.

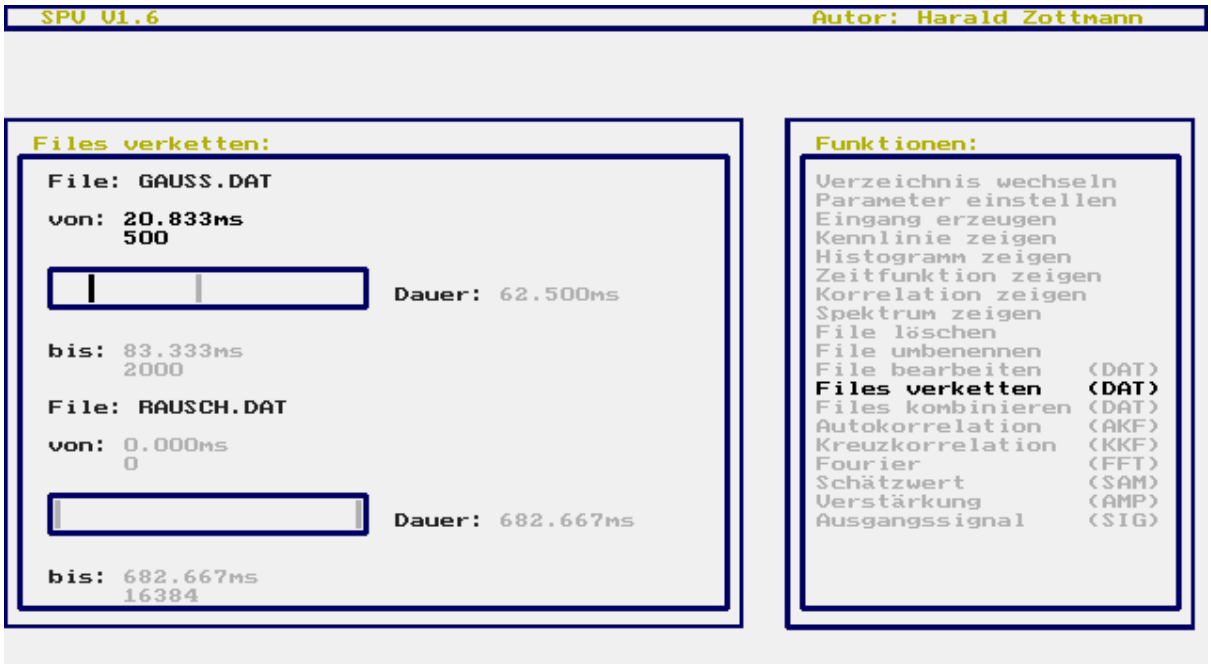

Abbildung 5.12: Files verketten

5.15 Files kombinieren

Diese Funktion dient dem multiplikativen oder additiven Kombinieren zweier Datenfiles. Sie ermöglicht so etwa die Belegung einer Nutzsignaldatei mit einer Störung oder die Erzeugung von K_0-verteilten Prozessen durch Multiplikation mehrerer Gaußprozesse [6].

Der Eingaberequester (Abbildung 5.13) erlaubt noch die Bestimmung von Faktoren mit denen die Ein- und Ausgangsdateien gewichtet werden sollen sowie des Dateinamens für die Ausgangsdatei. Weisen die beiden Eingangsdateien unterschiedliche Längen auf, erhält die Ausgangsdatei die Länge der kürzeren Eingangsdatei.

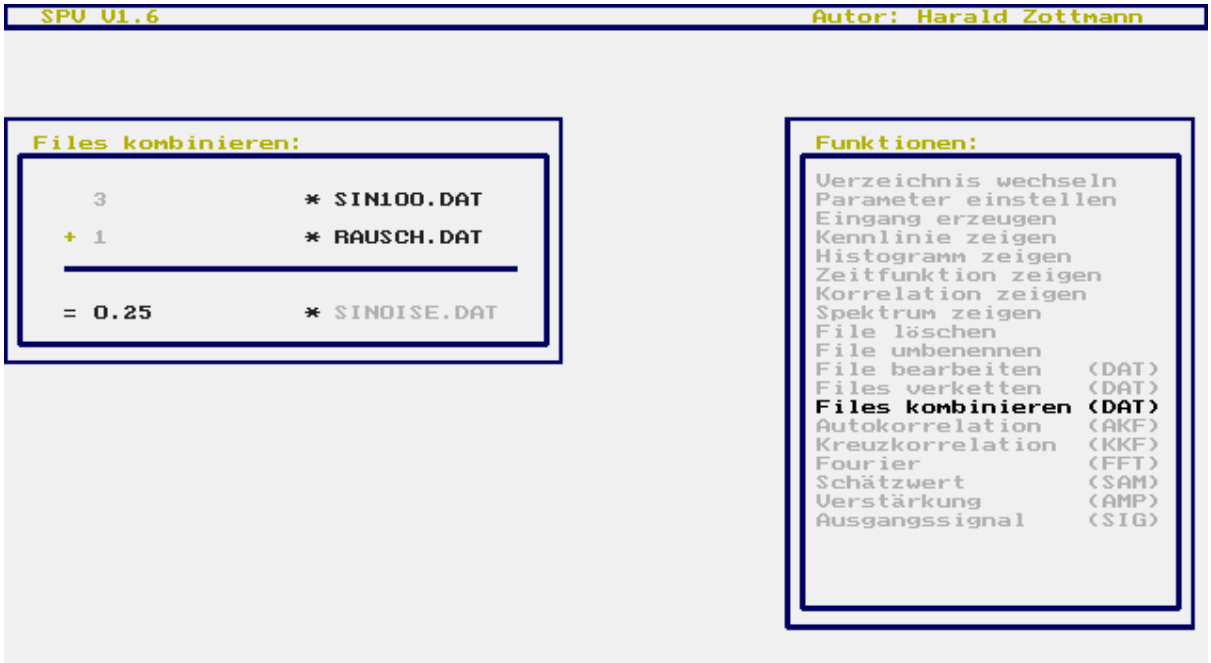

Abbildung 5.13: Files kombinieren

5.16 Autokorrelation (AKF)

Zur Bestimmung der AKF eines Datenfiles läßt sich ein beliebiger Ausschnitt der Funktion mit einem Teil der Datei korrelieren. Über den Requester in Abbildung 5.14 können alle nötigen Daten eingestellt werden. Das Anwendungsprinzip gleicht dem des Requesters aus Abbildung 5.12. Da die benötigten Dateiteile in den Arbeitsspeicher geladen werden ist ihre maximale Länge auf 32767 Werte beschränkt.

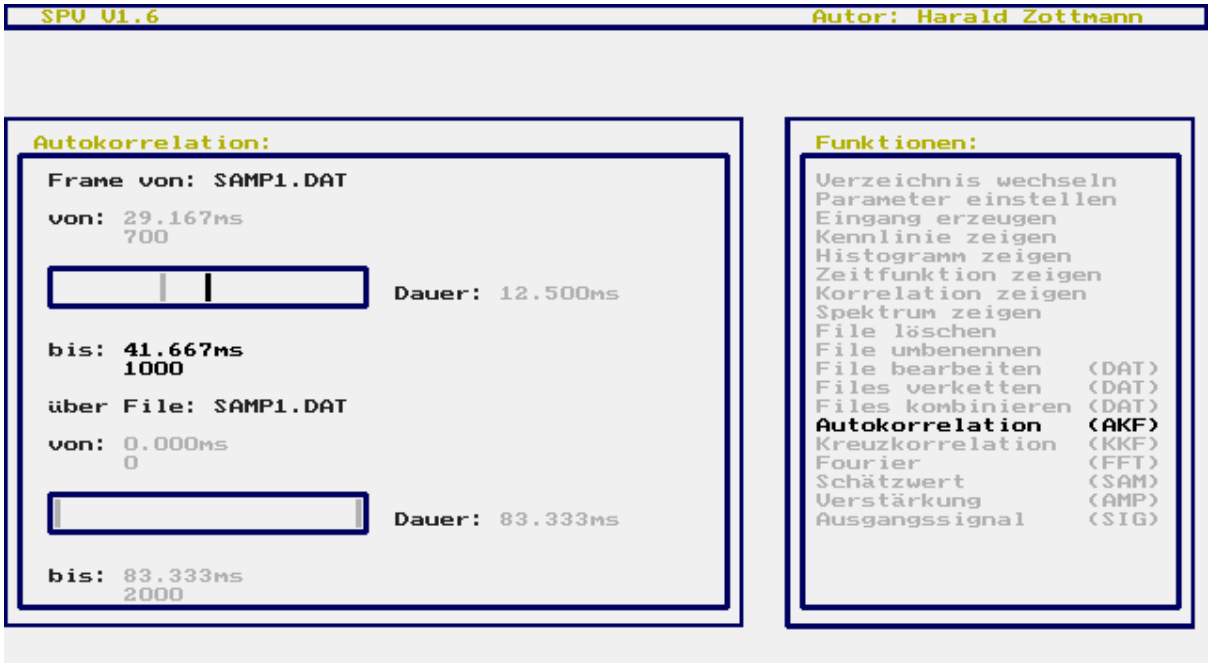

Abbildung 5.14: Autokorrelation

5.17 Kreuzkorrelation (KKF)

Diese Funktion gestattet die Berechnung von Kreuzkorrelationen (KKF) zwischen zwei Datenfiles. Auch hier lassen sich Start- und Endpunkte in beiden Dateien beliebig einstellen.

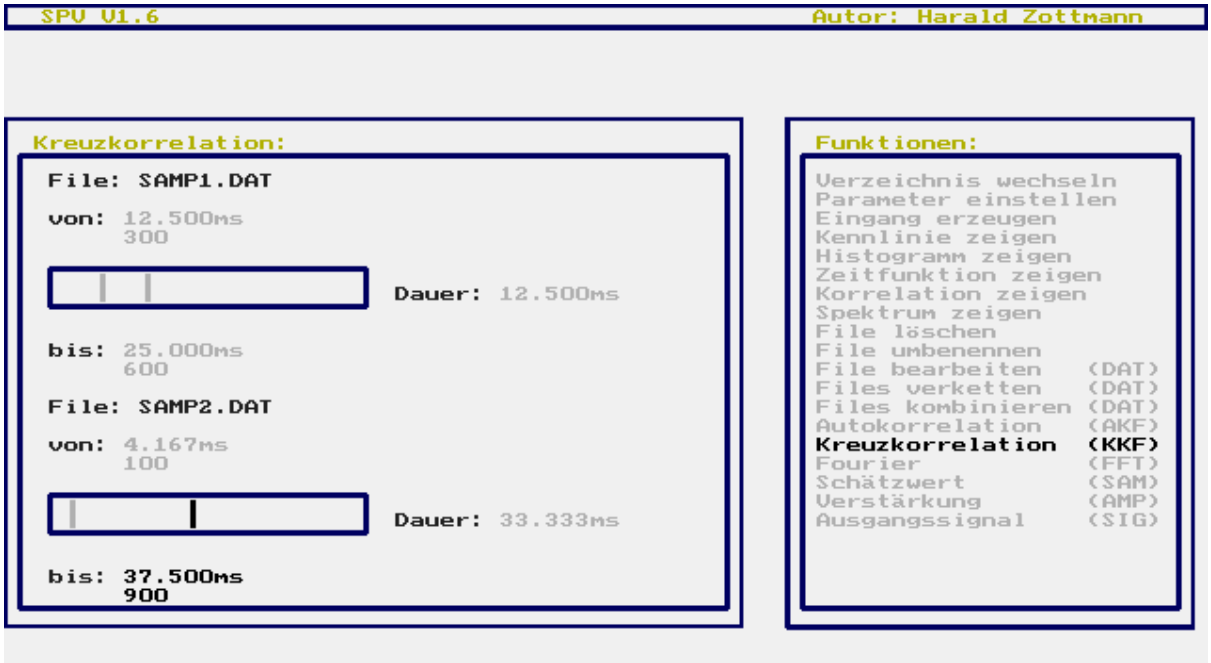

Abbildung 5.15: Kreuzkorrelation

5.18 Fouriertransformation (FFT)

Unter diesem Menüpunkt befindet sich eine Implementation des FFT-Algorithmus (Rechteckfenster) mit Look-Up-Table für die benötigten Sinus- und Cosinus-Werte. Zusätzlich zum Namen der zu transformierenden Datei, muß noch die Framelänge (512, 1024, 2048, 4096) für die Transformation angegeben werden. Das Programm ermittelt dann die Anzahl der resultierenden Frames und gestattet die Wahl, alle Frames oder nur einen bestimmten Teil zu berechnen.

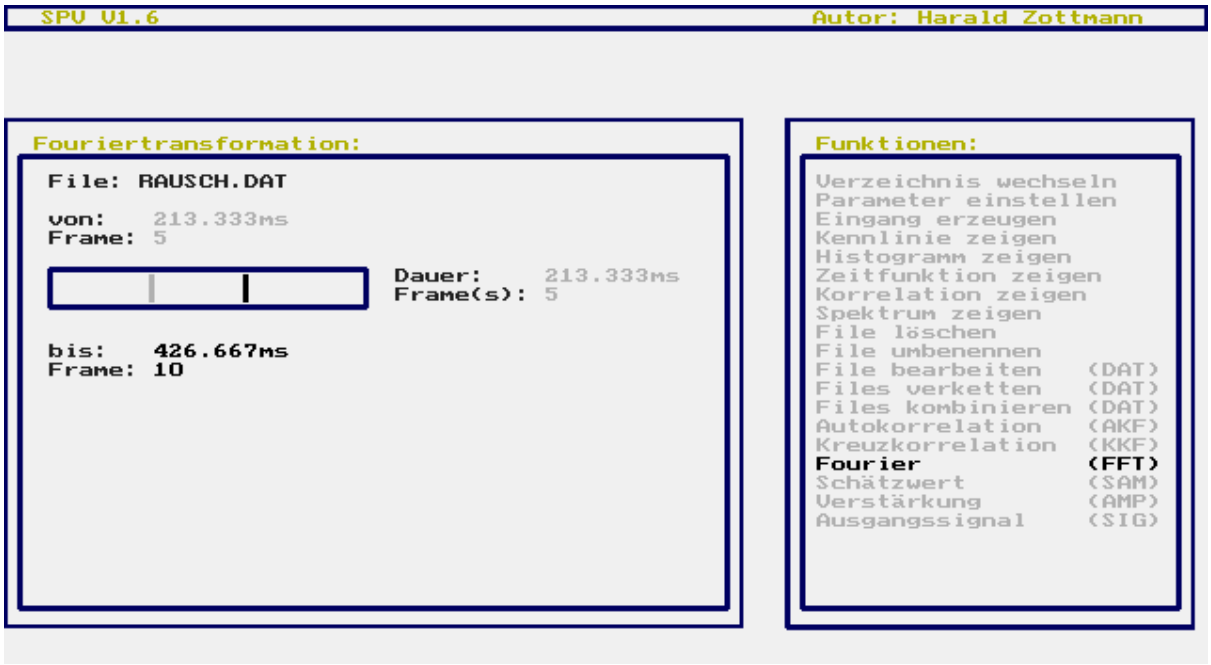

Abbildung 5.16: Fouriertransformation

Die im folgenden beschriebenen Funktionen dienen alle der Untersuchung des oberen Zweigs der Schaltung aus Abbildung 2.1.

Der Verlauf der einzelnen Ausgangssignale wird nur dann gespeichert, wenn der Parameter **QCALC** nicht in der Kommandozeile zum Starten von SPV angegeben war. Ansonsten ermitteln die Funktionen nur Mittelwert und Standardabweichung der Signale (siehe Kapitel 5.1).

5.19 Schätzwertverlauf (SAM)

Bestimmung des Schätzwertverlaufs aus einem Eingangssignal. Die SAM-Zeitkonstanten müssen im Parametermenü eingestellt werden.

- Direkt: Ausgang des SAM-Schätzers.

- Korrigiert: Ausgang des SAM-Schätzers und Multiplikation mit dem Korrekturfaktor Y_c aus dem Parametermenü.

5.20 Verstärkungsverlauf (AMP)

Bestimmung des Verlaufs der Verstärkung aus einem Eingangssignal (d.h. der Schätzwertverlauf muß nicht extra berechnet werden). Die Verstärkungskennlinie wird immer mit dem korrigierten SAM-Wert angesteuert. Um den direkten SAM-Wert zu nutzen muß im Parametermenü an der Stelle Y_c eine 1 eingetragen werden.

- Direkt: Verstärkungsverlauf nach der Kennlinie.
- Geglättet: Verstärkungsverlauf nach dem Glättungsfilter

5.21 Ausgangssignal (SIG)

Bestimmung des verstärkten Ausgangssignals aus einem Eingangssignal (d.h. wie schon beim Verstärkungsverlauf kann diese Funktion direkt auf eine Zeitsignaldatei angewendet werden). Um den Einfluß des Glättungsfilters auszuschalten müssen die Filterparameter im Parametermenü auf $\alpha_g = 1$ und $\beta_g = 0$ gesetzt werden. Zur Vereinfachung der Parametereingabe wird bei der Leereingabe eines Filterparameters dieser automatisch zu 1 minus dem korrespondierenden Filterparameter gesetzt.

Wenn bei der Abfrage des Ausgangsfilenamens eine Leereingabe erfolgt, so wird ein temporäres File SPV.TMP angelegt. Eine temporäre Datei wird, wenn möglich, auch von allen anderen Funktionen angelegt, wenn kein eigener Dateiname für die Ausgangsdatei eingegeben wird. Die Datei SPV.TMP kann nach ihrer Erstellung mit der ihr entsprechenden Funktion angezeigt, umbenannt oder gelöscht werden. Sie sollte jedoch sicherheitshalber nicht als Eingangsdatei für eine bearbeitende Funktion dienen, da der Dateiname SPV.TMP vom Programm intern noch zur Zwischenspeicherung anderer Daten genutzt wird.

6 Programmerweiterung (SPV)

Das Programm SPV wurde modular aufgebaut, um eine nachträgliche Erweiterung mit neuen Funktionen zu unterstützen. Dabei können alle Grundfunktionen des Programms wie Eingaberoutinen und Ausgaberoutinen auch von anderen Funktionen genutzt werden. Das wichtigste Format, das bei der Implementation neuer Programmteile beachtet werden muß ist das Fileformat. Davon kennt SPV drei verschiedene Arten:

- Zeitfunktionen: Die einzelnen Werte der Zeitfunktionen (DAT, SAM, AMP, SIG)

  Beispiel:

```
FILE  fp;
double wert;
.
.
fprintf (fp, "%16.7e", wert);
fputs ("\x0D\0A", fp);
.
.
```

- Korrelationen: Bei den AKF- und KKF-Dateien gilt das gleiche wie für die Zeitfunktionen. Einziger Unterschied ist der erste Eintrag im File, der die Position des Nullpunkts der Verschiebung markiert.

- Spektren: Bei FFT-Dateien steht im ersten Wert die Framelänge der Trans

Kapitel 6: Programmerweiterung (SPV)

<u>Um SPV mit einer neuen Funktion zu ergänzen, sind folgende Schritte notwendig:</u>

1.) Im Modul SPV.C muß der Name der neuen Funktion in die globale Variable **funktion[]* eingetragen werden.

Bsp.:

```
.
.
"Ausgangssignal     (SIG)",
"Neue Funktion"};
.
.
```

2.) Die Funktion *void select (int fkt)* muß um eine *case* Anweisung ergänzt werden.

Bsp.:

```
.
.
case 19:          /* Verstärkung * Signal */
.
.
break;

case 20:          /* Hier wird die neue Funktion aufgerufen */
neue_funktion ();
break;

default:
break;
.
.
```

Die Variable *fkt* wird automatisch von SPV gesetzt und beinhaltet die Nummer der gewählten Funktion.

3.) In der Includedatei SPVINC.H muß die Zahl hinter *#define FUNKANZ* um 1 erhöht werden. Weiterhin muß ein Prototyp der Funktion in dieser Datei deklariert werden.

Bsp.:

```
.
.
void     neue_funktion ();
.
.
```

4.) Neue Funktionen, welche die Includedateien zur Sirpgenerierung [5] benötigen, müssen ins Modul SPVSIRP.C eingebaut werden. Alle anderen Funktionen sollten im Modul SPVFUNC.C ergänzt werden.

5.) Zum Compilieren und Linken des Programms wird das Projektfile SPV.PRJ sowie die Objektdateien EGAVGA.OBJ und GRAPHICS.OBJ benötigt. SPV wurde auf dem Turbo C++ System V1.0 von Borland entwickelt.

7 Programmbedienung (SPVGEN)

Bei SPVGEN, handelt es sich um ein kommandozeilengesteuertes Tool zur Erzeugung von Musterfolgen. Diese Folgen können als Eingang für SPV dienen. Unter Ausnutzung der zur Verfügung stehenden Funktionen von SPV und SPVGEN lassen sich viele verschiedene Musterfolgen generieren. SPVGEN wurde so aufgebaut, daß es leicht um weitere Funktionstypen ergänzt werden kann. Die Eingabe von SPVGEN **?** gibt folgende Optionsliste aus:

SPVGEN V1.0 - Autor: Harald Zottmann - Datum: 1993

SPVGEN	**[SIN|COS]**	**[SGFREQ:]**	**[SPFREQ:]**	**[OFFSET:]**	**[LENGTH:]**	**[NAME:]**
	[SI]	**[SGFREQ:]**	**[SPFREQ:]**	**[OFFSET:]**	**[LENGTH:]**	**[NAME:]**
	[S01|S10]	**[T1: T2:]**		**[OFFSET:]**		**[NAME:]**
	[RAMP]	**[T1: T2:]**		**[OFFSET:]**	**[LENGTH:]**	**[NAME:]**
	[RECT]	**[T1: T2:]**		**[OFFSET:]**	**[LENGTH:]**	**[NAME:]**
	[E1|E2]	**[ALPHA:]**		**[OFFSET:]**	**[LENGTH:]**	**[NAME:]**

SIN|COS	**Sinusfunktion oder Cosinusfunktion**
SI	**SI-Funktion**
S01|S10	**Sprungfunktion 0 -> 1 oder 1 -> 0**
RAMP	**Rampenfunktion**
RECT	**Rechteckfolge**
E1|E2	**Normale oder quadratische Exponentialfunktion**

SGFREQ	**Signalfrequenz**
SPFREQ	**Samplefrequenz**
ALPHA	**Dämpfungsfaktor**
T1/T2	**Tastverhältnis**
OFFSET	**Summand für alle Werte**
LENGTH	**Anzahl der Werte in der Datei**
NAME	**Dateiname**

Parametername:	Defaultparameter:	Datentyp:
SGFREQ	100 Hz	Ganze Zahlen: 0-65536
SPFREQ	24000 Hz	Ganze Zahlen: 0-65536
ALPHA	0.01	Fließkommazahlen
T1/T2	10	Ganze Zahlen: 0-32767
OFFSET	0.0	Fließkommazahlen
LENGTH	8192	Ganze Zahlen: 1-65536
NAME	SPVGEN.DAT	Zeichenkette

Tabelle 7.1: Parameter von SPVGEN

Zum besseren Verständnis wird SPVGEN nun an Hand einiger Beispiele beschrieben:

SPVGEN SIN SGFREQ:500 SPFREQ:32000 LENGTH:4096 NAME:SIN500.DAT

Mittelwertfreies Sinussignal mit einer Frequenz von 500 Hz bezüglich einer Samplefrequenz von 32 kHz. Die Datei trägt den Namen SIN500.DAT und umfaßt 4096 Werte.

SPVGEN SI SGFREQ:200 LENGTH:1024

SI-Funktion mit einer Frequenz von 200 Hz bezüglich einer Samplefrequenz von 24 kHz. Die Datei trägt den Namen SPVGEN.DAT und umfaßt 1024 Werte. Das Maximum der SI-Funktion liegt in der Mitte der Datei. Die SI-Funktion ist wie folgt definiert:

$$\mathrm{si}(2\pi\,\mathrm{kT}) = \frac{\sin(2\pi\,\mathrm{kT})}{2\pi\,\mathrm{kT}}$$

SPVGEN S10 T1:100 T2:412 NAME:S10.DAT

Sprungfunktion bei der die ersten 100 Werte 1 und die nächsten 412 Werte 0 sind. Die Datei trägt den Namen S10.DAT und umfaßt 100+412=512 Werte.

SPVGEN S01 T1:512 T2:512 OFFSET:-0.5

Sprungfunktion bei der die ersten 512 Werte -0.5 und die nächsten 512 Werte 0.5 sind. Die Datei trägt den Namen SPVGEN.DAT und umfaßt 1024 Werte.

SPVGEN RAMP T1:100 T2:40 NAME:RAMP.DAT

Rampenfunktion mit einer Amplitude von -1 bis +1. Der Anstieg von -1 bis +1 dauert 100 Werte und der Abfall von +1 bis -1 dauert 40 Werte. Die Datei trägt den Namen RAMP.DAT und umfaßt 8192 Werte.

SPVGEN RAMP T1:50 T2:1 OFFSET:1.0 LENGTH:512

Sägezahnfunktion mit einer Amplitude von 2. Der Anstieg von 0 bis 2 dauert 50 Werte. Die Datei trägt den Namen SPVGEN.DAT und umfaßt 512 Werte.

SPVGEN RECT T1:1 OFFSET:1.0 LENGTH:1024 NAME:DIRAC.DAT

Folge von Diracimpulsen der Höhe 2 im Abstand von 10 Werten. Die Datei trägt den Namen DIRAC.DAT und umfaßt 1024 Werte.

SPVGEN RECT T1:20 T2:20

Rechteckfolge mit je 20 Werten -1 und 20 Werten +1. Die Datei trägt den Namen SPVGEN.DAT und umfaßt 8192 Werte.

SPVGEN E1 LENGTH:1024

Exponentialfunktion der Form: $e^{-\alpha|k|}$ *mit* $\alpha = 0.01$

Die Datei trägt den Namen SPVGEN.DAT und umfaßt 1024 Werte. Das Maximum und damit der zeitliche Nullpunkt der Funktion, liegt in der Mitte der Datei.

SPVGEN E2 ALPHA:0.002 LENGTH:2048 NAME:EXP2.DAT

Exponentialfunktion der Form: $e^{-\alpha k^2}$ *mit* $\alpha = 0.002$

Die Datei trägt den Namen EXP2.DAT und umfaßt 2048 Werte. Das Maximum der Funktion liegt in der Mitte der Datei.

SPVGEN gibt während der Erzeugung einer Musterfolge mit einer Prozentanzeige Auskunft über den Stand der Aktion. Die neue Datei kann danach sofort von SPV geladen und - wie in Kapitel 5 beschrieben - weiterbearbeitet werden.

Bei der Programmierung von SPV wurden von SPVGEN erzeugte Dateien eingesetzt, um einige Funktionen von SPV wie AKF, KKF, FFT usw. zu testen. Auch die 100 Hz-Sinusfunktion, die zur Ermittlung des Klirrfaktors in Kapitel 4.1 & 4.2 benötigt wurde, kann von SPVGEN erstellt werden.

8 Zusammenfassung und Ergebnis

In der vorliegenden Arbeit wurde ein Verfahren zur variablen Verstärkungssteuerung für Freisprechtelefone vorgestellt.

Der hierbei verwendete SAM-Schätzer, zur Schätzung des Kurzzeitmittelwerts des Betrags der Abtastwerte, mit unterschiedlichen Zeitkonstanten für steigende und fallende Eingangsleistung, wurde analysiert. Ziel war einen Zusammenhang zwischen dem mittleren SAM-Endwert und den SAM-Parametern zu finden. Bei der Analyse stellte sich heraus, daß man einen von der Eingangsleistung unabhängigen Faktor angeben kann, der es ermöglicht die SAM-Schätzwerte mit denen eines anderen Schätzers, mit nur einer Zeitkonstanten, zu vergleichen. Zur Bestimmung dieses Faktors Y_c, muß man jedoch die Dichtefunktion des Eingangssignals kennen. In dieser Arbeit wurden die Gleichungen zur Bestimmung von Y_c für Gleich- (Y_{cg}), Laplace- (Y_{cl}) und Normalverteilungen (Y_{cn}) ermittelt.

Der Analyse des SAM-Schätzers liegt die Annahme zugrunde, daß aufeinanderfolgende Werte des Eingangssignals statistisch unabhängig sind. Da realistische Signale diese Forderung nicht erfüllen, wurde am Beispiel einiger normalverteilter realistischer Signale überprüft, ob die theoretischen Ergebnisse mit den durch Versuchsmessungen ermittelten Ergebnissen übereinstimmen. Dabei wurde festgestellt, daß die Theorie um weniger als 4% von den im Versuch gemessenen Werten abweicht, und somit gute Ergebnisse für die Praxis liefert.

Wie bekannt, erfordert die Bestimmung von Y_c die Dichtefunktion des Eingangssignals. Da Sprachsignale eine andere Dichte (≈Laplace) als Rauschsignale (≈Gauß) aufweisen, muß entschieden werden, für welche Eingangssignale Y_c einzustellen ist. Um den Fehler für beide Fälle möglichst gering zu halten, bietet sich die Bestimmung eines Zwischenwerts von Y_{cl} und Y_{cn} an. Eine andere, wenn gleich aufwendigere Lösung könnte darin bestehen, die Dichtefunktion des Eingangssignals in regelmäßigen Intervallen zu schätzen und mit Hilfe dieser Information den passenden Faktor Y_c auszuwählen. Die Schätzung der Dichtefunktion kann mittels einer einfachen Histogrammberechnung erfolgen. Schon mit wenigen Abtastwerten (≈2000) läßt sich eine relativ zuverlässige Aussage über die Verteilung des Eingangssignals machen.

Literaturverzeichnis

[1] Heitkämper, Peter:
Erweiterung des SEL-Freisprechverfahrens (Abschlußbericht des zweiten Jahres der Zusammenarbeit zwischen THD und Alcatel-SEL Stuttgart, Januar 1993)

[2] Papoulis, Athanasios:
Probability & Statistics
Prentice-Hall International Editions 1990

[3] Hänsler, Eberhard:
Grundlagen der Theorie statistischer Signale
Springer-Verlag Berlin, Heidelberg 1983

[4] Borland Turbo C++ V1.0
Referenzhandbuch

[5] Rupp, Markus & Riedel, Sven:
Dokumentation zur Software (Sirpgenerierung)
Institut für Netzwerk- und Signaltheorie
Technische Hochschule Darmstadt, 1992

[6] Brehm, Helmut & Jüngst, Ernst-Werner & Wolf, Dietrich:
Simulation von Sprachsignalen
Archiv für Elektronik und Übertragungstechnik (AEÜ)
Band 28, Heft 11, November 1974

Anhang A

Herleitung des Leistungskorrekturfaktors

$$Y = \alpha X + \beta Y e^{j\Omega}$$

$$G(e^{j\Omega}) = \frac{\alpha}{1 - \beta e^{-j\Omega}}$$

$$\frac{S_{yy}(\Omega)}{S_{xx}(\Omega)} = G(e^{j\Omega})G(e^{-j\Omega}) = \frac{\alpha^2}{1 + \beta^2 - 2\beta\cos(\Omega)}$$

$$\int_{-\pi}^{+\pi} \frac{S_{yy}(\Omega)}{S_{xx}(\Omega)} d\Omega = \alpha^2 \int_{-\pi}^{+\pi} \frac{d\Omega}{(1 + \beta^2) + (-2\beta\cos(\Omega))}$$

$$Es\ gilt: (1 + \beta^2)^2 > (-2\beta)^2 \quad fÅr \quad 0 < \beta < 1$$

$$-\frac{2\alpha^2}{\sqrt{((1 + \beta^2)^2 - 4\beta^2)}} \arctan\left(\frac{(1 + \beta^2 + 2\beta)\tan(\frac{\Omega}{2})}{\sqrt{((1 + \beta^2)^2 - 4\beta^2)}} \right)\Bigg|_{-\pi}^{+\pi}$$

$$= \frac{2\alpha^2}{(1 - \beta^2)} \arctan\left(\frac{(1 + \beta)^2 \tan(\frac{\Omega}{2})}{(1 - \beta^2)} \right)\Bigg|_{-\pi}^{+\pi}$$

$$= \frac{2\alpha^2}{(1 - \beta^2)} \left[\arctan\left(\frac{(1 + \beta)^2}{(1 - \beta^2)} \tan(+\frac{\pi}{2}) \right) - \arctan\left(\frac{(1 + \beta)^2}{(1 - \beta^2)} \tan(-\frac{\pi}{2}) \right) \right]$$

$$Mit: \tan(-\xi) = -\tan(\xi) \quad und \quad \arctan(-\xi) = -\arctan(\xi)$$

$$= \frac{4\alpha^2}{(1 - \beta^2)} \arctan\left(\frac{(1 + \beta)^2}{(1 - \beta^2)} \tan(\frac{\pi}{2}) \right)$$

$$Mit: \tan(\frac{\pi}{2}) = Pol\ erster\ Ordnung \ _ \ \arctan(+\infty) = \frac{\pi}{2}$$

$$\int_{-\pi}^{+\pi} \frac{S_{yy}(\Omega)}{S_{xx}(\Omega)} d\Omega = 2\pi \frac{\alpha^2}{1 - \beta^2}$$

$$_\ Leistungskorrekturfaktor = \frac{\sqrt{1 - \beta^2}}{\alpha}$$

Anhang B

Zusammenhang zwischen Filterkoeffizient und Zeitkonstante

$$Y(k) = (1-\alpha)X(k) + \alpha Y(k-1)$$

$$Y(k) - \alpha Y(k-1) = (1-\alpha)X(k)$$

Mit : X(k)= S(k)= Sprungfolge.

$$_\ y_n - \alpha\, y_{n-1} = (1-\alpha)\, s_n$$

Z - Transformation

$$_\ Y(z) - Y(z)\alpha\, z^{-1} = (1-\alpha)\frac{z}{z-1}$$

$$Y(z) = \frac{1-\alpha}{1-\alpha\, z^{-1}}\frac{z}{z-1}$$

$$Y(z) = (1-\alpha)\frac{z^2}{(z-\alpha)(z-1)} = (1-\alpha)z\frac{z}{(z-\alpha)(z-1)}$$

$$Y(z) = (1-\alpha)z\left(\frac{d_1}{z-\alpha} + \frac{d_2}{z-1}\right)$$

$$_\ d_1 = 1 - \frac{1}{1-\alpha} \qquad d_2 = \frac{1}{1-\alpha}$$

$$_\ Y(z) = \frac{-\alpha z}{z-\alpha} + \frac{z}{z-1}$$

Z - Rücktransformation

$$_\ y_n = 1 - \alpha^{n+1}$$

$$n = \frac{\ln(1-y_n)}{\ln(\alpha)} - 1$$

$$_\ n_{90\%} - n_{10\%} = n_{Anstieg} = \frac{\ln(0.1) - \ln(0.9)}{\ln(\alpha)}$$

$$_\ t_{Anstieg}\,[ms] = \frac{1000\, n_{Anstieg}}{f_{Sample}} = \frac{\ln(0.1) - \ln(0.9)}{\ln(\alpha)}\frac{1000}{f_{Sample}}$$

$$_\ \alpha = e^{\frac{\ln(0.1) - \ln(0.9)}{t_{Anstieg}\,[ms]}\frac{1000}{f_{Sample}}}$$

Anhang C

Listing des Quellcodes von SPV

SPVINC.H

Anhang C: Listing des Quellcodes von SPV (SPVINC.H)

```
/*********************************************************************/
/* SPV                                                               */
/* Autor: Harald Zottmann                                            */
/* Matr.: 787378                                                     */
/* Studienarbeit 1992/93                                             */
/* Technische-Hochschule-Darmstadt                                   */
/*********************************************************************/

#define  BOOL         unsigned char
#define  UBYTE        unsigned char
#define  UWORD        unsigned int
#define  FALSE        0
#define  TRUE         1

#define  PI           3.14159265359
#define  PI2          6.28318530718

#define  XREL         51             /* Koordinaten der Auswahlbox */
#define  YREL         6
#define  OPTX         20
#define  WIDTH        28

#define  XSPACE       50             /* Freiraum im Graphikbildschirm */
#define  YSPACE       100

#define  MAXFUNK      23             /* Maximal angezeigte Funktionen */
#define  FUNKANZ      19             /* Anzahl der Funktionen */

#define  SPVCFG       "SPV.CFG"
#define  TEMPDATEI    "SPV.TMP"

#define  CURSORANYKEY "\x1B\x18\x19\x1A - Cursor  /  Weiter mit jeder anderen Taste !"
#define  PARAKEY      "\x1B\x18\x19\x1A - Cursor  /  [ESC] - Parameter speichern !"
#define  ANYKEY       "Weiter mit einer beliebigen Taste !"
#define  HISTOANYKEY  "[RETURN] - Nächstes Histogramm  /  Weiter mit jeder anderen Taste !"
#define  NODATAFILE   "Bitte nur Datenfiles anwählen !"

/*********************************************************************/

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <mem.h>
#include <dir.h>
#include <dos.h>
#include <alloc.h>
#include <math.h>

/*********************************************************************/

BOOL     savegraph            ();
```

Anhang C: Listing des Quellcodes von SPV (SPVINC.H)

```
BOOL      graphicon            ();
void      graphicoff           ();

void      settings             ();
void      koeff                ();
void      drawkenn             ();

BOOL      getdat               ();
BOOL      displaydata          ();
BOOL      histogramm           ();

BOOL      suffix               ();

int       option               ();
int       getsamplength ();
int       inputfilename        ();
int       inputnumber          ();

char      *getfilename  ();
int       input                ();
void      showpercent          ();
BOOL      showerror            ();
BOOL      showstats            ();
void      busy                 ();
void      beep                 ();
void      shortbeep            ();

BOOL      combine              ();
BOOL      chainfiles           ();
BOOL      norm                 ();
BOOL      korr                 ();
BOOL      fastfourier          ();
BOOL      kenn                 ();
BOOL      makesirp             ();

double    kennlinie            ();
double    erf                  ();
void      gleichvert           ();
void      laplacevert          ();
void      normalvert           ();

typedef struct  {
                        UBYTE   Ident, Version, Compression, BitPerPixel;
                        UWORD   XMin, YMin, XMax, YMax, XDpi, YDpi;
                        UBYTE   ColorMap [16] [3];
                        UBYTE   Reserved, BitPlanes;
                        UWORD   BytesPerRow, CMapInterpretation, Padding [29];
                        }PCXHEADER;

typedef struct          {
                        double  ua, us,         /* Kennlinienparameter */
                                d, p, q, r,
                                a0, a1, a2, a3, /* Polynom */
```

```
                        ag, bg;         /* Glättungsfilter */
                    }KENNLINIE;

typedef struct      {
                    double  ar,                     /* Alpha-Rising */
                            af,                     /* Alpha-Falling */
                            ac,                     /* Korrekturfaktor */
                            ae;                     /* Endwert */
                    }SAMPARAMETER;

typedef   struct    {
                    double as,
                           bs;
                    }SIRPFILTER;

typedef struct  {
                    double  a,                      /* -a...+a bei Gleichverteilung */
                            ld,                     /* Lambda  bei Laplaceverteilung */
                            sg;                     /* Sigma   bei Normalverteilung */
                    }VERTEILUNG;
```

SPV.C

Anhang C: Listing des Quellcodes von SPV (SPV.C)

```
/*********************************************************************/
/* SPV                                                               */
/* Autor: Harald Zottmann                                            */
/* Matr.: 787378                                                     */
/* Studienarbeit 1992/93                                             */
/* Technische-Hochschule-Darmstadt                                   */
/*********************************************************************/

#include "SPVINC.H"

/*********************************************************************/
/* Anzahl der Funktionen in FUNKANZ (File: SPVINC.H) definieren      */
/*********************************************************************/

char     *funktion[] = {      "Verzeichnis wechseln",
                              "Parameter einstellen",
                              "Eingang erzeugen",
                              "Kennlinie zeigen",
                              "Histogramm zeigen",
                              "Zeitfunktion zeigen",
                              "Korrelation zeigen ",
                              "Spektrum zeigen",
                              "File löschen",
                              "File umbenennen",
                              "File bearbeiten   (DAT)",
                              "Files verketten   (DAT)",
                              "Files kombinieren (DAT)",
                              "Autokorrelation   (AKF)",
                              "Kreuzkorrelation  (KKF)",
                              "Fourier           (FFT)",
                              "Schätzwert        (SAM)",
                              "Verstärkung       (AMP)",
                              "Ausgangssignal    (SIG)" };

/*********************************************************************/
/* Globale Variablen                                                 */
/*********************************************************************/

int                    xgmax, ygmax;                 /* Maximale Graphikauflösung */

BOOL                   qcalc   = FALSE;              /* Schneller Rechenmodus */
BOOL                   qdisp   = FALSE;              /* Schneller Displaymodus */
BOOL                   nobeep  = FALSE;              /* Klang */

double                 fg      = 12000.0;            /* fg  = 1 / (2*ta) */

KENNLINIE kennpar;
SAMPARAMETER           sampar;
SIRPFILTER             sirpfil;
VERTEILUNG             verteilung;

char                   workpath [MAXPATH];           /* Startdirectory */
```

Anhang C: Listing des Quellcodes von SPV (SPV.C)

```
/*********************************************************************/
/* Programm bei Fließkommafehler sauber beenden                      */
/*********************************************************************/

int matherr (struct exception *ep)
{
         beep ();
         showerror (7);
         ep->retval = 0.0;
         chdir (workpath);
         graphicoff ();

         abort ();            /* Programmende mit Fehlercode */

return (0);
}

/*********************************************************************/
/* Hauptprogramm                                                     */
/*********************************************************************/

void     main (int argc, char *argv[])
{
void     maske  ();
void     select ();

FILE     *fp;
int      fkt          = 1;
BOOL     ok           = TRUE;

         while (argc > 1)
                      {
                      if      (stricmp (argv[argc-1], "QCALC") == 0)
                              {
                              qcalc = TRUE;
                              }
                      else if (stricmp (argv[argc-1], "QDISP") == 0)
                              {
                              qdisp = TRUE;
                              }
                      else if (stricmp (argv[argc-1], "NOBEEP") == 0)
                              {
                              nobeep = TRUE;
                              }
                      else
                              {
         printf ("\r\n\nSPV V1.6  -  Autor: Harald Zottmann  -  Datum: 1993\r\n\n");
         printf ("SPV [QCALC] [QDISP] [NOBEEP]\r\n\n");
         printf ("QCALC        Nur Mittelwert und Sigma für  \" Schätzwert     (SAM) \"\r\n");
         printf ("                                           \" Verstärkung    (AMP) \"\r\n");
         printf ("                                           \" Ausgangssignal (SIG) \"\r\n\n");
         printf ("QDISP        Schnellmodus für              \" Zeitfunktion zeigen  \"\r\n");
         printf ("                                           \" Korrelation zeigen   \"\r\n");
```

```
printf ("                                          \" Spektrum zeigen      \"\r\n\n");
printf ("NOBEEP       Sound aus\r\n\n\n");
                    exit (0);
                    }
            argc--;
            }

graphicon ();

getcwd (workpath, MAXPATH);
fp = fopen (SPVCFG, "rb");
if (fp == NULL)
            {
            fp = fopen (SPVCFG, "wb");
                    {
                    if (fp == NULL)
                            {
                            beep ();
                            showerror (2);
                            }
                    else
                            {
                            kennpar.ua = 0.5;      kennpar.us = 0.1;
                            kennpar.d  = 0.708;    kennpar.p  = 6.0;
                            kennpar.q  = 0.708;    kennpar.r  = 2.0;
                            kennpar.ag = 1.0;      kennpar.bg = 0.0;
                            koeff ();

                            sampar.ar = 0.9;
                            sampar.af = 0.9;
                            sampar.ae = 0.0;
                            sampar.ac = 1.0;

                            verteilung.a  = 1.73205;
                            verteilung.ld = 1.41421;
                            verteilung.sg = 1.0;

                            sirpfil.as = 1.0;
                            sirpfil.bs = 0.0;

                            fwrite (&fg,           sizeof (double),       1, fp);
                            fwrite (&sampar.ar,    sizeof (SAMPARAMETER), 1, fp);
                            fwrite (&kennpar.ua,   sizeof (KENNLINIE),    1, fp);
                            fwrite (&sirpfil.as,   sizeof (SIRPFILTER),   1, fp);
                            fwrite (&verteilung.a, sizeof (VERTEILUNG),   1, fp);
                            fclose (fp);
                            }
                    }
            }
else
            {
            fread (&fg,           sizeof (double),       1, fp);
            fread (&sampar.ar,    sizeof (SAMPARAMETER), 1, fp);
```

```
                fread (&kennpar.ua,   sizeof (KENNLINIE),    1, fp);
                fread (&sirpfil.as,   sizeof (SIRPFILTER),   1, fp);
                fread (&verteilung.a, sizeof (VERTEILUNG),   1, fp);
                fclose (fp);
                }

        maske (&fkt, 1);

        do{
                switch (getch ())
                        {
                        case 0x00:
                        switch (getch ())
                                {
                                case 'P':               /* Cursor Down */
                                fkt++;
                                maske (&fkt, 0);
                                break;

                                case 'H':                /* Cursor Up */
                                fkt--;
                                maske (&fkt, 0);
                                break;

                                default:
                                break;
                                }
                        break;

                        case 0x67:                      /* G */
                        savegraph ((UWORD) 349);
                        maske (&fkt, 1);
                        break;

                        case 0x1B:                      /* Escape */
                        if (option (OPTX, YREL, 2, "Programm beenden:", "Nein", "Ja") == 2)
                                {
                                ok = FALSE;
                                }
                        else
                                {
                                maske (&fkt, 1);
                                }
                        break;

                        case 0x0D:                      /* Return */
                        select (fkt);
                        maske (&fkt, 1);
                        break;

                        default:
                        break;
                        }
```

```
        }while (ok);

        chdir (workpath);
        graphicoff ();

exit (0);
}

/*********************************************************************/

void    maske (int *fkt, BOOL mode)
{
static  int         offset = 0;
int     i;

        if (*fkt > FUNKANZ)
                    {
                    *fkt = 1;
                    }
        if (*fkt < 1)
                    {
                    *fkt = FUNKANZ;
                    }
        if (*fkt > (offset + MAXFUNK))
                    {
                    offset++;
                    setviewport ((XREL+3)*8, (YREL+3)*10, (XREL+WIDTH-2)*8, (YREL+MAXFUNK+2)*10, 1);
                    clearviewport ();
                    }
        if (*fkt < offset + 1)
                    {
                    offset = *fkt - 1;
                    setviewport ((XREL+3)*8, (YREL+3)*10, (XREL+WIDTH-2)*8, (YREL+MAXFUNK+2)*10, 1);
                    clearviewport ();
                    }

        if (mode == 1)
                    {
                    setviewport (0, 0, xgmax, ygmax, 1);
                    clearviewport ();
                    setcolor (BLUE);
                    setlinestyle (SOLID_LINE, 0xFFFF, THICK_WIDTH);
                    setwritemode (COPY_PUT);
                    rectangle (0, 0, xgmax, 12);
                    setcolor (YELLOW);
                    outtextxy (20 , 3, "SPV V1.6");
                    outtextxy (430, 3, "Autor: Harald Zottmann");

                    setviewport (XREL*8, YREL*10, (XREL+WIDTH)*8, (YREL+MAXFUNK+5)*10, 1);
                    clearviewport ();
                    setcolor (BLUE);
                    rectangle (1*8, 0*10,  WIDTH   *8, (MAXFUNK+4)*10);
                    rectangle (2*8, 2*10, (WIDTH-1)*8, (MAXFUNK+3)*10);
```

```
                        setcolor (YELLOW);
                        outtextxy (3*8, 1*10,"Funktionen:");
                        }

          for (i = offset; i < offset+MAXFUNK; i++)
                        {
                        if (i >= FUNKANZ)
                                {
                                break;
                                }
                        if (i+1 == *fkt)
                                {
                                setcolor (WHITE);
                                }
                        else
                                {
                                setcolor (DARKGRAY);
                                }
                        outtextxy (3*8, (i-offset+3)*10, funktion[i]);
                        }
return;
}

/*********************************************************************/

void      select (int fkt)
{
char     *filenameptr,
          in_file1 [13] = "",
          in_file2 [13] = "";
int       ret;

          switch (fkt)
                        {
                        case 1:        /* Verzeichnis wechseln */
                        filenameptr = getfilename ();
                        break;

                        case 2:        /* Parameter eingeben */
                        settings ();
                        break;

                        case 3:        /* Eingang erzeugen */
                        makesirp ();
                        break;

                        case 4:        /* Kennlinie anzeigen */
                        drawkenn ();
                        break;

                        case 5:        /* Histogramm */
                        filenameptr = getfilename ();
                        if (*filenameptr != 0x00)
```

```
        {
        histogramm (filenameptr);
        }
break;

case 6:        /* Zeitfunktion anzeigen */
ret = option (OPTX, YREL, 2, "Anzahl:", "Eine Funktion", "Zwei Funktionen");
if (ret == 0)
        {
        break;
        }
filenameptr = getfilename ();
if (*filenameptr == 0x00)
        {
        break;
        }
strcpy (&in_file1[0], filenameptr);

if (ret == 2)
        {
        filenameptr = getfilename ();
        if (*filenameptr == 0x00)
                {
                break;
                }
        strcpy (&in_file2[0], filenameptr);
        }
displaydata (&in_file1[0], &in_file2[0], 0);
break;

case 7:        /* Korrelation anzeigen */
ret = option (OPTX, YREL, 2, "Anzahl:", "Eine Korrelation", "Zwei Korrelationen");
if (ret == 0)
        {
        break;
        }
filenameptr = getfilename ();
if (*filenameptr == 0x00)
        {
        break;
        }
strcpy (&in_file1[0], filenameptr);

if (ret == 2)
        {
        filenameptr = getfilename ();
        if (*filenameptr == 0x00)
                {
                break;
                }
        strcpy (&in_file2[0], filenameptr);
        }
displaydata (&in_file1[0], &in_file2[0], 1);
```

```
break;

case 8:        /* Spektrum anzeigen */
ret = option (OPTX, YREL, 2, "Anzahl:", "Ein Spektrum", "Zwei Spektren");
if (ret == 0)
        {
        break;
        }
filenameptr = getfilename ();
if (*filenameptr == 0x00)
        {
        break;
        }
strcpy (&in_file1[0], filenameptr);

if (ret == 2)
        {
        filenameptr = getfilename ();
        if (*filenameptr == 0x00)
                {
                break;
                }
        strcpy (&in_file2[0], filenameptr);
        }
displaydata (&in_file1[0], &in_file2[0], 2);
break;

case 9:        /* File löschen */
filenameptr = getfilename ();
if (*filenameptr != 0x00)
        {
        beep ();
        if (showerror (20))
                {
                if (remove (filenameptr) != 0)
                        {
                        beep ();
                        showerror (11);
                        }
                }
        }
break;

case 10:       /* File umbenennen */
filenameptr = getfilename ();
if (*filenameptr != 0x00)
        {
        if (strchr (filenameptr, '.') == NULL)
                {
                strcpy (in_file2, "DAT");
                }
        else
                {
```

```
                strcpy (in_file2, strchr (filenameptr, '.') + 1);
                }
        ret = inputfilename ("Neuer Filename:", in_file2, filenameptr, in_file1);
        if ((strcmp (filenameptr, in_file1) == 0) || (ret == 0))
                {
                break;
                }
        if (rename (filenameptr, in_file1) != 0)
                {
                beep ();
                showerror (12);
                break;
                }
        }
break;

case 11:        /* File bearbeiten */
filenameptr = getfilename ();
if (*filenameptr != 0x00)
        {
        norm (filenameptr);
        }
break;

case 12:        /* Files verketten */
chainfiles ();
break;

case 13:        /* Files linear kombinieren */
combine ();
break;

case 14:        /* AKF */
korr (0);
break;

case 15:        /* KKF */
korr (1);
break;

case 16:        /* FFT */
filenameptr = getfilename ();
if (*filenameptr != 0x00)
        {
        fastfourier (filenameptr);
        }
break;

case 17:        /* SAM */
filenameptr = getfilename ();
if (*filenameptr != 0x00)
        {
        kenn (filenameptr, 0);
```

```
                        }
                    break;

                    case 18:       /* Verstärkung */
                    filenameptr = getfilename ();
                    if (*filenameptr != 0x00)
                        {
                        kenn (filenameptr, 1);
                        }
                    break;

                    case 19:       /* Verstärkung * Signal */
                    filenameptr = getfilename ();
                    if (*filenameptr != 0x00)
                        {
                        kenn (filenameptr, 2);
                        }
                    break;

                    default:
                    break;
                    }
return;
}

/*********************************************************************/
/* Parameter einstellen                                              */
/*********************************************************************/

void     settings ()
{
void     settingsmaske        ();
FILE                 *fp;
char                 num      [20],
                     fullpath [80];
double               old;
static int           pos = 1;
int                  ret = 0;

         settingsmaske        ();

         do{
         switch (pos)
                     {
                     case 1:
                     sprintf (num, "%.1f", fg*2);
                     ret = input (6, 5, 13, &num[0],1);
                     if (ret == 20)
                         {
                         savegraph ((UWORD) 349);
                         }
                     fg  = atof (&num[0]) / 2;
                     if (fg <= 0.0)
```

```
                        {
                        shortbeep ();
                        showerror (10);
                        ret = 0;
                        }
                break;

                case 2:
                sprintf (num, "%.8g", sampar.ar);
                ret = input (6, 9, 13, &num[0],1);
                if (ret == 20)
                        {
                        savegraph ((UWORD) 349);
                        }
                sampar.ar  = atof (&num[0]);
                if ((sampar.ar < 0.0) || (sampar.ar > 1.0))
                        {
                        shortbeep ();
                        showerror (10);
                        ret = 0;
                        }
                break;

                case 3:
                sprintf (num, "%.8g", sampar.af);
                ret = input (25, 9, 13, &num[0],1);
                if (ret == 20)
                        {
                        savegraph ((UWORD) 349);
                        }
                sampar.af  = atof (&num[0]);
                if ((sampar.af < 0.0) || (sampar.af > 1.0))
                        {
                        shortbeep ();
                        showerror (10);
                        ret = 0;
                        }
                break;

                case 4:
                sprintf (num, "%.8g", sampar.ac);
                sampar.ac  = atof (&num[0]);
                old = sampar.ac;
                ret = input (65, 9, 13, &num[0],1);
                if (ret == 20)
                        {
                        savegraph ((UWORD) 349);
                        }
                if (ret == 0)
                        {
ret = option (42, 5-3, 3, "SAM - Korrekturfaktor:", "Gleichverteilung",
                                        "Laplaceverteilung",
                                        "Gaußverteilung");
```

```
        switch (ret)
                {
                case 1:        /* Gleichverteilung */
                gleichvert ();
                break;

                case 2:        /* Laplaceverteilung */
                setcolor (LIGHTRED);
                outtextxy (65*8, 9*10, "Bitte Warten");
                laplacevert ();
                break;

                case 3:        /* Gaußverteilung */
                setcolor (LIGHTRED);
                outtextxy (65*8, 9*10, "Bitte Warten");
                normalvert ();
                break;

                default:
                break;
                }
        settingsmaske ();
        ret = 0;
        }
else
        {
        sampar.ac  = atof (&num[0]);
        if ((sampar.ac != old) && (ret != 50))
                {
                settingsmaske ();
                }
        }
break;

case 5:
sprintf (num, "%.8g", kennpar.ua);
ret = input (6, 13, 13, &num[0],1);
if (ret == 20)
        {
        savegraph ((UWORD) 349);
        }
kennpar.ua  = atof (&num[0]);
if (kennpar.ua <= 0.0)
        {
        shortbeep ();
        showerror (10);
        ret = 0;
        }
break;

case 6:
sprintf (num, "%.8g", kennpar.us);
ret = input (25, 13, 13, &num[0],1);
```

```
if (ret == 20)
        {
        savegraph ((UWORD) 349);
        }
kennpar.us  = atof (&num[0]);
if (kennpar.us <= 0.0)
        {
        shortbeep ();
        showerror (10);
        ret = 0;
        }
break;

case 7:
sprintf (num, "%.8g", kennpar.d);
ret = input (6, 14, 13, &num[0],1);
if (ret == 20)
        {
        savegraph ((UWORD) 349);
        }
kennpar.d  = atof (&num[0]);
if (kennpar.d <= 0.0)
        {
        shortbeep ();
        showerror (10);
        ret = 0;
        }
break;

case 8:
sprintf (num, "%.8g", kennpar.p);
ret = input (25, 14, 13, &num[0],1);
if (ret == 20)
        {
        savegraph ((UWORD) 349);
        }
kennpar.p  = atof (&num[0]);
if (kennpar.p <= 1.0)
        {
        shortbeep ();
        showerror (10);
        ret = 0;
        }
break;

case 9:
sprintf (num, "%.8g", kennpar.q);
ret = input (46, 14, 13, &num[0],1);
if (ret == 20)
        {
        savegraph ((UWORD) 349);
        }
kennpar.q  = atof (&num[0]);
```

```
if (kennpar.q <= 0.0)
        {
        shortbeep ();
        showerror (10);
        ret = 0;
        }
break;

case 10:
sprintf (num, "%.8g", kennpar.r);
ret = input (65, 14, 13, &num[0],1);
if (ret == 20)
        {
        savegraph ((UWORD) 349);
        }
kennpar.r  = atof (&num[0]);
if (kennpar.r <= 0.0)
        {
        shortbeep ();
        showerror (10);
        ret = 0;
        }
break;

case 11:
sprintf (num, "%.8g", kennpar.ag);
ret = input (6, 18, 13, &num[0],1);
if (ret == 20)
        {
        savegraph ((UWORD) 349);
        }
if (ret == 0)
        {
        kennpar.ag = 1.0 - kennpar.bg;
        }
else
        {
        kennpar.ag = atof (&num[0]);
        }
break;

case 12:
sprintf (num, "%.8g", kennpar.bg);
ret = input (25, 18, 13, &num[0],1);
if (ret == 20)
        {
        savegraph ((UWORD) 349);
        }
if (ret == 0)
        {
        kennpar.bg = 1.0 - kennpar.ag;
        }
else
```

```
        {
        kennpar.bg = atof (&num[0]);
        }
break;

case 13:
sprintf (num, "%.8g", sirpfil.as);
ret = input (46, 18, 13, &num[0],1);
if (ret == 20)
        {
        savegraph ((UWORD) 349);
        }
if (ret == 0)
        {
        sirpfil.as = 1.0 - sirpfil.bs;
        }
else
        {
        sirpfil.as = atof (&num[0]);
        }
break;

case 14:
sprintf (num, "%.8g", sirpfil.bs);
ret = input (65, 18, 13, &num[0],1);
if (ret == 20)
        {
        savegraph ((UWORD) 349);
        }
if (ret == 0)
        {
        sirpfil.bs = 1.0 - sirpfil.as;
        }
else
        {
        sirpfil.bs = atof (&num[0]);
        }
break;

case 15:
sprintf (num, "%.8g", verteilung.a);
ret = input (5, 25, 13, &num[0],1);
if (ret == 20)
        {
        savegraph ((UWORD) 349);
        }
verteilung.a  = atof (&num[0]);
if (verteilung.a <= 0.0)
        {
        shortbeep ();
        showerror (10);
        ret = 0;
        }
```

```
                break;

                case 16:
                sprintf (num, "%.8g", verteilung.ld);
                ret = input (25, 25, 13, &num[0],1);
                if (ret == 20)
                        {
                        savegraph ((UWORD) 349);
                        }
                verteilung.ld = atof (&num[0]);
                if (verteilung.ld <= 0.0)
                        {
                        shortbeep ();
                        showerror (10);
                        ret = 0;
                        }
                break;

                case 17:
                sprintf (num, "%.8g", verteilung.sg);
                ret = input (45, 25, 13, &num[0],1);
                if (ret == 20)
                        {
                        savegraph ((UWORD) 349);
                        }
                verteilung.sg = atof (&num[0]);
                if (verteilung.sg <= 0.0)
                        {
                        shortbeep ();
                        showerror (10);
                        ret = 0;
                        }
                break;

                default:
                break;
                }

        switch (ret)
                {
                case 1:
                case 2:
                pos++;
                break;

                case -1:
                case -2:
                pos--;
                break;

                default:
                break;
                }
```

```
        if (pos <= 0 )       pos = 17;
        if (pos >= 18)  pos =  1;

        }while (ret != 50);

        koeff ();

        strcpy (fullpath,workpath);
        strcat (fullpath,"\\");
        strcat (fullpath,SPVCFG);
        fp = fopen (fullpath, "wb");
        if (fp == NULL)
                    {
                    beep ();
                    showerror (2);
                    return;
                    }
        fwrite (&fg,           sizeof (double),       1, fp);
        fwrite (&sampar.ar,    sizeof (SAMPARAMETER), 1, fp);
        fwrite (&kennpar.ua,   sizeof (KENNLINIE),    1, fp);
        fwrite (&sirpfil.as,   sizeof (SIRPFILTER),   1, fp);
        fwrite (&verteilung.a, sizeof (VERTEILUNG),   1, fp);
        fclose (fp);
        setviewport (0, 0, xgmax, ygmax, 1);
        clearviewport ();

return;
}

void    settingsmaske ()
{
void    lambda       ();
char    num          [20];

        setviewport (0, 0, xgmax, ygmax, 1);
        clearviewport ();
        setcolor (BLUE);
        rectangle (0, 3*10, 20*8, 6*10+2);
        setcolor (YELLOW);
        outtextxy (2*8, 4*10, "Samplefrequenz:");
        outtextxy (2*8, 5*10, "\x9Fs:");
        setcolor (DARKGRAY);
        sprintf (num, "%.1f",fg*2);
        outtextxy (6*8, 5*10, num);

        setcolor (BLUE);
        rectangle (0, 7*10, 39*8, 10*10+2);
        setcolor (YELLOW);
        outtextxy (2*8, 8*10,"SAM - Zeitkonstanten:");
        outtextxy (2*8, 9*10, "\xE0\x72:               \xE0\x66:");
        setcolor (DARKGRAY);
        sprintf (num, "%.8g",sampar.ar);
        outtextxy (6*8, 9*10, num);
```

```
sprintf (num, "%.8g",sampar.af);
outtextxy (25*8, 9*10, num);

setcolor (BLUE);
rectangle (40*8, 7*10, 79*8, 10*10+2);
setcolor (YELLOW);
outtextxy (42*8, 8*10, "SAM - Korrekturfaktor:");
outtextxy (42*8, 9*10, "Y :                Yc:");
outtextxy (43*8, 9*10+2, "\xEC");
setcolor (LIGHTGRAY);
sprintf (num, "%.8g",sampar.ae);
outtextxy (46*8, 9*10, num);
setcolor (DARKGRAY);
sprintf (num, "%.8g",sampar.ac);
outtextxy (65*8, 9*10, num);

setcolor (BLUE);
rectangle (0, 11*10, 79*8, 15*10+2);
setcolor (YELLOW);
outtextxy (2*8, 12*10, "Kennlinienparameter:");
outtextxy (2*8, 13*10, "Ya:                Yo:");
outtextxy (2*8, 14*10, "kd:                kp:                  kq:                kr:");
setcolor (DARKGRAY);
sprintf (num, "%.8g",kennpar.ua);
outtextxy (6*8, 13*10, num);
sprintf (num, "%.8g",kennpar.us);
outtextxy (25*8, 13*10, num);
sprintf (num, "%.8g",kennpar.d);
outtextxy (6*8,  14*10, num);
sprintf (num, "%.8g",kennpar.p);
outtextxy (25*8, 14*10, num);
sprintf (num, "%.8g",kennpar.q);
outtextxy (46*8, 14*10, num);
sprintf (num, "%.8g",kennpar.r);
outtextxy (65*8, 14*10, num);

setcolor (BLUE);
rectangle (0, 16*10, 39*8, 19*10+2);
setcolor (YELLOW);
outtextxy (2*8, 17*10, "Glättungsfilter - Zeitkonstanten:");
outtextxy (2*8, 18*10, "\xE0g:                \xE1g:");
setcolor (DARKGRAY);
sprintf (num, "%.8g",kennpar.ag);
outtextxy ( 6*8, 18*10 ,num);
sprintf (num, "%.8g",kennpar.bg);
outtextxy (25*8, 18*10 ,num);

setcolor (BLUE);
rectangle (40*8, 16*10, 79*8, 19*10+2);
setcolor (YELLOW);
outtextxy (42*8, 17*10, "Eingangsfilter - Zeitkonstanten:");
outtextxy (42*8, 18*10, "\xE0\x65:                \xE1\x65:");
setcolor (DARKGRAY);
```

```
sprintf (num, "%.8g",sirpfil.as);
outtextxy (46*8, 18*10 ,num);
sprintf (num, "%.8g",sirpfil.bs);
outtextxy (65*8, 18*10 ,num);

setcolor (BLUE);
rectangle (0, 20*10, 19*8, 26*10+2);
setcolor (LIGHTGREEN);
setlinestyle (SOLID_LINE, 0, NORM_WIDTH);
outtextxy (3*8,   23*10-4, "1");
outtextxy (2*8+4, 24*10-4, "2a");
line (2*8,   24*10-6, 4*8+4, 24*10-6);
outtextxy (8*8,   23*10, "\xB3y\xB3 \xF3 a");
setcolor (YELLOW);
outtextxy (2*8,   21*10, "Gleichverteilung:");
outtextxy (2*8, 25*10, "a:");
setcolor (DARKGRAY);
sprintf (num, "%.8g", verteilung.a);
outtextxy (5*8, 25*10, num);

setcolor (BLUE);
setlinestyle (SOLID_LINE, 0, THICK_WIDTH);
rectangle (20*8, 20*10, 39*8, 26*10+2);
setcolor (LIGHTGREEN);
setlinestyle (SOLID_LINE, 0, NORM_WIDTH);
outtextxy (23*8, 23*10-3, "1");
lambda (23*8, 24*10-3);
line (22*8, 24*10-5, 24*8+4, 24*10-5);
outtextxy (25*8, 23*10+1, "e");
outtextxy (26*8, 23*10-7, "- \xB3y\xB3");
lambda (27*8, 23*10-7);
setcolor (YELLOW);
outtextxy (22*8, 21*10, "Laplace:");
lambda (22*8, 25*10);
outtextxy (23*8, 25*10, ":");
setcolor (DARKGRAY);
sprintf (num, "%.8g", verteilung.ld);
outtextxy (25*8, 25*10, num);

setcolor (BLUE);
setlinestyle (SOLID_LINE, 0, THICK_WIDTH);
rectangle (40*8, 20*10, 59*8, 26*10+2);
setcolor (LIGHTGREEN);
setlinestyle (SOLID_LINE, 0, NORM_WIDTH);
outtextxy (44*8,   23*10-6, "1");
outtextxy (43*8,   24*10-2, "2\xE3 \xE5");
line      (42*8,   24*10-8, 47*8+4, 24*10-8);
outtextxy (42*8,   24*10-4, "\xFB");
outtextxy (43*8,   24*10-8, "\xC4\xC4");
outtextxy (48*8,   23*10-2, "e");
outtextxy (49*8+4, 21*10+4, "-y\xFD");
outtextxy (49*8+4, 22*10+4, "2\xE5\xFD");
line      (49*8,   22*10+2, 52*8+8, 22*10+2);
```

```
         setcolor (YELLOW);
         outtextxy (42*8,   21*10, "Gauß:");
         outtextxy (42*8,   25*10, "\xE5:");
         setcolor (DARKGRAY);
         sprintf (num, "%.8g", verteilung.sg);
         outtextxy (45*8, 25*10, num);

         setcolor (WHITE);
         outtextxy (0, 340, PARAKEY);
return;
}

/*********************************************************************/
/* Zeichnen von Lambda                                               */
/*********************************************************************/

void     lambda (int lx, int ly)
{
         ly -= 1;

         line (lx+0, ly+1, lx+1, ly+1);
         line (lx+2, ly+2, lx+6, ly+9);
         line (lx+6, ly+9, lx+7, ly+9);

         line (lx+0, ly+9, lx+1, ly+9);
         line (lx+1, ly+9, lx+3, ly+5);

return;
}

/*********************************************************************/
/* Gleichungssystem des Übergangspolynoms lösen                      */
/*********************************************************************/

void     koeff ()
{
double   a [4] [5],
         f;
int      i, s, x, y;

         a [0] [0] = a [1] [0] = a [2] [1] = a [3] [1] = 1.0;
         a [2] [0] = a [3] [0] = 0.0;
         a [0] [1] =          kennpar.q - 1.0;
         a [1] [1] =          kennpar.r - 1.0;
         a [0] [2] =       pow (kennpar.q - 1.0, 2.0);
         a [1] [2] =       pow (kennpar.r - 1.0, 2.0);
         a [2] [2] = 2.0 *     (kennpar.q - 1.0);
         a [3] [2] = 2.0 *     (kennpar.r - 1.0);
         a [0] [3] =       pow (kennpar.q - 1.0, 3.0);
         a [1] [3] =       pow (kennpar.r - 1.0, 3.0);
         a [2] [3] = 3.0 * pow (kennpar.q - 1.0, 2.0);
         a [3] [3] = 3.0 * pow (kennpar.r - 1.0, 2.0);
```

```
        if ((kennpar.us == 0.0) || (kennpar.r == 0.0))
                    {
                    showerror (8);
                    return;
                    }
        f = kennpar.ua / kennpar.us;
        a [0] [4] = f * (pow (kennpar.q, kennpar.p - 1));
        a [1] [4] = f * ( 1 / kennpar.r);
        a [2] [4] = f * ((kennpar.p - 1) * pow (kennpar.q, kennpar.p - 2));
        a [3] [4] = f * (-1 / pow (kennpar.r, 2));

        for (s = 1; s <= 3; s++)                          /* Gauß */
                    {
                    for (y = s; y <= 3; y++)
                            {
                            for (x = s; x <= 4; x++)      /* RR */
                                    {
                                    if (a [s-1] [s-1] == 0.0)
                                            {
                                            showerror (8);
                                            return;
                                            }
                                    a [y] [x] = a [y] [x] - (a [s-1] [x] * a [y] [s-1])
                                              /  a [s-1] [s-1];
                                    }
                            }
                    if (a [s] [s] == 0.0)
                            {
                            for (i = s; i <= 4; i++)
                                    {
                                    f= a [s  ] [i];
                                       a [s  ] [i] = a [s+1] [i];
                                       a [s+1] [i] = f;
                                    }
                      }
                    }

        if ((a [3] [3] == 0.0) || (a [2] [2] == 0.0) ||
            (a [1] [1] == 0.0) || (a [0] [0] == 0.0))
                    {
                    showerror (8);
                    return;
                    }

kennpar.a3 =  a [3][4] / a [3][3];
kennpar.a2 = (a [2][4] - kennpar.a3 * a [2][3]) / a [2][2];
kennpar.a1 = (a [1][4] - kennpar.a3 * a [1][3] - kennpar.a2 * a[1][2]) / a [1][1];
kennpar.a0 = (a [0][4] - kennpar.a3 * a [0][3] - kennpar.a2 * a[0][2] - kennpar.a1 * a[0][1]) / a[0][0];

return;
}

/*********************************************************************/
```

Anhang C: Listing des Quellcodes von SPV (SPV.C)

```
/* Graphikmodus einschalten                                          */
/*********************************************************************/

BOOL     graphicon ()
{
int      gdriver = DETECT,
         gmode,
         errorcode;

         errorcode  =  registerbgidriver (EGAVGA_driver);
         if (errorcode < 0)
                     {
                     xgmax = 0;
                     ygmax = 0;
                     return (FALSE);
                     }

         initgraph (&gdriver, &gmode, "");
         errorcode = graphresult ();
         if (errorcode != grOk)
                     {
                     xgmax = 0;
                     ygmax = 0;
                     return (FALSE);
                     }
         switch (gdriver)
                     {
                     case EGA:
                     setgraphmode (EGAHI);
                     break;

                     case VGA:
                     setgraphmode (VGAMED);
                     break;

                     default:
                     graphicoff ();
                     return (FALSE);
                     }

         setcolor (getmaxcolor ());
         xgmax = getmaxx ();
         ygmax = getmaxy ();

return (TRUE);
}

/*********************************************************************/

void     graphicoff ()
{
         closegraph ();
```

```
return;
}

/*********************************************************************/
/* Daten anzeigen                                                    */
/* mode:  0 - Zeitfunktion                                           */
/*        1 - Korrelation                                            */
/*        2 - Frequenzgang                                           */
/*********************************************************************/

BOOL     displaydata (char *filename1, char *filename2, BOOL mode)
{
void     drawkoord ();                        /* Koordinate zeichnen */
void     drawpage  ();                /* Seite zeichnen */
void     loadpage  ();                /* Seite laden */

FILE     *fp          [2];                    /* Filepointer */

char     num          [64] = "",
         outstr       [80] = "";               /* Ausgabestrings für Graphik */

double  far *ptr [2] =  {NULL, NULL}; /* Pointer auf Speicherbereich */

double   number       =  0.0,        /* Hilfsvariable */
         spnorm       =  0.0,        /* Normierung für Spektrum */
         freq         =  0.0,        /* Grenzfrequenz */
         time         [2] = {0.0, 0.0},      /* Sampledauer */
         gmittel      [2] = {0.0, 0.0},      /* Gesamt-Mittelwert */
         fmittel      [2] = {0.0, 0.0},      /* Frame-Mittelwert */
         gvar         [2] = {0.0, 0.0},      /* Gesamt-Varianz */
         fvar         [2] = {0.0, 0.0},      /* Frame-Varianz */
         wert         [2] = {0.0, 0.0},      /* Aktueller Wert */
         amplifie     [2] = {0.0, 0.0};      /* Eingestellte Verstärkung */

long     sampanz      [2] = {0L, 0L},        /* Anzahl der Werte */
         ordinate     [2] = {0L, 0L},        /* Ordinatenposition bei AKF & KKF */
         offset       [2] = {0L, 0L},        /* Offset für AKF & KKF */
         header;                             /* Fileheader bei AKF, KKF & DFT */

int      samplen      [2] = {0,0},           /* Länge eines Wertes */
         fr           [2] = {0,0},           /* Framelängen bei Spektrum */
         frame,                              /* Anzahl der Werte pro Screen */
         step,                               /* Stauchfaktor bei Spektrum normieren */
         sam,                                /* Anzahl der Werte pro Pixel */
         page,                               /* Aktuelle Seite (Startwert) */
         maxpage,                            /* Letzte Seite */
         ret,                                /* Returnwert von Options */
         xmove,                              /* Cursor um diesen Wert verschieben */
         x, i, j;

BOOL     fmt   = FALSE,                      /* Spektren mitteln */
         vgl   = TRUE,                /* Vergleichen (Zwei Files) */
         inner = TRUE,                /* Innere Anzeigeschleife */
```

```
        outer = TRUE;                       /* Äußere Anzeigeschleife */


        if (mode != 2)                      /* Frame für Zeitfkt. AKF & KKF */
                {
                ret = option (OPTX, YREL, 4, "Framelänge", "512", "1024", "2048", "4096");
                switch (ret)
                        {
                        case 1: frame =   512; break;
                        case 2: frame =  1024; break;
                        case 3: frame =  2048; break;
                        case 4: frame =  4096; break;
                        default:
                        return (FALSE);
                        }
                }

        fp[0] = fopen (filename1,"rb");
        if (fp[0] == NULL)
                {
                beep ();
                showerror (1);
                return (FALSE);
                }
        fp[1] = fopen (filename2,"rb");
        if (fp[1] == NULL)
                {
                vgl = FALSE;
                }

        busy (TRUE);
if (!getdat (fp [0], &header, &sampanz [0], &samplen [0], &amplifie [0], &gmittel [0], &gvar[0], mode))
                {
                fcloseall ();
                busy (FALSE);
                beep ();
                showerror (3);
                return (FALSE);
                }

        switch (mode)
                {
                case 0:
                time [0] = ((double) sampanz [0] / (fg * 2.0)) * 1000.0;
                break;

                case 1:
                ordinate [0] = header;
                break;

                case 2:
                frame = (int) (header / 2);
                fr [0] = frame;
```

```
                        freq  = fg;
                        break;

                        default:
                        break;
                        }


          ptr [0] = farcalloc ((unsigned long) frame, (unsigned long) sizeof(double));
          if (ptr [0] == NULL)
                        {
                        fcloseall ();
                        busy (FALSE);
                        beep ();
                        showerror (4);
                        return (FALSE);
                        }

          if (vgl)
          {
if (!getdat (fp[1], &header, &sampanz[1], &samplen[1], &amplifie[1], &gmittel[1], &gvar[1], mode))
                        {
                        farfree (ptr [0]);
                        fcloseall ();
                        busy (FALSE);
                        beep ();
                        showerror (3);
                        return (FALSE);
                        }

          switch (mode)
                        {
                        case 0:
                        time [1] = ((double) sampanz [1] / (fg * 2.0)) * 1000.0;
                        break;

                        case 1:
                        ordinate [1] = header;
                        if (ordinate [0] > ordinate [1])
                                {
                                offset [0] = 0;
                                offset [1] = ordinate [0] - ordinate [1];
                                }
                        else
                                {
                                offset [0] = ordinate [1] - ordinate [0];
                                offset [1] = 0;
                                }
                        break;

                        case 2:
                        fr [1] = (int) (header / 2);
                        if (fr [0] != fr [1])
```

```
                        {
                        fmt = TRUE;
                        }
                else
                        {
                        busy (FALSE);
                        ret = option (OPTX, YREL, 2, "Spektren anzeigen:", "Frames", "Mitteln");
                        switch (ret)
                                {
                                case 1:
                                busy (TRUE);
                                fmt = FALSE;
                                break;

                                case 2:
                                busy (TRUE);
                                fmt = TRUE;
                                break;

                                default:
                                farfree (ptr [0]);
                                fcloseall ();
                                return (FALSE);
                                }
                        }
                break;

                default:
                break;
                }
if (fmt)
                {
                ptr [1] = farcalloc ((unsigned long) fr[1], (unsigned long) sizeof(double));
                }
else
                {
                ptr [1] = farcalloc ((unsigned long) frame, (unsigned long) sizeof(double));
                }
if (ptr [1] == NULL)
                {
                farfree (ptr [0]);
                fcloseall ();
                busy (FALSE);
                beep ();
                showerror (4);
                return (FALSE);
                }
} /* if (vgl) */
else
{
if (mode == 2)
                {
                busy (FALSE);
```

```
ret = option (OPTX, YREL, 2, "Spektren anzeigen:", "Frames", "Mitteln");
switch (ret)
        {
        case 1:
        busy (TRUE);
        fmt = FALSE;
        break;

        case 2:
        busy (TRUE);
        fmt = TRUE;
        break;

        default:
        farfree (ptr [0]);
        fcloseall ();
        return (FALSE);
        }

if (fmt)
        {
        rewind (fp[0]);
        fread (&num[0], samplen[0], 1, fp[0]);

        for (j = 0; j < (int) (sampanz[0] / fr [0]); j++)
                {
                for (i = 0; i < fr [0]; i++)
                        {
                        fread (&num[0], samplen[0], 1, fp[0]);
                        *(ptr[0] + i) +=  atof (num);
                        }
                }
        if (j > 0)
                {
                number = (double) j;
                for (i = 0; i < fr [0]; i++)
                        {
                        *(ptr[0] + i) /=  number;
                        }
                }
        frame = fr [0];

        for (i = 0; i < frame; i++)
                {
                number = *(ptr[0] + i);
                if (fabs(number) > amplifie[0])
                        {
                        amplifie[0] = fabs(number);
                        }
                }
         if (amplifie[0] > 0.0)
                {
                amplifie[0] = 1.0 / amplifie[0];
```

```
                        }

                maxpage = 1;
                page = 0;
                }
        }
}

if (fmt && vgl)     /* Spektren mitteln, normieren und in Speicher schreiben */
        {
        for (x = 0; x <= 1; x++)
                {
                rewind (fp[x]);
                fread (&num[0], samplen[x], 1, fp[x]);

                for (j = 0; j < (int) (sampanz[x] / fr [x]); j++)
                        {
                        for (i = 0; i < fr [x]; i++)
                                {
                                fread (&num[0], samplen[x], 1, fp[x]);
                                *(ptr[x] + i) +=  atof (num);
                                }
                        }
                number = (double) (j+1);
                for (i = 0; i < fr [x]; i++)
                        {
                        *(ptr[x] + i) /=  number;
                        }
                }

        if (fr [0] > fr [1])   /* Spektren auf Länge normieren */
                {
                step = fr[0] / fr[1];
                for (i = 0; i < fr [0]; i += step)
                        {
                        number = 0.0;
                        for (j = 0; j < step; j++)
                                {
                                number += *(ptr[0] + i + j);
                                }
                        number /= step;
                        *(ptr[0] + (int) (i / step)) = number;
                        }
                frame = fr [1];
                }
        else
                {
                step = fr[1] / fr[0];
                for (i = 0; i < fr [1]; i += step)
                        {
                        number = 0.0;
                        for (j = 0; j < step; j++)
                                {
```

```
                                        number += *(ptr[1] + i + j);
                                        }
                                number /= step;
                                *(ptr[1] + (int) (i / step)) = number;
                                }
                        frame = fr [0];
                        }

                for (x = 0; x <= 1; x++)
                        {
                        for (i = 0; i < frame; i++)
                                {
                                number = *(ptr[x] + i);
                                if (fabs(number) > amplifie[x])
                                        {
                                        amplifie[x] = fabs(number);
                                        }
                                }
                        if (amplifie[x] > 0.0)
                                {
                                amplifie[x] = 1.0 / amplifie[x];
                                }
                        }

                maxpage = 1;
                page = 0;
                } /* if (fmt && vgl) */
        else if (!fmt)
                {
                maxpage = (int) ((sampanz[0] + offset [0]) / frame);
                if (((sampanz[0] + offset[0]) % frame) != 0) maxpage++;
                if (vgl)
                        {
                        page    = (int) ((sampanz[1] + offset[1]) / frame);
                        if ((sampanz[1] + offset[1]) % frame != 0) page++;
                        if (page > maxpage) maxpage = page;
                        }
                }
        page   = 0;
        sam    = frame / 512;
        spnorm = 511.0 / log10 (frame);
        if (sam < 1)
                {
                sam = 1;
                }

        busy (FALSE);
        setactivepage (1);
        setviewport (0, 0, xgmax, ygmax, 1);
        clearviewport ();
        setvisualpage (1);

        while (outer)
```

```
                        {
                        drawkoord (mode);
                        if (!fmt)
                                {
loadpage (fp[0], ptr[0], sampanz [0], offset [0], page, frame, samplen[0], &fmittel[0], &fvar[0],
&amplifie[0], mode);
                                }
                        setcolor (LIGHTCYAN);
drawpage  (ptr[0], frame, page, sam, ordinate[0], offset[0], amplifie[0], spnorm, mode);
                        if (vgl)
                                {
                        if (!fmt)
                                {
loadpage (fp[1], ptr[1], sampanz [1], offset [1], page, frame, samplen[1], &fmittel[1], &fvar[1],
&amplifie[1], mode);
                                }
                        setcolor (LIGHTMAGENTA);
drawpage  (ptr[1], frame, page, sam, ordinate [1], offset[1], amplifie[1], spnorm, mode);
                                }

                        setlinestyle (DOTTED_LINE,0,NORM_WIDTH);
                        setwritemode (XOR_PUT);

                        switch (mode)
                                {
                                case 0:
                                setcolor (WHITE);
                                outtextxy (  0, 260, "Filename:");
                                outtextxy (120, 260, "Samplelänge [ms]:");
                                if (!qdisp)
                                        {
                                        outtextxy (290, 260, "Mittelwert (Gesamt):");
                                        }
                                outtextxy (480, 260, "Mittelwert (Frame):");
                                outtextxy (  0, 300, "Zeit [ms]:");
                                outtextxy (120, 300, "Wert:");
                                if (!qdisp)
                                        {
                                        outtextxy (290, 300, "Sigma (Gesamt):");
                                        }
                                outtextxy (480, 300, "Sigma (Frame):");
                                setcolor (LIGHTCYAN);
                                outtextxy (  0, 270, filename1);
                                sprintf (num, "%.3f", time[0]);
                                outtextxy (120, 270, num);
                                if (!qdisp)
                                        {
                                        sprintf (num, "%.3f", gmittel[0]);
                                        outtextxy (290, 270, num);
                                        }
                                sprintf (num, "%.3f", fmittel[0]);
                                outtextxy (480, 270, num);
                                if (!qdisp)
```

```
                {
                sprintf (num, "%.3f", sqrt (gvar[0]));
                outtextxy (290, 310, num);
                }
        sprintf (num, "%.3f", sqrt (fvar[0]));
        outtextxy (480, 310, num);
        if (vgl)
                {
                setcolor (LIGHTMAGENTA);
                outtextxy ( 0, 280, filename2);
                sprintf (num, "%.3f", time[1]);
                outtextxy (120, 280, num);
                if (!qdisp)
                        {
                        sprintf (num, "%.3f", gmittel[1]);
                        outtextxy (290, 280, num);
                        }
                sprintf (num, "%.3f", fmittel[1]);
                outtextxy (480, 280, num);
                if (!qdisp)
                        {
                        sprintf (num, "%.3f", sqrt (gvar[1]));
                        outtextxy (290, 320, num);
                        }
                sprintf (num, "%.3f", sqrt (fvar[1]));
                outtextxy (480, 320, num);
                }
        break;

        case 1:
        setcolor (WHITE);
        outtextxy (  0, 260, "Filename:");
        outtextxy (  0, 300, "Zeit [ms]:");
        outtextxy (120, 300, "Wert:");
        setcolor (LIGHTCYAN);
        outtextxy (  0, 270, filename1);
        if (vgl)
                {
                setcolor (LIGHTMAGENTA);
                outtextxy ( 0, 280, filename2);
                }
        break;

        case 2:
        setcolor (WHITE);
        outtextxy (  0, 260, "Filename:");
        outtextxy (120, 260, "Grenzfrequenz [Hz]:");
        outtextxy (  0, 300, "Frequenz [Hz]:");
        outtextxy (120, 300, "Wert:");
        setcolor (LIGHTCYAN);
        outtextxy (  0, 270, filename1);
        if (vgl)
                {
```

```
                setcolor (LIGHTMAGENTA);
                outtextxy (  0, 280, filename2);
                }
        setcolor (YELLOW);
        sprintf (num, "%.1f", freq);
        outtextxy (120, 270, num);
        break;

        default:
        break;
        }
setcolor (WHITE);
outtextxy (0, 340, CURSORANYKEY);

if (mode == 2)         /* Startposition des Cursors */
        {
        i = 0;
        x = 0;
        }
else
        {
        i = frame / 2;
        x = i / sam;
        }

xmove = 0;
setcolor (YELLOW);
line (60 + x, 0, 60 + x, 250);
inner = TRUE;

while (inner)
        {
        setcolor (YELLOW);
        line (60 + x, 0, 60 + x, 250);
        setviewport (68 + x, 245, 124 + x, 255, 1);
        clearviewport ();
        setviewport (0, 0, xgmax, ygmax, 1);
        i += xmove;
        if (i <  0)    i = 0;
        if (i >= frame) i = frame-1;

        x  = i / sam;

        if ((mode == 2) && (i > 0))
                {
                x = (int) (log10 ((double) i) * spnorm + 1.0);
                }

        line (60 + x, 0, 60 + x, 250);

                 wert[0] = *(ptr[0] + i);
        if (vgl) wert[1] = *(ptr[1] + i);
```

```
                        setviewport (  0, 310, 280, 330, 1);
                        clearviewport ();
                        setviewport (0, 0, xgmax, ygmax, 1);

                        switch (mode)
                                {
                                case 0:
                number = (((double) i +  ((double) frame * (double) page)) / (fg * 2.0)) * 1000.0;
                sprintf (num, "%.3f", number);
                setcolor (YELLOW);
                outtextxy (  0, 310, num);
                sprintf (num, "%13.7e",wert[0]);
                setcolor (LIGHTCYAN);
                outtextxy (120, 310, num);
                if (vgl)
                        {
                        sprintf (num, "%13.7e",wert[1]);
                        setcolor (LIGHTMAGENTA);
                        outtextxy (120, 320, num);

                        if (wert [0] != 0.0)
                                {
                if ((fabs (wert [1] / wert [0]) < 99999.9) && (fabs (wert [1] / wert [0]) > 0.1))
                                        {
                                        sprintf (num, "%.1f", fabs (wert [1] / wert [0]));
                                        }
                                else
                                        {
                                        sprintf (num, "%.2g", fabs (wert [1] / wert [0]));
                                        }
                                }
                        else
                                {
                                strcpy (num, "EEEEE.E");
                                }
                        setcolor (YELLOW);
                        outtextxy (68 + x, 245, num);
                        }
                                break;

                                case 1:
number = 1000.0 * ((double) (i+1) + (double) ((double) frame * (double) page) - (double) ordinate [0] -
(double) offset [0]) / (fg * 2.0);
                                sprintf (num, "%.3f", number);
                                setcolor (YELLOW);
                                outtextxy (  0, 310, num);
                                sprintf (num, "%13.7e",wert[0]);
                                setcolor (LIGHTCYAN);
                                outtextxy (120, 310, num);
                                if (vgl)
                                        {
                                        sprintf (num, "%13.7e",wert[1]);
                                        setcolor (LIGHTMAGENTA);
```

```
                outtextxy (120, 320, num);
                }
        break;

        case 2:
        number = fg /  (double) frame * (double) (i);
        sprintf (num, "%.3f", number);
        setcolor (YELLOW);
        outtextxy (  0, 310, num);
        sprintf (num, "%13.7e",wert[0]);
        setcolor (LIGHTCYAN);
        outtextxy (120, 310, num);
        if (vgl)
                {
                sprintf (num, "%13.7e",wert[1]);
                setcolor (LIGHTMAGENTA);
                outtextxy (120, 320, num);
                }
        break;

        default:
        break;
        }
setcolor (YELLOW);
strcpy (outstr, "Seite: ");
sprintf (num, "%d", page + 1);
strcat (outstr, num);
strcat (outstr, "/");
sprintf (num, "%d", maxpage);
strcat (outstr, num);
outtextxy (  0, 320, outstr);

switch (getch ())
        {
        case 0x00:
        switch (getch ())
                {
                case 'K':       /* Cursor Left */
                xmove = -1;
                break;

                case 'M':       /* Cursor Right */
                xmove =  1;
                break;

                case 'P':       /* Cursor Down */
                xmove = -50;
                break;

                case 'H':       /* Cursor Up */
                xmove =  50;
                break;
```

```
case 'Q':      /* Bild Down */
page--;
if (page < 0)
        {
        page = 0;
        }
else
        {
        inner = FALSE;
        }
break;

case 'I':      /* Bild Up */
page++;
if (page >= maxpage)
        {
        page = maxpage-1;
        }
else
        {
        inner = FALSE;
        }
break;

case 'O':      /* Ende */
if (page == maxpage-1)
        {
        break;
        }
else
        {
        page = maxpage-1;
        inner = FALSE;
        }
break;

case 'G':      /* Pos 1 */
if (page == 0)
        {
        break;
        }
else
        {
        page = 0;
        inner = FALSE;
        }
break;

default:
inner = FALSE;
outer = FALSE;
break;
}
```

```
                                        break;

                                        case 0x67:     /* G */
                                        savegraph ((UWORD) 329);
                                        break;

                                        default:       /* Anzeige verlassen */
                                        inner = FALSE;
                                        outer = FALSE;
                                        break;
                                        }
                                } /* while (inner) */
                        } /* while (outer) */

          farfree (ptr[0]);
          fcloseall ();
          if (vgl)
                        {
                        farfree   (ptr[1]);
                        }
          setactivepage (0);
          setvisualpage (0);
return (TRUE);
}

/*********************************************************************/
/* Koordinatensystem zeichnen                                        */
/*********************************************************************/

void drawkoord (BOOL mode)
{
          setviewport (0, 0, xgmax, ygmax, 1);
          clearviewport ();
          setcolor (WHITE);
          setlinestyle (SOLID_LINE,0,NORM_WIDTH);
          setwritemode (COPY_PUT);

          line (60,   0,  60, 250);   /* Vertikal */
          line (60, 125, 600, 125);   /* Horizontal */

          line (60 -5,     5,  60,   0);      line (60 +5,     5,  60,   0);
          line (60 -5, 250-5,  60, 250);      line (60 +5, 250-5,  60, 250);
          line (600-5, 125-5, 600, 125);      line (600-5, 125+5, 600, 125);

          line (60  -5, 125-115, 60  +5, 125-115);
          line (60  -5, 125+115, 60  +5, 125+115);
          line (60+512, 125  -5, 60+512, 125  +5);

          switch (mode)
                        {
                        case 0:
                        case 1:
                        outtextxy (600-10, 125+10, "t");
```

```
                        outtextxy (60-30,      5, "+Y");
                        outtextxy (60-30, 250-13, "-Y");
                        break;

                        case 2:
                        outtextxy (600-40, 125+10, "Log(f)");
                        outtextxy ( 60-55,      5, "Log(Y)");
                        break;

                        default:
                        break;
                        }
return;
}

/*********************************************************************/
/* Eine Seite der Funktion zeichnen                                  */
/*********************************************************************/

void drawpage (double far *ptr, int frame, int page, int sam, long ordinate, long offset, double amplifie,
double spnorm, BOOL mode)
{
double   wert;
int      i, x;


         moveto (60, 125);
         for ( i = 0; i < frame; i++)        /* Daten aus File lesen */
                        {
                        wert     = *(ptr + i);
                        wert    *= amplifie;           /* -1  <->  +1 */
                        if (mode == 2)         /* Spektrum */
                                {
                                wert *= 10;
                                wert += 1;
                                wert  = log10 (wert);
                                wert /= 1.1;
                                }
                        wert    *= -1;
                        wert    +=  1;
                        if      (wert > 2.0)
                                {
                                wert = 2.0;
                                }
                        else if (wert < 0.0)
                                {
                                wert = 0.0;
                          }
                        wert *= 115.0;
                        wert += 10.0;

                        x = i / sam;
```

```
                        if ((mode == 2) && (i > 0))
                                {
                                x = (int) (log10 ((double) i) * spnorm + 1.0);
                                }

                        if (
                              ( ((long) i + (long) frame * (long) page) == (ordinate + offset - 1) )
                           ||
                              (i == 0)
                           )
                                {
                                if (mode == 1)
                                        {
                                        int color;
                                        color = getcolor ();
                                        setcolor (WHITE);
                                        line (60 + x, 0, 60 + x, 250);
                                        setcolor (color);
                                        }
                                line (60 + x - 5, (int) wert - 5, 60 + x,(int) wert);
                                line (60 + x - 5, (int) wert + 5, 60 + x,(int) wert);
                                line (60 + x + 5, (int) wert - 5, 60 + x,(int) wert);
                                line (60 + x + 5, (int) wert + 5, 60 + x,(int) wert);
                                moveto (60 + x,   (int) wert);
                                }
                        else
                                {
                                lineto (x + 60, (int) wert);
                                }
                        }
return;
}

/*********************************************************************/
/* Eine Seite der Funktion laden                                     */
/*********************************************************************/

void     loadpage (FILE *fp, double far *ptr, long sampanz, long offset, int page, int frame, int samplen,
double *mittel, double *var, double *amplifie, BOOL mode)

{
char     num [64]      = "";
double   wert          = 0.0;
long     pos,
         headcorr      = 0L;          /* Korrektur für Header */
int      i;
BOOL     first         = TRUE; /* FALSE wenn erster Wert gelesen */

         for (i = 0; i < frame; i++)
                        {
                        *(ptr + i) = 0.0;
                        }
```

```
        *mittel = 0.0;
        *var    = 0.0;
        if (qdisp)
                    {
                    *amplifie = 0.0;
                    }

        if (mode != 0)              /* AKF, KKF & DFT */
                    {
                    headcorr = 1L;
                    }

        for (i = 0; i < frame; i++)
                    {
                    pos = (long) ((long) page * (long) frame) - offset + (long) i;
                    if (pos <      0L)     continue;
                    if (pos > sampanz)      break;

               if (first)
                            {
fseek (fp, (long) ((((long) page * (long) frame) - offset + headcorr + (long) i) * (long) samplen),
SEEK_SET);
                            first = FALSE;
                            }

                    if (!fread (&num[0], samplen, 1, fp))
                            {
                            break;
                            }
                    else
                            {
                            wert = atof (&num[0]);
                            }
                    *mittel += wert;
                    *var    += wert * wert;
                    *(ptr + i) = wert;
                    if (qdisp)
                            {
                            if (fabs(wert) > *amplifie)
                                    {
                                    *amplifie = fabs(wert);
                                    }
                            }
                    }

        if ((qdisp) && (*amplifie > 0.0))
                            {
                            *amplifie = 1.0 / *amplifie;
                            }

        *mittel /= (double) (i+1);

        if (i != 0)
```

```
                        {
                        *var /= (double) i;
                        *var -= *mittel * *mittel;
                        }
          else
                        {
                        *var = 0.0;
                        }
return;
}

/*********************************************************************/
/* Liefert die Zeichenanzahl einer Zahl im File auf das "fp" zeigt      */
/*********************************************************************/

int       getsamplength (FILE *fp)
{
int       samplength           = -1;
double    wert                 = 0.0;
char      c                    = 0x00,
          num [80]             = "",
          *endptr              = NULL;
BOOL      end                  = FALSE;

          rewind (fp);

          while (((!end) || (c == 0x0D) || (c == 0x0A)) && (samplength < 64))
                        {
                        c = fgetc (fp);
                        samplength++;
                        if ((c == 0x0D) || (c == 0x0A))
                                {
                                end = TRUE;
                                }
                        }

          rewind (fp);
          if (samplength == 64)               /* Ungültiges File */
                        {
                        fclose (fp);
                        return (0);
                        }

          fread (&num[0], samplength, 1, fp);
          rewind (fp);
          wert = strtod (&num[0], &endptr);
          if ((wert == 0.0) && ((*endptr != 0x0D)  && (*endptr != 0x20)))
                        {
                        fclose (fp);
                        return (0);
                        }

return (samplength);
```

```
}

/*********************************************************************/
/* Ermittelt: Fileheader, Anzahl der Werte, Wertlänge (ASCII),          */
/*            Verstärkungsfaktor und Mittelwert eines Files             */
/*********************************************************************/

BOOL     getdat (FILE *fp,  long *header, long *sampanz, int *samplen,
         double *amplifie, double *mittel, double *var, BOOL mode)
{
char            *endptr,
               num [64]       = "";
long           i,
               filelength;
double         wert           = 0.0;
BOOL           error          = FALSE;

         *samplen = getsamplength (fp);
         if (*samplen == 0)
               {
               return (FALSE);
               }

         switch (mode)
               {
               case 0:                        /* Zeitfunktion */
               fseek (fp,0L,SEEK_END);
               filelength = ftell (fp);
               *sampanz = (long) (filelength / *samplen);
               rewind (fp);
               break;

               case 1:                        /* AKF oder KKF */
               case 2:                          /* DFT */
               fread (&num[0], *samplen, 1, fp);
               *header = (int) strtol (&num[0], &endptr, 10);
               if (*endptr != ' ')
                      {
                      fclose (fp);
                      return (FALSE);
                      }
               fseek (fp,0L,SEEK_END);
               filelength = ftell (fp);
               *sampanz = (long) ((filelength - *samplen) / *samplen);
               fseek (fp, *samplen, SEEK_SET);
               break;

               default:
               break;
               }

         if (*sampanz < 1L)
               {
```

```
        fclose (fp);
        return (FALSE);
        }

    if (qdisp)
        {
        return (TRUE);
        }

    for ( i = 0; i < *sampanz; i++)             /* Daten auslesen */
        {
        if (!fread (&num[0], *samplen, 1, fp))
                {
                break;
                }
        wert = strtod (&num[0], &endptr);
        if ((wert == 0.0) && ((*endptr != 0x0D)  && (*endptr != 0x20)))
                {
                error = TRUE;
                break;
                }
        *mittel += wert;
        *var    += wert * wert;
        if (fabs(wert) > *amplifie)
                {
                *amplifie = fabs(wert);
                }
        }

    if (*amplifie > 0.0)
        {
        *amplifie = 1.0 / *amplifie;
        }
    else
        {
        error = TRUE;
        }

    if (error)
        {
        return (FALSE);
        }
    *mittel /= (double) *sampanz;
    if (i != 0)
        {
        *var /= (double) i;
        *var -= *mittel * *mittel;
        }
    else
        {
        *var = 0.0;
        }
```

```
return (TRUE);
}

/*********************************************************************/
/* Histogramm (Logarithmisch oder Linear)                            */
/*********************************************************************/

BOOL     histogramm (char *filename)
{
void                  drawhistokoord ();
FILE                  *fp;
char                  *endptr,
                      *filenameptr,
                      num [64]   = "";
double                wert      = 0.0,
                      mittel    = 0.0,
                      var       = 0.0,
                      max,
                      x, y,
                      gstep,
                      histo [196];                  /* Maximale Anzahl der Histogrammstufen */
double   static       showmaxamp = 2.0;             /* Wertebereich -2...+2 */
long                  filelength,
                      sampanz,
                      i;
int                   samplength,
                      ret;
int static            histoanz = 196;
unsigned int          j;
unsigned int static     step,
                      xkoord, ykoord,               /* Koordinatenlänge */
                      xnul,   ynul;                 /* Koordinatenursprung */
int      static       color,                        /* Zeichenfarbe */
                      vergl,                        /* Verteilungsfunktion */
                      mode;                         /* Modus Log/Lin */
BOOL     static       used  = FALSE;                /* Neu initialisieren */
BOOL                  error = FALSE,
                      ok    = TRUE;

         fp = fopen (filename,"rb");
         if (fp == NULL)
               {
               used = FALSE;
               beep ();
               showerror (1);
               return (FALSE);
               }
         fseek (fp,0L,SEEK_END);
         filelength = ftell (fp);
         samplength = getsamplength (fp);
         if (samplength == 0)
               {
               used = FALSE;
```

```
                beep ();
                showerror (3);
                return (FALSE);
                }
        sampanz    = (long) (filelength / samplength);

        if (!used)
                {
                ret = option (OPTX, YREL, 2, "Histogramm:",
                "Linear",
                "Logarithmisch");
                if (ret == 0)
                        {
                        used = FALSE;
                        fclose (fp);
                        return (FALSE);
                        }
                mode = ret;
                ret = option (OPTX, YREL, 5, "Auflösung:",
                "Sehr Hoch          196",
                "Hoch               147",
                "Mittel              98",
                "Niedrig             49",
                "Sehr Niedrig        19");
                if (ret == 0)
                        {
                        used = FALSE;
                        fclose (fp);
                        return (FALSE);
                        }
                switch (ret)
                        {
                        case 1: histoanz = 196;        break;
                        case 2: histoanz = 147;        break;
                        case 3: histoanz =  98;        break;
                        case 4: histoanz =  49;        break;
                        case 5: histoanz =  19;        break;
                        default:
                        break;
                        }
ret = option (OPTX, YREL, 3, "Vergleich:", "Keine Vorgaben", "Laplaceverteilung", "Gaußverteilung");
                if (ret == 0)
                        {
                        used = FALSE;
                        fclose (fp);
                        return (FALSE);
                        }
                vergl = ret;
                ret = inputnumber ("Wertebereich:", &showmaxamp);
                showmaxamp = fabs (showmaxamp);
                if (showmaxamp == 0.0)
                        {
                        switch (vergl)                /* Automatisches Skalieren */
```

```
                        {
                        case 2:                 /* Laplace */
                        if (mode == 2)          /* Log */
                                {
                                showmaxamp = fabs (log (0.1)    / verteilung.ld);
                                }
                        else                    /* Lin */
                                {
                                showmaxamp = fabs (log (0.0035) / verteilung.ld);
                                }
                        break;

                        case 3:                 /* Gauss */
                        if (mode == 2)          /* Log */
                                {
                        showmaxamp = sqrt (fabs (2.0 * pow (verteilung.sg, 2.0) * log (0.1)   ));
                                }
                        else                            /* Lin */
                                {
                        showmaxamp = sqrt (fabs (2.0 * pow (verteilung.sg, 2.0) * log (0.0035)));
                                }
                        break;

                        default:
                        if (verteilung.a > 0.0)
                                {
                                showmaxamp = verteilung.a;
                                }
                        else
                                {
                                showmaxamp = 2.0;
                                }
                        break;
                        }
                }
        if (ret == 50)
                {
                used = FALSE;
                fclose (fp);
                return (FALSE);
                }
        }

setactivepage (1);
if (!used)
        {
        setviewport (0, 0, xgmax, ygmax, 1);
        clearviewport ();
        }
else
        {
        setviewport (0, ygmax - YSPACE + 70, xgmax, ygmax, 1);
        clearviewport ();
```

```
            setviewport (0, 0, xgmax, ygmax, 1);
            }

      setvisualpage (1);
      setlinestyle (SOLID_LINE,0,NORM_WIDTH);
      setwritemode (COPY_PUT);
      if (!used)
            {
            color = WHITE;
            setcolor (color);
            drawhistokoord (&step, &xkoord, &ykoord, &xnul, &ynul, histoanz, mode, showmaxamp);

            if (vergl > 1)
                    {
                    setcolor (RED);
                    moveto (xnul,ynul);
                    x = -1.0 * showmaxamp;
                    gstep = (double) xkoord / 589.0;

                    for (j = 0; j < 589; j++)
                            {
                            x += (showmaxamp * 2.0) / 589.0;

                            switch (vergl)         /* Verleichsfunktion zeichnen */
                                    {
                                    case 2:        /* Laplace */
                                    y = exp (-1.0 * verteilung.ld * fabs (x));
                                    break;

                                    case 3:        /* Gauss */
                                    y = exp (-1.0 * pow (x, 2.0) / (2.0 * pow (verteilung.sg, 2.0)));
                                    break;
                                    }
                            if (mode == 2)         /* Log */
                                    {
                                    y *= 10;
                                    if (y < 1.0)
                                            {
                                            y = 0.0;
                                            }
                                    else
                                            {
                                            y = log10 (y);
                                            }
                                    }
                            lineto ((int) (xnul + (double) j * gstep), ynul - y * (ynul-10));
                            }
                    lineto ((int) (xnul + (double) (j-1) * gstep), ynul);
                    }
            }
      else
            {
            color--;
```

```
        }

    setcolor (color);

    rewind (fp);

    memset (&histo[0], 0x00, sizeof(double) * histoanz);
    mittel = var = 0.0;

    for (i = 0L; i < sampanz; i++)
        {
        if(!fread (&num[0],samplength,1,fp))
                {
                break;
                }
        wert = 0.0;
        wert = strtod (&num[0],&endptr);
        if ((wert == 0.0) && ((*endptr != 0x0D)  && (*endptr != 0x20)))
                {
                error = TRUE;
                break;
                }
        mittel += wert;
        var    += wert * wert;
        wert   += showmaxamp;         /* Normieren */
        wert   /= showmaxamp * 2.001;
        j = (int) floor (wert * histoanz);
        if (j < 0)            continue;
        if (j >= histoanz)      continue;
        (histo[j])++;
        }
    fclose (fp);

    mittel /= (double) sampanz;
    if (i != 0L)
        {
        var /= (double) i;
        var -= mittel * mittel;
        }
    else
        {
        var = 0.0;
        }

    max = 1.0;                  /* Werte auf 1 normieren */
    for (j = 0; j < histoanz; j++)
        {
        if (histo[j] > max)
                {
                max = histo[j];
                }
        }
```

```
for (j = 0; j < histoanz; j++)
      {
      histo[j] /= max;
      }

if (mode == 2)
      {
      for (j = 0; j < histoanz; j++)
             {
             histo[j] *= 10;
             if (histo[j] < 1.0)
                    {
                    histo[j] = 0.0;
                    }
             else
                    {
                    histo[j]  = log10 (histo[j]);
                    }
             }
      }

moveto (xnul,ynul);
for ( j = 0; j < histoanz; j++)
      {
      lineto (xnul + j * step, ynul - histo[j] * (ynul-10));
      }
lineto (xnul + (j-1) * step, ynul);

setcolor (WHITE);
if (error)
      {
      beep ();
      color++;
      showerror (3);
      outtextxy (0, ygmax - YSPACE + 80, NODATAFILE);
      }
else
      {
      int xpos, ypos;

      outtextxy (0, ygmax - YSPACE + 30, "Filename:");
      if      ((color >= LIGHTRED) && (color <= WHITE))
             {
             xpos = 80;
             ypos = ygmax - YSPACE + 30 + (3 - color + LIGHTRED) * 10;
             }
      else if ((color >= DARKGRAY) && (color <= LIGHTCYAN))
             {
             xpos = 200;
             ypos = ygmax - YSPACE + 30 + (3 - color + DARKGRAY) * 10;
             }
      else if ((color >= RED) && (color <= LIGHTGRAY))
             {
```

```
                xpos = 320;
                ypos = ygmax - YSPACE + 30 + (3 - color + RED) * 10;
                }
        else
                {
                xpos = 440;
                ypos = ygmax - YSPACE + 30 + (3 - color + BLACK) * 10;
                }
        setcolor (color);
        outtextxy (xpos, ypos, filename);

        outtextxy (  0, ygmax - YSPACE + 75, "Mittelwert:");
        sprintf (num, "%.3f", mittel);
        outtextxy ( 96, ygmax - YSPACE + 75, num);

        outtextxy (200, ygmax - YSPACE + 75, "Sigma:");
        sprintf (num, "%.3f", sqrt (var));
        outtextxy (256, ygmax - YSPACE + 75, num);
        }

setcolor (WHITE);
outtextxy (0, ygmax - YSPACE + 90, HISTOANYKEY);

while (ok)
        {
        switch (getch ())
                {
                case 0x00:
                switch (getch ())
                        {
                        default:
                        break;
                        }
                ok = FALSE;
                break;

                case 0x67:              /* G */
                savegraph ((UWORD) (ygmax - YSPACE + 70));
                break;

                case 0x0D:              /* Return */
                ok = FALSE;
                if (color <= BLUE)
                        {
                        break;
                        }
                setactivepage (0);
                setvisualpage (0);

                filenameptr = getfilename ();
                if (*filenameptr != 0x00)
                        {
                        used = TRUE;
```

```
                                histogramm (filenameptr);
                                }
                        ok = FALSE;
                        break;

                        default:
                        ok = FALSE;
                        break;
                        }
                }

          setactivepage (0);
          setvisualpage (0);
          used = FALSE;

return (TRUE);
}


/*********************************************************************/

void drawhistokoord (int *step, int *xkoord, int *ykoord, int *xnul, int *ynul, int histoanz, int mode,
double xmax)
{
char      outstr [20],
          num    [20];

          *step   = (xgmax - XSPACE) / histoanz;
          *xkoord = (histoanz - 1) * (*step);
          *ykoord = (ygmax - YSPACE);
          *xnul   = XSPACE / 2;               /* Koordinatenursprung */
          *ynul   = *ykoord;

          line (*xnul, *ynul, *xnul + *xkoord, *ynul);
          line (*xnul + *xkoord - 5, *ynul - 5, *xnul + *xkoord, *ynul);
          line (*xnul + *xkoord - 5, *ynul + 5, *xnul + *xkoord, *ynul);

          if (mode == 2)                            /* Log */
                {
                outtextxy (*xnul + (*xkoord / 2) + 10, 2, "Log(Y)");
                }
          else                                /* Linear */
                {
                outtextxy (*xnul + (*xkoord / 2) + 10, 2, "Y");
                }
          line (*xnul + (*xkoord / 2) - 1, 0, *xnul + (*xkoord / 2) - 1, *ykoord);
          line (*xnul + 5, *ynul - 5, *xnul, *ynul);
          line (*xnul + 5, *ynul + 5, *xnul, *ynul);
          line (*xnul + (*xkoord / 2) - 6, 5, *xnul + (*xkoord / 2) - 1, 0);
          line (*xnul + (*xkoord / 2) + 4, 5, *xnul + (*xkoord / 2) - 1, 0);
          line (*xnul + (*xkoord / 2) - 6, (*ynul) / 2, *xnul + (*xkoord / 2) + 4, (*ynul) / 2);
          line (*xnul + (*xkoord / 4) - 2, *ynul - 5, *xnul + (*xkoord / 4) - 2, *ynul + 5);
          line (*xnul + (*xkoord * 3 / 4), *ynul - 5, *xnul + (*xkoord * 3 / 4), *ynul + 5);
```

```
        sprintf (num, "%.3f", xmax);
        strcpy (outstr, "+");
        strcat (outstr, num);
        outtextxy (*xnul +  *xkoord                - 8*strlen (outstr), *ykoord + 10, outstr);
        strcpy (outstr, "-");
        strcat (outstr, num);
        outtextxy (*xnul                                            , *ykoord + 10, outstr);
        sprintf (num, "%.3f", xmax / 2.0);
        strcpy (outstr, "+");
        strcat (outstr, num);
        outtextxy (*xnul + (*xkoord * 3 / 4)  -  2 - 4*strlen (outstr), *ykoord + 10, outstr);
        strcpy (outstr, "-");
        strcat (outstr, num);
        outtextxy (*xnul + (*xkoord     / 4)  -  4 - 4*strlen (outstr), *ykoord + 10, outstr);
        outtextxy (*xnul + (*xkoord     / 2)  -  4                    , *ykoord + 10, "0");

return;
}

/*********************************************************************/
/* Eingabe eines Filenamens für zu erzeugende Datei                  */
/*********************************************************************/

int     inputfilename (char *header, char *suff, char *file, char *out)
{
int     ret;

        setviewport (0, 6*10, 21*8, 9*10, 1);
        setcolor (BLUE);
        rectangle (0, 0, 21*8, 3*10);
        setcolor (YELLOW);
        outtextxy (3*8, 1*10, header);
        strcpy (out,file);
        if (!suffix (out,suff))
              {
              memset (out,0x00,13);
              }
        ret = input(3, 8, 12, out, 1);
        if (ret == 20)
              {
              savegraph ((UWORD) 349);
              ret = 50;
              }
        if (ret == 0)
              {
              strcpy (out,TEMPDATEI);
              }
        setviewport (0, 6*10, 21*8, 9*10, 1);
        clearviewport ();

return (ret);
}
```

Anhang C: Listing des Quellcodes von SPV (SPV.C)

```
/*********************************************************************/
/* Eingabe einer Zahl                                                */
/*********************************************************************/

int      inputnumber (char *header, double *zahl)
{
char     num [20];
int      ret;

         setviewport (0, 6*10, 26*8, 9*10, 1);
         setcolor (BLUE);
         rectangle (0, 0, 26*8, 3*10);
         setcolor (YELLOW);
         outtextxy (3*8, 1*10, header);
         sprintf (num, "%.3f", *zahl);
         ret = input(3, 8, 18, num, 1);
         if (ret == 20)
               {
               savegraph ((UWORD) 349);
               ret = 50;
               }
         if (ret == 0) *zahl = 0.0;
         else                 *zahl = atof (num);

         setviewport (0, 6*10, 26*8, 9*10, 1);
         clearviewport ();

return (ret);
}

/*********************************************************************/
/* Prozentanzeige                                                    */
/*      mode: 0 - Prozent anzeigen                                   */
/*            1 - Anzeige initialisieren                             */
/*            2 - Anzeige löschen                                    */
/*********************************************************************/

void     showpercent (int percent, BOOL mode)
{
char     num [10];

         if    (mode == 1)
               {
               setviewport (0*8, 12*10, 20*8, 14*10, 1);
               setcolor (BLUE);
               rectangle (0, 0, 20*8, 2*10);
               setcolor (YELLOW);
               outtextxy (3*8, 1*10, "Fertig:");
               setviewport (12*8, 12*10+3, 19*8, 14*10-3, 1);
               }
         else if (mode == 2)
               {
```

```
                setviewport (0*8, 12*10, 20*8, 14*10, 1);
                clearviewport ();
                }
          else
                {
                sprintf (num, "%d \%", percent);
                clearviewport ();
                outtextxy (0, 10-3, num);
                }
return;
}

/*********************************************************************/
/* Statistische Daten anzeigen                                       */
/*********************************************************************/

BOOL      showstats (double invar, double inmittel, double outvar, double outmittel)
{
char      num [20];
BOOL      ret;

          setviewport (0*8, 12*10, 27*8, 21*10, 1);
          setcolor (BLUE);
          rectangle (0, 0, 27*8, 9*10);
          setcolor (YELLOW);
          outtextxy (3*8, 1*10, "Mittelwert:");
          outtextxy (3*8, 2*10, "Eingang:");
          outtextxy (3*8, 3*10, "Ausgang:");

          outtextxy (3*8, 5*10, "Sigma:");
          outtextxy (3*8, 6*10, "Eingang:");
          outtextxy (3*8, 7*10, "Ausgang:");

          sprintf (num, "%.8g", inmittel);
          outtextxy (12*8, 2*10, num);
          sprintf (num, "%.8g", outmittel);
          outtextxy (12*8, 3*10, num);
          sprintf (num, "%.8g", sqrt (invar));
          outtextxy (12*8, 6*10, num);
          sprintf (num, "%.8g", sqrt (outvar));
          outtextxy (12*8, 7*10, num);

          switch (getch ())
                {
                case 0x00:
                getch ();
                ret = 1;
                break;

                case 0x1B:
                ret = 0;
                break;
```

```
                case 0x67:
                savegraph ((UWORD) 349);
                ret = 0;
                break;

                default:
                ret = 1;
                break;
                }
          clearviewport ();
return (ret);
}

/*********************************************************************/
/* Bitte Warten                                                      */
/*********************************************************************/

void      busy (BOOL mode)
{
          if (mode == TRUE)
                {
                setviewport (0*8, 12*10, 19*8, 14*10, 1);
                setcolor (BLUE);
                rectangle (0, 0, 19*8, 2*10);
                setcolor (YELLOW);
                outtextxy (3*8, 1*10, "Bitte Warten");
                }
          else
                {
                setviewport (0*8, 12*10, 19*8, 14*10, 1);
                clearviewport ();
                }

return;
}

/*********************************************************************/
/* An die Zeichenkette, auf die "ptr" zeigt, wird das Suffix in "suff"  */
/* angehängt                                                         */
/*********************************************************************/

BOOL      suffix (char *ptr, char *suff)
{
int       i;
BOOL      suffix = FALSE;

          for (i = 0; i < 13; i++)
                {
                if (suffix)
                        {
                        *(ptr+i) = 0x00;
                        }
                if (*(ptr+i) == '.')
```

```
                        {
                        suffix = TRUE;
                        }
                }
          if (strlen (ptr) < 10)
                {
                strcat (ptr,suff);
                return (TRUE);
                }
return (FALSE);
}

/*********************************************************************/
/* Kennlinie zeichnen                                                */
/*********************************************************************/

void      drawkenn ()
{
char      num   [20];
double    x, y,
          step;
int       i, j;
BOOL      ok = TRUE;

          setactivepage (1);
          setviewport (0, 0, xgmax, ygmax, 1);
          clearviewport ();
          setvisualpage (1);
          setcolor (WHITE);
          setlinestyle (SOLID_LINE, 0, NORM_WIDTH);
          setwritemode (COPY_PUT);

          line (50  ,   0,  50, 250);
          line (50  , 250, 600, 250);
          line (50-5, 5, 50, 0);
          line (50+5, 5, 50, 0);
          line (600-5, 250-5, 600, 250);
          line (600-5, 250+5, 600, 250);

          outtextxy (0,  10-5, "10000");
          outtextxy (0,  50-5, "1000");
          outtextxy (0,  90-5, "100");
          outtextxy (0, 130-5, "10");
          outtextxy (0, 170-5, "1.0");
          outtextxy (0, 210-5, "0.1");
          outtextxy (0, 250-5, "0.01");

          outtextxy ( 50-10, 250+10, "0.0001");
          outtextxy (150-10, 250+10, "0.001");
          outtextxy (250-10, 250+10, "0.01");
          outtextxy (350-10, 250+10, "0.1");
          outtextxy (450-10, 250+10, "1.0");
          outtextxy (550-10, 250+10, "10");
```

```
setcolor (DARKGRAY);

x = 0.0001;                 /* Vertikales Raster */
step = 0.0001;
for (j = 1; j < 6; j++)
      {
      for (i = 1; i < 10; i++)
             {
      x += step;
      line ((int) (50+(log10 (x*10000.0)*100)), 10, (int) (50+(log10 (x*10000.0)*100)), 250-2);
             }
      step *= 10;
      }

x = 0.01;                   /* Horizontales Raster */
step = 0.01;
for (j = 1; j < 7; j++)
      {
      for (i = 1; i < 10; i++)
             {
      x += step;
      line (50+2, (int) (250-(log10 (x*100.0)*40)), 550, (int) (250-(log10 (x*100.0)*40)));
             }
      step *= 10;
      }

setcolor (LIGHTRED);

x = 0.0001;
step = 0.00001;
for (j = 1; j < 6; j++)
      {
      for (i = 1; i < 91; i++)
             {
             x += step;
             y = kennlinie (x);
             if (y < 0.01)
                    {
                    y = 0.01;
                    }
             if ((j == 1) && (i == 1))
                    {
                    moveto (50, (int) (250 - log10 (y*100.0) * 40));
                    }
             lineto ((int) (50+(log10 (x*10000.0)*100)), (int) (250 - log10 (y*100.0)*40));
             }
      step *= 10;
      }

setcolor (WHITE);

outtextxy (13*8, 28*10, "Kennlinienparameter:");
```

```
        outtextxy (13*8, 29*10, "Ya =                d =                q =");
        outtextxy (13*8, 30*10, "Yo =                p =                r =");
        sprintf (num, "%.8g",kennpar.ua);
        outtextxy (18*8,  29*10, num);
        sprintf (num, "%.8g",kennpar.us);
        outtextxy (18*8,  30*10, num);
        sprintf (num, "%.8g",kennpar.d);
        outtextxy (37*8, 29*10, num);
        sprintf (num, "%.8g",kennpar.p);
        outtextxy (37*8, 30*10, num);
        sprintf (num, "%.8g",kennpar.q);
        outtextxy (56*8, 29*10, num);
        sprintf (num, "%.8g",kennpar.r);
        outtextxy (56*8, 30*10, num);

        outtextxy (0, 340, ANYKEY);

        while (ok)
              {
              if (getch () == 0x67)  /* G */
                     {
                     savegraph ((UWORD) 309);
                     }
              else
                     {
                     ok = FALSE;
                     }
              }
        setactivepage (0);
        setvisualpage (0);

return;
}

/*********************************************************************/
/* Zeitbereiche auswählen                                            */
/*      mode: 0 - Autokorrelation                                    */
/*            1 - Kreuzkorrelation                                   */
/*            2 - Files verketten                                    */
/*********************************************************************/

BOOL     timeselect (BOOL mode, char *file1, char *file2,long *min1, long *max1, long *min2, long *max2)
{
char     outstr [40];
double   wert;
long     maxf1, maxf2;
int      pos = 0;
BOOL     ok  = TRUE,
         ret;

         maxf1 = *max1;
         maxf2 = *max2;
         if (mode != 2)
```

```
        {
        if (*max1 > 32767L)    *max1 = 32767L;
        if (*max2 > 32767L)    *max2 = 32767L;
        }

  setviewport (0*8, YREL*10, 49*8, (YREL+27)*10, 1);
  clearviewport ();
  setcolor (BLUE);
  setlinestyle (SOLID_LINE, 0xFFFF, THICK_WIDTH);
  setwritemode (COPY_PUT);
  rectangle (0*8,  0*10, 49*8, 27*10);
  rectangle (1*8,  2*10, 48*8, 26*10);

  rectangle (3*8,  8*10, 24*8, 10*10);
  rectangle (3*8, 20*10, 24*8, 22*10);

  setcolor (YELLOW);

  switch (mode)
        {
        case 0:
        outtextxy (2*8, 1*10, "Autokorrelation:");
        break;

        case 1:
        outtextxy (2*8, 1*10, "Kreuzkorrelation:");
        break;

        case 2:
        outtextxy (2*8, 1*10, "Files verketten:");
        break;
        }

 setcolor (LIGHTGRAY);

  switch (mode)
        {
        case 0:
        outtextxy (3*8,   3*10, "Frame von:");
        outtextxy (14*8,  3*10, file1);

        outtextxy (3*8,  15*10, "über File:");
        outtextxy (14*8, 15*10, file1);
        break;

        default:
        outtextxy (3*8, 3*10, "File:");
        outtextxy (9*8, 3*10, file1);

        outtextxy (3*8, 15*10, "File:");
        outtextxy (9*8, 15*10, file2);
        break;
        }
```

```
outtextxy ( 3*8,  5*10, "von:");
outtextxy ( 3*8, 12*10, "bis:");
outtextxy (26*8,  9*10, "Dauer:");

outtextxy ( 3*8, 17*10, "von:");
outtextxy ( 3*8, 24*10, "bis:");
outtextxy (26*8, 21*10, "Dauer:");

do{
      if (pos == 0)  setcolor (WHITE);
      else           setcolor (DARKGRAY);
      wert = ((double) (*min1) / (fg * 2.0)) * 1000.0;
      sprintf (outstr, "%.3f", wert);
      strcat (outstr, "ms");
      outtextxy ( 8*8, 5*10, outstr);
      sprintf (outstr, "%ld", *min1);
      outtextxy ( 8*8, 6*10, outstr);
      line (3*8+4+ (int) ((*min1)*160/maxf1),  8*10+3,
            3*8+4+ (int) ((*min1)*160/maxf1), 10*10-3);

      if (pos == 1)  setcolor (WHITE);
      else           setcolor (DARKGRAY);
      wert = ((double) (*max1) / (fg * 2.0)) * 1000.0;
      sprintf (outstr, "%.3f", wert);
      strcat (outstr, "ms");
      outtextxy ( 8*8, 12*10, outstr);
      sprintf (outstr, "%ld", *max1);
      outtextxy ( 8*8, 13*10, outstr);
      line (3*8+4+ (int) ((*max1)*160/maxf1),  8*10+3,
            3*8+4+ (int) ((*max1)*160/maxf1), 10*10-3);

      setcolor (DARKGRAY);
      wert = ((double) (*max1 - *min1) / (fg * 2.0)) * 1000.0;
      sprintf (outstr, "%.3f", wert);
      strcat (outstr, "ms");
      outtextxy (33*8, 9*10, outstr);

      if (pos == 2)  setcolor (WHITE);
      else           setcolor (DARKGRAY);
      wert = ((double) (*min2) / (fg * 2.0)) * 1000.0;
      sprintf (outstr, "%.3f", wert);
      strcat (outstr, "ms");
      outtextxy ( 8*8, 17*10, outstr);
      sprintf (outstr, "%ld", *min2);
      outtextxy ( 8*8, 18*10, outstr);
      line (3*8+4+ (int) ((*min2)*160/maxf2), 20*10+3,
            3*8+4+ (int) ((*min2)*160/maxf2), 22*10-3);

      if (pos ==3)   setcolor (WHITE);
      else           setcolor (DARKGRAY);
      wert = ((double) (*max2) / (fg * 2.0)) * 1000.0;
      sprintf (outstr, "%.3f", wert);
```

```
        strcat (outstr, "ms");
        outtextxy ( 8*8, 24*10, outstr);
        sprintf (outstr, "%ld", *max2);
        outtextxy ( 8*8, 25*10, outstr);
        line (3*8+4+ (int) ((*max2)*160/maxf2), 20*10+3,
              3*8+4+ (int) ((*max2)*160/maxf2), 22*10-3);

        setcolor (DARKGRAY);
        wert = ((double) (*max2 - *min2) / (fg * 2.0)) * 1000.0;
        sprintf (outstr, "%.3f", wert);
        strcat (outstr, "ms");
        outtextxy ( 33*8, 21*10, outstr);

    switch (getch ())
        {
        case '\x0':
        switch (getch ())
                {
                case 'K':      /* Cursor Left */
                switch (pos)
                        {
                        case 0:
                        (*min1)--;
                        if (*min1 < 0L)        *min1 = 0L;
                        break;

                        case 1:
                        (*max1)--;
                        if (*max1 <= *min1)    *max1 = *min1 + 1L;
                        break;

                        case 2:
                        (*min2)--;
                        if (*min2 < 0L)        *min2 = 0L;
                        break;

                        case 3:
                        (*max2)--;
                        if (*max2 <= *min2)    *max2 = *min2 + 1L;
                        break;
                        }
                break;

                case 'M':      /* Cursor Right */
                switch (pos)
                        {
                        case 0:
                        (*min1)++;
                        if (*min1 >= *max1)    *min1 = *max1 - 1L;
                        break;

                        case 1:
                        (*max1)++;
```

```
        if (*max1 >= maxf1)    *max1 = maxf1;
        break;

        case 2:
        (*min2)++;
        if (*min2 >= *max2)    *min2 = *max2 - 1L;
        break;

        case 3:
        (*max2)++;
        if (*max2 >= maxf2)    *max2 = maxf2;
        break;
        }
break;

case 'P':      /* Cursor Down */
switch (pos)
        {
        case 0:
        *min1 -= 100;
        if (*min1 < 0L)        *min1 = 0L;
        break;

        case 1:
        *max1 -= 100;
        if (*max1 <= *min1)    *max1 = *min1 + 1L;
        break;

        case 2:
        *min2 -= 100;
        if (*min2 < 0L)        *min2 = 0L;
        break;

        case 3:
        *max2 -= 100;
        if (*max2 <= *min2)    *max2 = *min2 + 1L;
        break;
        }
break;

case 'H':      /* Cursor Up */
switch (pos)
        {
        case 0:
        *min1 += 100;;
        if (*min1 >= *max1)    *min1 = *max1 - 1L;
        break;

        case 1:
        *max1 += 100;
        if (*max1 >= maxf1)    *max1 = maxf1 - 1L;
        break;
```

```
                case 2:
                *min2 += 100;
                if (*min2 >= *max2)    *min2 = *max2 - 1L;
                break;

                case 3:
                *max2 += 100;
                if (*max2 >= maxf2)    *max2 = maxf2 - 1L;
                break;
                }
        break;

        default:
        break;
        }
break;

case 0x67:              /* G */
savegraph ((UWORD) 349);
ok  = FALSE;
ret = FALSE;
break;

case 0x1B:              /* Escape */
ok  = FALSE;
ret = FALSE;
break;

case 0x09:              /* Tabulator */
pos++;
if (pos == 4)
        {
        pos = 0;
        }
break;

case 0x0D:              /* Return */
if (mode != 2)
        {
        if ((*max1 - *min1) > 32767L)
                {
                beep ();
                break;
                }
        if ((*max2 - *min2) > 32767L)
                {
                beep ();
                break;
                }
        }
ok  = FALSE;
ret = TRUE;
break;
```

```
            default:
            break;
            }

        switch (pos)
            {
            case 0:
            setviewport ( 8*8, (YREL+5)*10, 18*8, (YREL+ 6)*10+8, 1);
            clearviewport ();
            break;

            case 1:
            setviewport ( 8*8, (YREL+12)*10, 18*8, (YREL+13)*10+8, 1);
            clearviewport ();
            break;

            case 2:
            setviewport ( 8*8, (YREL+17)*10, 18*8, (YREL+18)*10+8, 1);
            clearviewport ();
            break;

            case 3:
            setviewport ( 8*8, (YREL+24)*10, 18*8, (YREL+25)*10+8, 1);
            clearviewport ();
            break;

            default:
            break;
            }

        if (pos <= 1)
            {
            setviewport (33*8,  (YREL+ 9)*10,   43*8,   (YREL+ 9)*10+8, 1);
            clearviewport ();
            setviewport (3*8+3, (YREL+ 8)*10+3, 24*8-3, (YREL+10)*10-3, 1);
            clearviewport ();
            }
        else
            {
            setviewport (33*8,  (YREL+21)*10,   43*8,   (YREL+21)*10+8, 1);
            clearviewport ();
            setviewport (3*8+3, (YREL+20)*10+3, 24*8-3, (YREL+22)*10-3, 1);
            clearviewport ();
            }

         setviewport (0*8, YREL*10, 49*8, (YREL+27)*10, 1);

         } while (ok);

        clearviewport ();

return (ret);
```

```
}

/*********************************************************************/
/* Frames für Fouriertransformation auswählen                        */
/*********************************************************************/

BOOL     frameselect (char *file, int *min, int *max, int frame)
{
char     outstr [40];
double   wert;
int      maxf,
         pos = 0;
BOOL     ok  = TRUE,
         ret;

         maxf = *max;

         setviewport (0*8, YREL*10, 49*8, (YREL+27)*10, 1);
         clearviewport ();
         setcolor (BLUE);
         setlinestyle (SOLID_LINE, 0xFFFF, THICK_WIDTH);
         setwritemode (COPY_PUT);
         rectangle (0*8,  0*10, 49*8, 27*10);
         rectangle (1*8,  2*10, 48*8, 26*10);

         rectangle (3*8,  8*10, 24*8, 10*10);
         setcolor (YELLOW);
         outtextxy (2*8, 1*10, "Fouriertransformation:");

         setcolor (LIGHTGRAY);
         outtextxy (3*8, 3*10, "File:");
         outtextxy (9*8, 3*10, file);

         outtextxy ( 3*8,  5*10, "von:");
         outtextxy ( 3*8,  6*10, "Frame:");
         outtextxy ( 3*8, 12*10, "bis:");
         outtextxy ( 3*8, 13*10, "Frame:");
         outtextxy (26*8,  8*10, "Dauer:");
         outtextxy (26*8,  9*10, "Frame(s):");

         do{
               if (pos == 0)  setcolor (WHITE);
               else           setcolor (DARKGRAY);
               wert = ((double) (*min) * frame / (fg * 2.0)) * 1000.0;
               sprintf (outstr, "%.3f", wert);
               strcat (outstr, "ms");
               outtextxy (10*8, 5*10, outstr);
               sprintf (outstr, "%d", *min);
               outtextxy (10*8, 6*10, outstr);
               line (3*8+4+ (int) ((*min)*160/maxf),  8*10+3,
                     3*8+4+ (int) ((*min)*160/maxf), 10*10-3);

               if (pos == 1)  setcolor (WHITE);
```

```
        else            setcolor (DARKGRAY);
        wert = ((double) (*max) * frame / (fg * 2.0)) * 1000.0;
        sprintf (outstr, "%.3f", wert);
        strcat (outstr, "ms");
        outtextxy (10*8, 12*10, outstr);
        sprintf (outstr, "%d", *max);
        outtextxy (10*8, 13*10, outstr);
        line (3*8+4+ (int) ((*max)*160/maxf),  8*10+3,
              3*8+4+ (int) ((*max)*160/maxf), 10*10-3);

        setcolor (DARKGRAY);
        wert = ((double) (*max - *min) * frame / (fg * 2.0)) * 1000.0;
        sprintf (outstr, "%.3f", wert);
        strcat (outstr, "ms");
        outtextxy (36*8, 8*10, outstr);
        sprintf (outstr, "%d", *max - *min);
        outtextxy (36*8, 9*10, outstr);

    switch (getch ())
        {
        case '\x0':
        switch (getch ())
                {
                case 'K':       /* Cursor Left */
                switch (pos)
                        {
                        case 0:
                        (*min)--;
                        if (*min < 0)           *min = 0;
                        break;

                        case 1:
                        (*max)--;
                        if (*max <= *min)       *max = *min + 1;
                        break;
                        }
                break;

                case 'M':       /* Cursor Right */
                switch (pos)
                        {
                        case 0:
                        (*min)++;
                        if (*min >= *max)       *min = *max - 1;
                        break;

                        case 1:
                        (*max)++;
                        if (*max >= maxf)       *max = maxf;
                        break;
                        }
                break;
```

```
        case 'P':       /* Cursor Down */
        switch (pos)
                {
                case 0:
                *min -= 5;
                if (*min < 0)           *min = 0;
                break;

                case 1:
                *max -= 5;
                if (*max <= *min)       *max = *min + 1;
                break;
                }
        break;

        case 'H':       /* Cursor Up */
        switch (pos)
                {
                case 0:
                *min += 5;;
                if (*min >= *max)       *min = *max - 1;
                break;

                case 1:
                *max += 5;
                if (*max >= maxf)       *max = maxf - 1;
                break;
                }
        break;

        default:
        break;
        }
break;

case 0x67:              /* G */
savegraph ((UWORD) 349);
ok  = FALSE;
ret = FALSE;
break;

case 0x1B:              /* Escape */
ok  = FALSE;
ret = FALSE;
break;

case 0x09:              /* Tabulator */
pos++;
if (pos == 2)
        {
        pos = 0;
        }
break;
```

```
                case 0x0D:            /* Return */
                ok  = FALSE;
                ret = TRUE;
                break;

                default:
                break;
                }

          if    (pos == 0)
                {
                setviewport (10*8, (YREL+ 5)*10, 21*8, (YREL+ 6)*10+8, 1);
                clearviewport ();
                }
          else if (pos == 1)
                {
                setviewport (10*8, (YREL+12)*10, 21*8, (YREL+13)*10+8, 1);
                clearviewport ();
                }
          setviewport (36*8,  (YREL+ 8)*10,   46*8,   (YREL+ 9)*10+8, 1);
          clearviewport ();
          setviewport (3*8+3, (YREL+ 8)*10+3, 24*8-3, (YREL+10)*10-3, 1);
          clearviewport ();
          setviewport (0*8, YREL*10, 49*8, (YREL+27)*10, 1);

         } while (ok);

          clearviewport ();

return (ret);
}

/*********************************************************************/
/* Faktoren für "Files kombinieren" auswählen                        */
/*********************************************************************/

BOOL      combineselect (char *file1, char *file2,char *file3,double *f1,double *f2,double *f3,int mode)
{
char      num    [20];
int       pos = 1,
          ret = 0;


          setviewport (0*8, YREL*10, 37*8, (YREL+13)*10, 1);
          clearviewport ();
          setcolor (BLUE);
          setlinestyle (SOLID_LINE, 0xFFFF, THICK_WIDTH);
          setwritemode (COPY_PUT);
          rectangle (0*8,  0*10, 37*8, 13*10);
          rectangle (1*8,  2*10, 36*8, 12*10);
          line (4*8, 8*10, 34*8, 8*10);
```

```
setcolor (YELLOW);
outtextxy (2*8, 1*10, "Files kombinieren:");

setcolor (LIGHTGRAY);
outtextxy (20*8,  4*10, "*");
outtextxy (22*8,  4*10, file1);
outtextxy (20*8,  6*10, "*");
outtextxy (22*8,  6*10, file2);
outtextxy ( 4*8, 10*10, "=");
outtextxy (20*8, 10*10, "*");
outtextxy (22*8, 10*10, file3);

setcolor (YELLOW);
if (mode == 1)       outtextxy ( 4*8,  6*10, "+");
else                 outtextxy ( 4*8,  6*10, "*");

setcolor (DARKGRAY);
sprintf (num, "%.8g", *f1);
outtextxy (6*8,  4*10, num);
sprintf (num, "%.8g", *f2);
outtextxy (6*8,  6*10, num);
sprintf (num, "%.8g", *f3);
outtextxy (6*8, 10*10, num);

do{
switch (pos)
      {
      case 1:
      sprintf (num, "%.8g", *f1);
      ret = input (6, YREL+4, 13, &num[0],1);
      if (ret == 20)
             {
             savegraph ((UWORD) 349);
             }
      *f1  = atof (&num[0]);
      break;

      case 2:
      sprintf (num, "%.8g", *f2);
      ret = input (6, YREL+6, 13, &num[0],1);
      if (ret == 20)
             {
             savegraph ((UWORD) 349);
             }
      *f2  = atof (&num[0]);
      break;

      case 3:
      sprintf (num, "%.8g", *f3);
      ret = input (6, YREL+10, 13, &num[0],1);
      if (ret == 20)
             {
             savegraph ((UWORD) 349);
```

```
                }
        if (ret == 0)
                {
                if ((*f1 != 0.0) || (*f2 != 0.0))
                        {
                        *f3 = 1.0 / (fabs(*f1) + fabs(*f2));
                        }
                else
                        {
                        *f3 = 1.0;
                        }
                sprintf (num, "%.8g", *f3);
                setcolor (LIGHTGRAY);
                setviewport (0*8, YREL*10, 37*8, (YREL+13)*10, 1);
                  outtextxy (6*8, 10*10, num);
                ret = 1;
                }
        else
                {
                *f3  = atof (&num[0]);
                }
        break;

        case 4:
        ret = input (22, YREL+10, 12, file3,1);
        if (ret == 20)
                {
                savegraph ((UWORD) 349);
                }
        if (  (ret == 0)
           ||
             ((ret == 1) && ((*file3 == 0x00) || (*file3 == '.'))) )
                {
                beep ();
                ret = 0;
                }
        break;

        default:
        break;
        }

if ((pos == 4) && (ret == 1))
        {
        pos = 10;
        ret = 50;
        }
else
        {
        switch (ret)
                {
                case 1:
                case 2:
```

```
                        pos++;
                        break;

                        case -1:
                        case -2:
                        pos--;
                        break;

                        default:
                        break;
                        }
                  if (pos <= 0 ) pos = 4;
                  if (pos >= 5)   pos = 1;
                  }
            }while (ret != 50);

           setviewport (0*8, YREL*10, 37*8, (YREL+13)*10, 1);
           clearviewport ();

            if (pos == 10)
                  {
                  return (TRUE);
                  }
return (FALSE);
}

/*********************************************************************/
/* Option Auswahlbox                                                 */
/*     anz     - Anzahl der Optionen                                 */
/*     op      - Überschrift                                         */
/*     op1-op8 - Optionen                                            */
/*********************************************************************/

int      option (int xpos, int ypos,
                  int   anz, char *op,
                  char *op1, char *op2, char *op3, char *op4,
                  char *op5, char *op6, char *op7, char *op8)
{
void              *ptr;
unsigned int      size;
int               i,
                  num = 1;
BOOL              ok  = TRUE;

            setviewport (0, 0, xgmax, ygmax, 1);
            size = imagesize (xpos*8, ypos*10, (xpos+WIDTH)*8, (ypos+anz+5)*10);
            ptr = malloc (size);
            if (!ptr)
                  {
                  beep ();
                  return (0);
                  }
            getimage (xpos*8, ypos*10, (xpos+WIDTH)*8, (ypos+anz+5)*10, ptr);
```

```
setviewport (xpos*8, ypos*10, (xpos+WIDTH)*8, (ypos+anz+5)*10, 1);
clearviewport ();
setcolor (BLUE);
setlinestyle (SOLID_LINE, 0xFFFF, THICK_WIDTH);
setwritemode (COPY_PUT);
rectangle (0*8, 0*10,  WIDTH   *8, (anz+4)*10);
rectangle (1*8, 2*10, (WIDTH-1)*8, (anz+3)*10);
setcolor (YELLOW);
outtextxy (2*8, 1*10, op);

do{
      if (num > anz)
             {
             num = 1;
             }
      if (num < 1)
             {
             num = anz;
             }

      for (i = 1; i <= anz; i++)
             {
             if (i == num)
                    {
                    setcolor (WHITE);
                    }
             else
                    {
                    setcolor (DARKGRAY);
                    }
             switch (i)
                    {
                    case 1: outtextxy (2*8, (2+i)*10, op1);      break;
                    case 2: outtextxy (2*8, (2+i)*10, op2);      break;
                    case 3: outtextxy (2*8, (2+i)*10, op3);      break;
                    case 4: outtextxy (2*8, (2+i)*10, op4);      break;
                    case 5: outtextxy (2*8, (2+i)*10, op5);      break;
                    case 6: outtextxy (2*8, (2+i)*10, op6);      break;
                    case 7: outtextxy (2*8, (2+i)*10, op7);      break;
                    case 8: outtextxy (2*8, (2+i)*10, op8);      break;
                    default:
                    break;
                    }
             }

      switch (getch ())
             {
             case 0x00:
             switch (getch ())
                    {
                    case 'P':              /* Cursor Down */
                    num++;
                    break;
```

```
                        case 'H':               /* Cursor Up */
                        num--;
                        break;

                        default:
                        break;
                        }
                    break;

                    case 0x67:                  /* G */
                    savegraph ((UWORD) 349);
                    ok  = FALSE;
                    num = 0;
                    break;

                    case 0x1B:                  /* Escape */
                    ok  = FALSE;
                    num = 0;
                    break;

                    case 0x0D:                  /* Return */
                    ok  = FALSE;
                    break;

                    default:
                    break;
                    }
        }while (ok);

        setviewport (0, 0, xgmax, ygmax, 1);
        putimage (xpos*8, ypos*10, ptr, COPY_PUT);
        free (ptr);

return (num);
}

/*********************************************************************/
/* Fehlermeldung ausgeben                                            */
/*      number: 1 - File öffnen (lesen)                              */
/*              2 - File öffnen (schreiben)                          */
/*              3 - Kein Datenfile                                   */
/*              4 - Zu wenig Speicher                                */
/*              5 - Gleichung nicht lösbar                           */
/*              6 - Iteration konvergiert nicht                      */
/*              7 - Fließkommafehler                                 */
/*              8 - Gleichungssystem nicht lösbar                    */
/*              9 - Filterparameter                                  */
/*             10 - Wertebereich                                     */
/*             11 - File konnte nicht gelöscht werden                */
/*             12 - File konnte nicht umbenannt werden               */
/*                                                                   */
/* Warnung ausgeben                                                  */
```

Anhang C: Listing des Quellcodes von SPV (SPV.C)

```
/*             20 - File wirklich löschen ?                            */
/*************************************************************************/

BOOL showerror (int number)
{
void           *ptr    = NULL;
unsigned int   size    = 0;
BOOL            ok     = FALSE,
               ret     = FALSE;

         setviewport (0, 0, xgmax, ygmax, 1);
         size = imagesize (150, 100, 490, 250);
         ptr = malloc (size);
         if (!ptr)
               {
               beep ();
               return (ret);
               }
         getimage (150, 100, 490, 250, ptr);
         setviewport (150, 100, 490, 250, 1);
         clearviewport ();
         setcolor (WHITE);
         setlinestyle (SOLID_LINE, 0, THICK_WIDTH);
         setwritemode (COPY_PUT);
         rectangle (0, 0, 340, 150);
         setfillstyle (SOLID_FILL, RED);
         floodfill (10, 10, WHITE);
         setcolor (WHITE);
         rectangle (10, 10, 330, 140);

         if (number < 20)
               {
               outtextxy ((340-23*8)/2,  30, "---  FEHLERMELDUNG  ---");
               outtextxy ((340-15*8)/2, 120, "<<<  Taste  >>>");
               }
         else
               {
               outtextxy ((340-17*8)/2,  30, "---  WARNUNG  ---");
               outtextxy ((340-37*8)/2, 120, "<<<  ESC-Abbruch / RETURN-Weiter  >>>");
               }

         switch (number)
               {
               case 1:
               outtextxy ((340-27*8)/2,  50, "File konnte nicht zum lesen");
               outtextxy ((340-15*8)/2,  70, "geöffnet werden.");
               outtextxy ((340-22*8)/2,  90, "Datenträger überprüfen.");
               break;

               case 2:
               outtextxy ((340-31*8)/2,  50, "File konnte nicht zum schreiben");
               outtextxy ((340-15*8)/2,  70, "geöffnet werden.");
               outtextxy ((340-22*8)/2,  90, "Datenträger überprüfen.");
```

```
break;

case 3:
outtextxy ((340-25*8)/2,  60, "Angewähltes File war kein");
outtextxy ((340- 9*8)/2,  80, "Datenfile.");
break;

case 4:
outtextxy ((340-39*8)/2,  60, "Nicht genug Speicher für diese Funktion");
outtextxy ((340- 9*8)/2,  80, "verfügbar.");
break;

case 5:
outtextxy ((340-32*8)/2,  60, "Gleichung nicht lösbar für diese");
outtextxy ((340- 9*8)/2,  80, "Parameter.");
break;

case 6:
outtextxy ((340-28*8)/2,  60, "Iteration konvergiert nicht.");
outtextxy ((340-20*8)/2,  80, "Noch einmal starten.");
break;

case 7:
outtextxy ((340-28*8)/2,  50, "Wertebereich wurde bei einer");
outtextxy ((340-35*8)/2,  70, "Fließkommaberechnung überschritten.");
outtextxy ((340-25*8)/2,  90, "Programm wird terminiert.");
break;

case 8:
outtextxy ((340-26*8)/2,  60, "Gleichungssystem ist nicht");
outtextxy ((340- 7*8)/2,  80, "lösbar.");
break;

case 9:
outtextxy ((340-36*8)/2,  50, "Leistung läßt sich bei den gewählten");
outtextxy ((340-33*8)/2,  70, "Parametern für das Eingangsfilter");
outtextxy ((340-22*8)/2,  90, "nicht konstant halten.");
break;

case 10:
outtextxy ((340-28*8)/2,  60, "Erlaubter Wertebereich wurde");
outtextxy ((340-10*8)/2,  80, "verlassen.");
break;

case 11:
outtextxy ((340-26*8)/2,  60, "File konnte nicht gelöscht");
outtextxy ((340- 7*8)/2,  80, "werden.");
break;

case 12:
outtextxy ((340-27*8)/2,  60, "File konnte nicht umbenannt");
outtextxy ((340- 7*8)/2,  80, "werden.");
break;
```

```
                case 20:
                outtextxy ((340-31*8)/2,  60, "Angewählte Datei wird gelöscht.");
                outtextxy ((340-27*8)/2,  80, "Befehl wirklich ausführen ?");
                break;

                default:
                outtextxy ((340-19*8)/2,  70, "Unbekannter Fehler.");
                break;
                }

          while (!ok)
                {
                switch (getch ())
                        {
                        case 0x67:
                        savegraph ((UWORD) 349);
                        break;

                        case 0x1B:     /* Escape */
                        ret = FALSE;
                        ok++;
                        break;

                        case 0x0D:     /* Return */
                        ret = TRUE;
                        ok++;
                        break;

                        default:
                        if (number < 20)
                                {
                                ok++;
                                }
                        break;
                        }
                }

          setviewport (0, 0, xgmax, ygmax, 1);
          putimage (150, 100, ptr, COPY_PUT);
          free (ptr);

return (ret);
}

/*********************************************************************/
/* Grafik speichern (PCX)                                            */
/*********************************************************************/

BOOL      savegraph (UWORD yend)
{
FILE            *fp;
PCXHEADERheader;
```

Anhang C: Listing des Quellcodes von SPV (SPV.C)

```
char            buffer   [80] [ 4],
                fullpath [80],
                num      [ 4],
                type, count;
int              red, green, blue,
                colbar,
                color,
                i, j, k;

          setviewport (0, 0, xgmax, ygmax, 1);

          for (i = 0; i < 100; i++)
                {
                strcpy (fullpath, workpath);
                strcat (fullpath, "\\SPVPIC");
                sprintf (num, "%d", i);
                strcat (fullpath, num);
                strcat (fullpath, ".PCX");
                fp = fopen (fullpath, "rb");
                if (fp == NULL)
                        {
                        break;
                        }
                fclose (fp);
                }
          fp = fopen (fullpath, "wb");
          if (fp == NULL)
                {
                showerror (2);
                return (FALSE);
                }

          header.Ident = 10;          /* Erkennungsbyte */
          header.Version       = 2;           /* Version 2.8 mit Palette */
          header.Compression   = 1;           /* Komprimiert */
          header.BitPerPixel   = 1;    /* 1 Bit pro Bildpunkt */
          header.XMin          = 0;
          header.YMin          = 0;
          header.XMax          = xgmax;
          header.YMax          = yend;
          header.XDpi          = 0;
          header.YDpi          = 0;

          for (i = 0; i < 16; i++)
                {
                switch (i)
                        {
/* RGB-Werte:  0 16 32 48 64 80 96 112 128 144 160 176 192 208 224 240 */
/* Color-Reverse */
/* BLACK */            case  WHITE:          red =   0; green =   0; blue =   0; break;
/* BLUE */             case  LIGHTBLUE:      red =   0; green =   0; blue = 208; break;
/* GREEN */            case  LIGHTGREEN:     red =   0; green = 208; blue =   0; break;
/* CYAN */             case  LIGHTCYAN:      red =   0; green = 208; blue = 208; break;
```

Anhang C: Listing des Quellcodes von SPV (SPV.C)

```
/* RED */      case  LIGHTRED:       red = 208; green =   0; blue =   0; break;
/* MAGENTA */         case  LIGHTMAGENTA:   red = 208; green =   0; blue = 208; break;
/* BROWN */           case  YELLOW:         red = 176; green = 176; blue =   0; break;
/* LIGHTGRAY */case  DARKGRAY:       red = 176; green = 176; blue = 176; break;
/* DARKGRAY */ case  LIGHTGRAY:      red =  32; green =  32; blue =  32; break;
/* LIGHTBLUE */case  BLUE:           red =   0; green =   0; blue =  96; break;
/* LIGHTGREEN */      case  GREEN:          red =   0; green =  96; blue =   0; break;
/* LIGHTCYAN */case  CYAN:           red =   0; green =  96; blue =  96; break;
/* LIGHTRED */ case  RED:            red =  96; green =   0; blue =   0; break;
/* LIGHTMAGENTA */    case  MAGENTA: red =  96; green =   0; blue =  96; break;
/* YELLOW */          case  BROWN:          red = 144; green = 144; blue =   0; break;
/* WHITE */           case  BLACK:          red = 240; green = 240; blue = 240; break;
                      }
               header.ColorMap[i][0]  = (UBYTE) red;
               header.ColorMap[i][1]  = (UBYTE) green;
               header.ColorMap[i][2]         = (UBYTE) blue;
               }

         header.Reserved             = 0;
         header.BitPlanes            = 4;
         header.BytesPerRow          = 80;   /* Bytes pro Zeile (640 / 8 = 80) */
         header.CMapInterpretation   = 1;    /* Farb oder SW-Grafik */

         for (i = 0; i < 29; i++)
               {
               header.Padding[i] = 0;        /* Leerbytes */
               }

         fwrite (&header,  sizeof (PCXHEADER), 1, fp);

         for (j = 0; j <= yend; j++)
               {
               colbar = j % 16;
               if (j == yend)
                      {
                      colbar = BLACK;
                      }
               memset (&buffer[0][0], 0x00, 80*4);
               for (i = 0; i <= xgmax; i++)
                      {
                      putpixel (i, ygmax, colbar);
                      color = getpixel (i, j);
                      if ((UBYTE) color & 0x01) buffer [i / 8][0] |= 0x80 >> (i % 8);
                      if ((UBYTE) color & 0x02) buffer [i / 8][1] |= 0x80 >> (i % 8);
                      if ((UBYTE) color & 0x04) buffer [i / 8][2] |= 0x80 >> (i % 8);
                      if ((UBYTE) color & 0x08) buffer [i / 8][3] |= 0x80 >> (i % 8);
                      }

               for (k = 0; k < 4; k++)
                      {
                      type    = buffer[0][k];
                      count   = 1;
                      for (i = 1; i < 80; i++)      /* Run-Length-Encoding */
```

```
                        {
                        if (buffer[i][k] == type)
                                {
                                count++;
                                if (count > 63)
                                        {
                                        fputc (0xFF, fp);
                                        fputc (type, fp);
                                        count = 1;
                                        }
                                }
                        else
                                {
                                count |= 0xC0;
                                fputc (count, fp);
                                fputc (type,  fp);
                                type  = buffer[i][k];
                                count = 1;
                                }
                        }
                if ((count > 1) || ((count == 1) && (i == 80)))
                        {
                        count |= 0xC0;
                        fputc (count, fp);
                        fputc (type,  fp);
                        }
                } /* k */
        } /* j */

    fclose (fp);

return (TRUE);
}
```

SPVFUNC.C

Anhang C: Listing des Quellcodes von SPV (SPVFUNC.C)

```
/*********************************************************************/
/* SPV                                                               */
/* Autor: Harald Zottmann                                            */
/* Matr.: 787378                                                     */
/* Studienarbeit 1992/93                                             */
/* Technische-Hochschule-Darmstadt                                   */
/*********************************************************************/

#include "SPVINC.H"

extern   BOOL         qcalc;
extern   double       fg;
extern   SAMPARAMETER sampar;
extern   KENNLINIE    kennpar;
extern   SIRPFILTER   sirpfil;
extern   VERTEILUNG   verteilung;

/*********************************************************************/
/* Autokorrelation / Kreuzkorrelation                                */
/*********************************************************************/

BOOL     korr (BOOL mode)
{
BOOL     timeselect ();

FILE           *fp            [2],
               *tp;
char           *filenameptr,
               *endptr,
               str            [40]           = "",
               num            [64]           = "",
               in_file [2] [13]       = {"",""},
               out_file       [13]           = "";
double far     *src           [2]            = {NULL,NULL};
double         wert                          = 0.0;
long           filelength,
               sampanz,
               min            [2]            = {0L,0L},
               max            [2],
               s              [2];
int            smin,
               ordinate,
               samplength     [2],
               d,i,j,n,
               n1,n2,
               percent,
               ret;

         for (j = 0; j <= 1; j++)
               {
               if ((mode == 0) && (j == 1))  /* AKF */
                      {
                      s [1]                  = s [0];
```

```
                fp[1]                   = fp[0];
                samplength[1]  = samplength[0];
                }

        else                            /* KKF */
                {
                filenameptr = getfilename ();
                if (*filenameptr == 0x00)
                        {
                        fcloseall ();
                        return (FALSE);
                        }

                strcpy (&in_file[j][0], filenameptr);
                fp[j] = fopen (&in_file[j][0], "rb");
                if (fp[j] == NULL)
                        {
                        fcloseall ();
                        beep ();
                        showerror (1);
                        return (FALSE);
                        }
                fseek (fp[j], 0L, SEEK_END);
                filelength = ftell (fp[j]);
                samplength[j] = getsamplength (fp[j]);
                if (samplength[j] == 0)
                        {
                        fcloseall ();
                        beep ();
                        showerror (3);
                        return (FALSE);
                        }
                s[j] = (long) (filelength / samplength[j]);
                }
        }

max[0] = s[0];
max[1] = s[1];
if (!timeselect (mode, &in_file[0][0], &in_file[1][0], &min[0], &max[0], &min[1], &max[1]))
        {
        fcloseall ();
        return (FALSE);
        }
s[0] = max[0] - min[0] + 1L;
s[1] = max[1] - min[1] + 1L;

busy (TRUE);

for (j = 0; j <= 1; j++)
        {
        src[j] = (double far *) farcalloc ((unsigned long) s[j], (unsigned long) sizeof(double));
        if (src[j] == NULL)
                {
```

```
                fcloseall ();
                if (src[0])    farfree (src[0]);
                if (src[1])    farfree (src[1]);
                busy (FALSE);
                beep ();
                showerror (4);
                return (FALSE);
                }

        fseek (fp[j], (long) (min[j] * (long) samplength[j]), SEEK_SET);

        for (i = 0; i < (int) s[j]; i++)
                {
                fread (&num[0], samplength[j], 1, fp[j]);
                wert = strtod (&num[0], &endptr);
                if ((wert == 0.0) && ((*endptr != 0x0D)  && (*endptr != 0x20)))
                        {
                        fcloseall ();
                        if (src[0])    farfree (src[0]);
                        if (src[1])    farfree (src[1]);
                        busy (FALSE);
                        beep ();
                        showerror (3);
                        return (FALSE);
                        }
                *(src[j] + i) = wert;
                }
        }
fcloseall ();

busy (FALSE);

switch (mode)
        {
        case 0:
        ret = inputfilename ("AKF-Filename:", "AKF", filenameptr, out_file);
        break;

        case 1:
        ret = inputfilename ("KKF-Filename:", "KKF", filenameptr, out_file);
        break;
        }

if (ret == 50)
        {
        farfree (src[0]);
        farfree (src[1]);
        return (FALSE);
        }

tp = fopen (out_file,"wb");
if (tp == NULL)
        {
```

```
        beep ();
        farfree (src[0]);
        farfree (src[1]);
        showerror (2);
        return (FALSE);
        }

smin = (int) s[0];                  /* Kleinste Samplelänge */
if ((int) s[1] < smin)
        {
        smin = (int) s[1];
        }

sampanz = s[0] + s[1] - 1L;

switch (mode)                /* Position der Ordinate */
        {
        case 0:
        if ((min[0] < min[1]) || (min[0] > max[1]))
                {
                ordinate = 0;
                }
        else
                {
                ordinate = (int) (max[0] - min[1] + 1L);
                }
        break;

        case 1:
        ordinate = (int) s[1];
        break;
        }

fprintf (tp, "%-16d", ordinate);
fputs ("\x0D\x0A", tp);

if (mode == 1)
        {
        setviewport (0*8, 6*10, 32*8, 10*10, 1);
        setcolor (BLUE);
        rectangle (0*8, 0*10, 32*8, 4*10);
        setcolor (YELLOW);
        outtextxy (3*8,1*10,"Kreuzkorrelation:");
        setcolor (DARKGRAY);
        strcpy (str, &in_file[0][0]);
        strcat (str, " + ");
        strcat (str, &in_file[1][0]);
        outtextxy (3*8,2*10, str);
        setcolor (WHITE);
        outtextxy (3*8, 3*10, out_file);
        }

showpercent (0, 1);
```

```
        d = (int) (s[1] - s[0]);                    /* Differenz */
        for ( i = (int) -(s[0]-1L); i <= (int) (s[1]-1L); i++)
              {
              percent = (int) ( ((long) i +  s[0] ) * 100L / sampanz);
              showpercent (percent, 0);

              if (i >= 0)    n1 = 0;
              else            n1 = -i;
              if (i <= d)     n2 = (int) (s[0] - 1L);
              else           n2 = (int) (s[0] - 1L + (long) d - (long) i);
              wert = 0.0;
              for (n = n1; n <= n2; n++)
                     {
                     wert += (*(src[0] + n)  *  *(src[1] + n + i));
                     }
              wert /= smin;
              fprintf (tp, "%16.7e", wert);
              fputs ("\x0D\x0A", tp);
              }
        farfree (src[0]);
        farfree (src[1]);
        fclose (tp);

        if (mode == 1)
              {
              setviewport (0*8, 6*10, 32*8, 10*10, 1);
              clearviewport ();
              }

        showpercent (0, 2);
        shortbeep ();
        displaydata (out_file, "", 1);

return (TRUE);
}

/*********************************************************************/
/* Es wird eine Bitreversierung von j durchgeführt                   */
/* (Spiegelunglung um das mittlere Bit)                              */
/*********************************************************************/

int      bitreverse (int j, int nu)
{
int      j1 , j2,
         i  , ib;

         j1 = j;
         ib = 0;

         for (i = 1; i <= nu; i++)
              {
              j2 =  j1 >> 1;
              ib = (ib << 1) + (j1 - (j2 << 1));
```

```
                j1 =  j2;
                }

return (ib);
}

/*********************************************************************/
/* Schnelle Fouriertransformation                                    */
/*********************************************************************/

BOOL     fastfourier (char *filename)
{
BOOL     frameselect ();
int      bit_reverse ();

FILE           *fp, *tp;

char            *endptr,
                num [64]       = "",
                out_file [13]  = "";
double far      *xreal                 = NULL;
double far      *ximag                 = NULL;
double far      *s_ptr                 = NULL;
double far      *c_ptr                 = NULL;
double          c, s,
                arg,
                treal, timag,
                wert                   = 0.0;
long            filelength,
                sampanz;               /* Anzahl der Werte */
int             h, i, j, k, l,
                n, nu,
                n2, nu1, kn2,
                min, max,
                samplength,
                percent,
                ret;
BOOL             error                 = FALSE;

          fp = fopen (filename,"rb");
          if (fp == NULL)
                {
                beep ();
                showerror (1);
                return (FALSE);
                }
          fseek (fp,0L,SEEK_END);
          filelength = ftell (fp);
          samplength = getsamplength (fp);
          if (samplength == 0)
                {
                beep ();
                showerror (3);
```

```
        return (FALSE);
        }
sampanz    = (long) (filelength / samplength);

ret = option (OPTX, YREL, 4, "Framelänge:", "512", "1024", "2048", "4096");
switch (ret)
        {
        case 1: n =  512; nu =  9; break;
        case 2: n = 1024; nu = 10; break;
        case 3: n = 2048; nu = 11; break;
        case 4: n = 4096; nu = 12; break;

        default:
        fclose (fp);
        return (FALSE);
        }

min = 0;
max = (int) (sampanz / n);          /* Anzahl der Frames */
if (max == 0)               /* File kleiner als Frame */
        {
        fclose (fp);
        beep ();
        showerror (-1);
        return (FALSE);
        }

if (max > 1)                        /* Mehr als ein Frame */
        {
        if (!frameselect (filename, &min, &max, n))
                {
                fclose (fp);
                return (FALSE);
                }
        }

ret = inputfilename ("FFT-Filename:", "FFT", filename, out_file);
if (ret == 50)
        {
        fclose (fp);
        return (FALSE);
        }

tp = fopen (out_file,"wb"); /* Zieldatei öffnen */
if (tp == NULL)
        {
        fclose (fp);
        beep ();
        showerror (2);
        return (FALSE);
        }

xreal = (double far *) farcalloc ((unsigned long) n,     (unsigned long) sizeof (double));
```

```
ximag = (double far *) farcalloc ((unsigned long) n,     (unsigned long) sizeof (double));
s_ptr = (double far *) farcalloc ((unsigned long) n / 2, (unsigned long) sizeof (double));
c_ptr = (double far *) farcalloc ((unsigned long) n / 2, (unsigned long) sizeof (double));
if ((xreal == NULL) || (ximag == NULL) || (s_ptr == NULL) || (c_ptr == NULL))
      {
      if (xreal)     farfree (xreal);
      if (ximag)      farfree (ximag);
      if (s_ptr)     farfree (s_ptr);
      if (c_ptr)     farfree (c_ptr);
      fclose (fp);
      fclose (tp);
      beep ();
      showerror (4);
      return (FALSE);
      }

busy (TRUE);

for (i = 0; i < n / 2; i++)  /* Sinus & Cosinustabelle */
      {
      arg = PI2 * i / n;
      *(s_ptr + i) = sin (arg);
      *(c_ptr + i) = cos (arg);
      }

busy (FALSE);

fprintf (tp, "%-16d", n);           /* Framelänge eintragen */
fputs ("\x0D\x0A", tp);

showpercent (0, 1);                 /* Prozentanzeige */

for (j = min; j < max; j++)  /* Äußere Schleife */
      {
      for (i = 0; i < n; i++)
             {
             fread (&num[0], samplength, 1, fp);
             wert = strtod (&num[0],&endptr);
             if ((wert == 0.0) && ((*endptr != 0x0D)  && (*endptr != 0x20)))
                    {
                    error = TRUE;
                    break;
                    }
             *(xreal + i) = wert;   /* Werte in Speicher schreiben */
             *(ximag + i) = 0.0;
             }
      if (error)
             {
             farfree (xreal);
             farfree (ximag);
             farfree (s_ptr);
             farfree (c_ptr);
             fclose (fp);
```

```
        fclose (tp);
        beep ();
        showerror (3);
        return (FALSE);
        }

n2  = n >> 1;
nu1 = nu - 1;
h   = 1 << nu1;
k   = 0;


for (l = 1; l <= nu; l++)
        {
        percent = (int) (((long) (( j  - min) * nu + l) * 100L)
                                / ((max - min) * nu));
        showpercent (percent, 0);
        do {
                for (i = 1; i <= n2; i++)
                        {
                        s = *(s_ptr + bitreverse (k / h, nu));
                        c = *(c_ptr + bitreverse (k / h, nu));

                        kn2     = k + n2;
                        treal   = xreal [kn2] * c + ximag [kn2] * s;
                        timag   = ximag [kn2] * c - xreal [kn2] * s;
                        xreal [kn2] = xreal [k] - treal;
                        ximag [kn2] = ximag [k] - timag;

                        xreal [k]  += treal;
                        ximag [k]  += timag;

                        k++;
                        }
                k += n2;
                }
        while (k < n);

        k = 0;
        nu1--;
        h  = 1  << nu1;
        n2 = n2 >> 1;
        }

for (k = 0; k < n; k++)
        {
        i = bitreverse (k, nu);
        if (i > k)
                {
                treal     = xreal [k];
                timag     = ximag [k];
                xreal [k] = xreal [i];
                ximag [k] = ximag [i];
```

```
                                xreal [i] = treal;
                                ximag [i] = timag;
                                }
                        }
                for (k = 0; k < n / 2; k++)
                        {
                        wert  = sqrt ((xreal[k]*xreal[k]) + (ximag[k]*ximag[k]));
                        fprintf (tp, "%16.7e", wert);
                        fputs ("\x0D\x0A", tp);
                        }
                } /* Äußere Schleife */

        farfree (xreal);
        farfree (ximag);
        farfree (s_ptr);
        farfree (c_ptr);
        fclose (fp);
        fclose (tp);

        showpercent (0, 2);
        shortbeep ();
        displaydata (out_file, "", 2);

return (TRUE);
}

/*********************************************************************/
/* Kennlinie für Verstärkung                                         */
/* mode: 0 - SAM                        (SAM)                        */
/*       1 - SAM korrigiert             (SAM)                        */
/*       2 - Verstärkung                (AMP)                        */
/*       3 - Verstärkung geglättet      (AMP)                        */
/*       4 - Verstärkung * Signal       (SIG)                        */
/*********************************************************************/

BOOL     kenn (char *filename, BOOL mode)
{
FILE           *fp,*tp;
char           *endptr,
               header   [15] = "",
               suffix   [ 5] = "",
               num      [64] = "",
               out_file [13] = "";
double         outmittel      = 0.0,         /* Mittelwert (Ausgang) */
               outvar         = 0.0,         /* Varianz    (Ausgang) */
               inmittel       = 0.0,         /* Mittelwert (Eingang) */
               invar          = 0.0,         /* Varianz    (Eingang) */
               wert           = 0.0,         /* Signalwert (Eingang) */
               vk             = 0.0,         /* Verstärkung v(k) */
               vk_1           = 0.0,         /* v(k-1) */
               uk             = 0.0,         /* SAM-Wert u(k) */
               uk_1           = 0.0;         /* u(k-1) */
int            samplength,
```

```
                ret;
long            filelength,
                sampanz,
                i;
BOOL            error  = FALSE;


          if    (mode == 0)
                {
                ret = option (OPTX, YREL, 2, "Schätzwert:", "Direkt", "Korrigiert");
                switch (ret)
                        {
                        case 1: mode = 0; break;
                        case 2: mode = 1; break;
                        default: return (FALSE);
                        }
                }
          else if       (mode == 1)
                {
                ret = option (OPTX, YREL, 2, "Verstärkung:", "Direkt", "Geglättet");
                switch (ret)
                        {
                        case 1: mode = 2; break;
                        case 2: mode = 3; break;
                        default: return (FALSE);
                        }
                }
          else
                {
                mode = 4;
                }

          fp = fopen (filename,"rb");
          if (fp == NULL)
                {
                beep ();
                showerror (1);
                return (FALSE);
                }
          fseek (fp,0L,SEEK_END);
          filelength = ftell (fp);
          samplength = getsamplength (fp);
          if (samplength == 0)
                {
                beep ();
                showerror (3);
                return (FALSE);
                }
          sampanz    = (long) (filelength / samplength);

          if (!qcalc)
                {
                if      (mode < 2)
```

```
                {
                strcpy (header, "SAM-Filename:");
                strcpy (suffix, "SAM");
                }
        else if (mode < 4)
                {
                strcpy (header, "AMP-Filename:");
                strcpy (suffix, "AMP");
                }
        else
                {
                strcpy (header, "SIG-Filename:");
                strcpy (suffix, "SIG");
                }

        ret = inputfilename (header, suffix, filename, out_file);

        if (ret == 50)
                {
                fclose (fp);
                return (FALSE);
                }

        tp = fopen (out_file,"wb");
        if (tp == NULL)
                {
                beep ();
                fclose (fp);
                showerror (2);
                return (FALSE);
                }
        }

    rewind (fp);

    showpercent (0, 1);
    for (i = 0L; i < sampanz; i++)
        {
        if ((i % 100L) == 0L)
                {
                showpercent ((int) (i * 100L / sampanz), 0);
                }

        fread (&num[0], samplength, 1, fp);
        wert = strtod (&num[0], &endptr);
        if ((wert == 0.0) && ((*endptr != 0x0D)  && (*endptr != 0x20)))
                {
                error = TRUE;
                break;
                }

        inmittel += wert;
        invar    += wert * wert;
```

```
uk = wert;
if (i == 0L)                  /* Ersten Wert initialisieren */
        {
        uk_1 = fabs(wert);
        }
if (fabs(uk) > uk_1)   /* SAM */
        {
        uk = (1 - sampar.ar) * fabs(uk) + sampar.ar * uk_1;
        }
else
        {
        uk = (1 - sampar.af) * fabs(uk) + sampar.af * uk_1;
        }
uk_1 = uk;

if (mode > 0)
        {
        uk *= sampar.ac;       /* SAM-Korrektur */
        vk  = kennlinie (uk);  /* Kennlinie */
        if (mode > 2)          /* Glättungsfilter */
                {
                vk = kennpar.ag * vk + kennpar.bg * vk_1;
                vk_1 = vk;
                }
        }

if      (mode < 2)
        {
        if (!qcalc)
                {
                fprintf (tp, "%16.7e", uk);
                }
        outmittel += uk;
        outvar    += uk * uk;
        }
else if (mode < 4)
        {
        if (!qcalc)
                {
                fprintf (tp, "%16.7e", vk);
                }
        outmittel += vk;
        outvar    += vk * vk;
        }
else
        {
        if (!qcalc)
                {
                fprintf (tp, "%16.7e", vk * wert);
                }
        outmittel += (vk * wert);
        outvar    += (vk * wert) * (vk * wert);
```

```
                        }
                if (!qcalc)
                        {
                        fputs ("\x0D\x0A", tp);
                        }
                }
        fclose (fp);
        if (!qcalc)
                {
                fclose (tp);
                }
        showpercent (0, 2);

        if (error)
                {
                if (!qcalc)
                        {
                        remove (out_file);
                        }
                showerror (3);
                beep ();
                return (FALSE);
                }

        outmittel /= (double) sampanz;
        inmittel  /= (double) sampanz;
        if (i != 0L)
                {
                outvar /= (double) i;
                outvar -= outmittel * outmittel;
                invar  /= (double) i;
                invar  -= inmittel * inmittel;
                }
        else
                {
                outvar = 0.0;
                invar  = 0.0;
                }

        shortbeep ();
        if (showstats (invar, inmittel, outvar, outmittel))
                {
                if (!qcalc)
                        {
                        displaydata (filename, out_file , 0);
                        }
                }
return (TRUE);
}

/*********************************************************************/
/* Kennlinie des Verstärkungsverlaufs                                */
/*********************************************************************/
```

Anhang C: Listing des Quellcodes von SPV (SPVFUNC.C)

```
double   kennlinie (double x)
{
double y;

         if    (x <= (kennpar.d * kennpar.us))
               {
               y =( kennpar.ua * pow (kennpar.d, kennpar.p - 1)) / kennpar.us;
               }
         else if (  ((kennpar.d * kennpar.us) < x)
               && (x <= (kennpar.q * kennpar.us)) )
               {
               y = (kennpar.ua / kennpar.us) * pow (x / kennpar.us, kennpar.p - 1);
               }
         else if       (  ((kennpar.q * kennpar.us) < x)
               && (x <= (kennpar.r * kennpar.us)) )
               {
               y =    kennpar.a0
                    + kennpar.a1 *      (x / kennpar.us - 1)
                    + kennpar.a2 * pow ((x / kennpar.us - 1), 2)
                    + kennpar.a3 * pow ((x / kennpar.us - 1), 3);
               }
         else
               {
               y = kennpar.ua / x;
               }

return (y);
}

/*********************************************************************/
/* Gaußsches Fehlerintegral   e^(-t^2)                               */
/* Berechnung nach Simpson-Regel                                     */
/*********************************************************************/

double   erf (double b)
{
double   f4 ();

double   x,
         y1 = 0.0,
         y2 = 0.0,
         y  = 0.0,
         h  = 0.001;                 /* Schrittweite */
int      i,
         n;                          /* Anzahl der Stützstellen */

         n = (int) (b / h);          /* Stützstellen ermitteln */
         if (n %  2) n++;            /* Gerade Anzahl an Stützstellen */
         if (n < 10) n = 10;         /* Mindestanzahl */
         h = (double) (b / n);       /* Genaue Schrittweite */

         for (i = 2; i < n; i += 2)
```

```
                {
                x =  h * i;
                y1 += exp (-1.0 * x * x);
                }
          y1 *= 2.0;

          for (i = 1; i < n; i += 2)
                {
                x = h * i;
                y2 += exp (-1.0 * x * x);
                }
          y2 *= 4.0;
          y = y1 + y2;

          y += 1.0;
          y += exp (-1.0 * b * b);

          y *= b / (3.0 * n);
          y -= f4 (b/2) * pow (b, 5.0) / (180.0 * pow (n, 4.0));

          y *= 2.0 / sqrt (PI);

return (y);
}

double   f4 (double x)
{
return (   (16.0 * pow (x, 4.0) - 48.0 * pow (x, 2.0) + 12.0) * (exp (-1.0 * pow (x, 2.0))) );
}

/*************************************************************************/
/* Berechnung des Korrekturfaktors für Gleichverteilung                  */
/*************************************************************************/

void     gleichvert ()
{
double   y,
         radikant,
         wurzel;

         if (fabs (verteilung.a) < 1.0E-10)
                {
                sampar.ae = sampar.ac = 0.0;
                beep ();
                showerror (5);
                return;
                }

         if (fabs (sampar.af - sampar.ar) < 1.0E-10)
                {
                sampar.ae = verteilung.a / 2.0;
                sampar.ac = 1.0;
                return;
```

```
                }

        radikant = 1.0 - sampar.af - sampar.ar + sampar.af * sampar.ar;

        if (radikant < 0.0)
                {
                sampar.ae = sampar.ac = 0.0;
                beep ();
                showerror (5);
                return;
                }

        wurzel = sqrt (radikant);

        y = (1.0 - sampar.ar - wurzel) / (sampar.af - sampar.ar);

        if (fabs (y) < 1.0E-10)
                {
                sampar.ae = sampar.ac = 0.0;
                beep ();
                showerror (5);
                return;
                }
        sampar.ae = y * verteilung.a;
        sampar.ac = 1.0 / (2.0 * y);
return;
}

/*********************************************************************/
/* Berechnung des Korrekturfaktors für Laplaceverteilung             */
/*********************************************************************/

void    laplacevert ()
{
double static  y1 = 0.0,
               y2 = 0.0;
int            i  = 30000;

        if ((y2 < 0.0) || (y2 > 100))
                {
                y2 = 0.0;
                }

        if ((fabs (sampar.af - 1.0) < 1.0E-10) || (fabs (verteilung.ld) < 1.0E-10))
                {
                sampar.ae = sampar.ac = 0.0;
                beep ();
                showerror (5);
                return;
                }

        do{
        y1 = y2;
```

```
        y2 = (1.0 - sampar.af) + (sampar.af * verteilung.ld * y1)
              + (sampar.af - sampar.ar) * exp (-1.0 * verteilung.ld * y1);
        y2 /= verteilung.ld;
        i--;
        }while ((y1 != y2) && (i > 0));

        if ((i == 0) || (fabs (y2) < 1.0E-10))
              {
              beep ();
              showerror (6);
              if ((y2 < 0.0) || (y2 > 100))
                     {
                     y2 = 0.0;
                     sampar.ae = sampar.ac = 0.0;
                     return;
                     }
              }
        sampar.ae = y2;
        sampar.ac = sampar.ae * verteilung.ld;
        if (sampar.ac == 0.0)
              {
              sampar.ae = sampar.ac = 0.0;
              beep ();
              showerror (5);
              return;
              }
        sampar.ac = 1.0 / sampar.ac;
return;
}

/*********************************************************************/
/* Berechnung des Korrekturfaktors für Normalverteilung              */
/*********************************************************************/

void     normalvert ()
{
double         nw2, sw2,
               su1, su2;
double static  y1 = 0.0,
               y2 = 0.0;
int            i  = 100;

        if ((y2 < 0.0) || (y2 > 100))
              {
              y2 = 0.0;
              }

        nw2 =   sqrt (2.0);
        sw2 =   sqrt (2.0 / PI);
        do{
        y1  = y2;

        su1 = exp (-0.5 * y1 * y1);
```

```
        su2 = erf (y1 / nw2);

        y2  = 1.0 - (1.0 - su1) * sampar.af - (su1 * sampar.ar);
        y2 /= 1.0 - (1.0 - su2) * sampar.ar - (su2 * sampar.af);
        y2 *= sw2;
        i--;
        }while ((fabs(y1 - y2) > 1.0E-9) && (i > 0));

        if ((i == 0) || (fabs (y2) < 1.0E-10))
              {
              beep ();
              showerror (6);
              if ((y2 < 0.0) || (y2 > 100))
                     {
                     y2 = 0.0;
                     sampar.ae = sampar.ac = 0.0;
                     return;
                     }
              }
        sampar.ae = y2 * verteilung.sg;
        sampar.ac = sw2 / y2;
return;
}

/*********************************************************************/
/* Files kombinieren                                                 */
/* f3  *  (f1 * file1   +||*   f2 * file2)   =   file3               */
/*********************************************************************/

BOOL     combine ()
{
BOOL     combineselect ();
FILE     *fp   [3];
char     *filenameptr,
         *endptr,
         num                 [64],
         file         [3] [13] = {"","",""};
double   wert                [3],
         f                   [3]      = {1.0,1.0,1.0};
int      i,
         mode,
         samplength          [2];
long     l,
         filelength,
         minsampanz,
         sampanz             [2];
BOOL     error = FALSE;

         filenameptr = getfilename ();
         if (*filenameptr == 0x00)
              {
              return (FALSE);
              }
```

```
strcpy (&file[0][0], filenameptr);
filenameptr = getfilename ();
if (*filenameptr == 0x00)
      {
      return (FALSE);
      }
strcpy (&file[1][0], filenameptr);
strcpy (&file[2][0], ".DAT");

mode = option (OPTX, YREL, 2, "Files kombinieren:", "Additiv", "Multiplikativ");
if (mode == 0)
      {
      return (FALSE);
      }

if (!combineselect (&file[0][0], &file[1][0], &file[2][0], &f[0], &f[1], &f[2], mode))
      {
      return (FALSE);
      }

for (i = 0; i < 2; i++)
      {
      fp[i] = fopen (&file[i][0],"rb");
      if (fp[i] == NULL)
             {
             fcloseall ();
             beep ();
             showerror (1);
             return (FALSE);
             }
      fseek (fp[i],0L,SEEK_END);
      filelength = ftell (fp[i]);
      samplength[i] = getsamplength (fp[i]);
      if (samplength[i] == 0)
             {
             fcloseall ();
             beep ();
             showerror (3);
             return (FALSE);
             }
      sampanz[i] = (long) (filelength / samplength[i]);
      rewind (fp[i]);
      }
minsampanz = sampanz[0];
if (sampanz[1] < minsampanz)
      {
      minsampanz = sampanz[1];
      }
fp[2] = fopen (&file[2][0],"wb");
if (fp[2] == NULL)
      {
      fcloseall ();
      beep ();
```

```
                showerror (2);
                return (FALSE);
                }

        showpercent (0, 1);
        for (l = 0L; l < minsampanz; l++)
                {
                if ((l % 100L) == 0L)
                        {
                        showpercent ((int) (l * 100L / minsampanz), 0);
                        }

                for (i = 0; i < 2; i++)
                        {
                        fread (&num[0], samplength[i], 1, fp[i]);
                        wert[i] = strtod (&num[0], &endptr);
                        if ((wert[i] == 0.0) && ((*endptr != 0x0D)  && (*endptr != 0x20)))
                                {
                                error = TRUE;
                                break;
                                }
                        }
                        if (error == TRUE)
                                {
                                break;
                                }
                        if (mode == 1)
                                {
                                wert[2] = f[2] * (f[0] * wert[0] + f[1] * wert[1]);
                                }
                        else
                                {
                                wert[2] = f[2] * (f[0] * wert[0] * f[1] * wert[1]);
                                }
                        fprintf (fp[2], "%16.7e", wert[2]);
                        fputs ("\x0D\x0A", fp[2]);
                }

        fcloseall ();
        showpercent (0, 2);

        if (error == TRUE)
                {
                remove (&file[2][0]);
                beep ();
                showerror (3);
                return (FALSE);
                }

        shortbeep ();
        displaydata (&file[2][0],"",0);
return (TRUE);
}
```

Anhang C: Listing des Quellcodes von SPV (SPVFUNC.C)

```
/*********************************************************************/
/* Files verketten                                                   */
/*********************************************************************/

BOOL     chainfiles ()
{
BOOL     timeselect ();

FILE     *fp   [3];
char     *filenameptr,
         *endptr,
         num          [64],
         file  [3] [13] = {"","",""};
double   wert;
int      i,
         ret,
         samplength  [2];
long     l,
         filelength,
         maxsampanz,
         min          [2] = {0L,0L},
         max          [2],
         sampanz      [2];
BOOL     error = FALSE;

         filenameptr = getfilename ();
         if (*filenameptr == 0x00)
               {
               return (FALSE);
               }
         strcpy (&file[0][0], filenameptr);
         filenameptr = getfilename ();
         if (*filenameptr == 0x00)
               {
               return (FALSE);
               }
         strcpy (&file[1][0], filenameptr);

         for (i = 0; i < 2; i++)
               {
               fp[i] = fopen (&file[i][0],"rb");
               if (fp[i] == NULL)
                      {
                      fcloseall ();
                      beep ();
                      showerror (1);
                      return (FALSE);
                      }
               fseek (fp[i],0L,SEEK_END);
               filelength = ftell (fp[i]);
               samplength[i] = getsamplength (fp[i]);
               if (samplength[i] == 0)
```

```
                {
                fcloseall ();
                beep ();
                showerror (3);
                return (FALSE);
                }
        sampanz[i] = (long) (filelength / samplength[i]);
        rewind (fp[i]);
        }

max[0] = sampanz[0];
max[1] = sampanz[1];
if (!timeselect (2, &file[0][0], &file[1][0], &min[0], &max[0], &min[1], &max[1]))
        {
        fcloseall ();
        return (FALSE);
        }
sampanz[0] = max[0] - min[0] + 1L;
sampanz[1] = max[1] - min[1] + 1L;

ret = inputfilename ("DAT-Filename:", "DAT", ".xxx", &file[2][0]);
if (ret == 50)
        {
        fcloseall ();
        return (FALSE);
        }

fp[2] = fopen (&file[2][0],"wb");
if (fp[2] == NULL)
        {
        fcloseall ();
        beep ();
        showerror (2);
        return (FALSE);
        }

maxsampanz = sampanz[0] + sampanz[1];

showpercent (0, 1);
for (i = 0; i < 2; i++)
        {
        fseek  (fp[i], min[i] * (long) samplength[i], SEEK_SET);

        for (l = min[i]; l < max[i]; l++)
                {
                if ((l % 100L) == 0L)
                        {
        showpercent ((int) ( (l - min [i] + (long) i * sampanz[0]) * 100L / maxsampanz), 0);
                        }
                fread (&num[0], samplength[i], 1, fp[i]);
                wert = strtod (&num[0], &endptr);
                if ((wert == 0.0) && ((*endptr != 0x0D)  && (*endptr != 0x20)))
                        {
```

```
                                error = TRUE;
                                break;
                                }
                        if (error == TRUE)
                                {
                                break;
                                }
                        fprintf (fp[2], "%16.7e", wert);
                        fputs ("\x0D\x0A", fp[2]);
                        }
                }

        fcloseall ();
        showpercent (0, 2);

        if (error == TRUE)
                {
                remove (&file[2][0]);
                beep ();
                showerror (3);
                return (FALSE);
                }

        shortbeep ();
        displaydata (&file[2][0],"",0);
return (TRUE);
}



/*********************************************************************/
/* File bearbeiten                                                   */
/*********************************************************************/

BOOL    norm (char *filename)
{
FILE            *fp,
                *tp;
char            out_file        [13] = "",
                num             [64] = "",
                *endptr;
double          wert            = 0.0,
                wert_1          = 0.0,
                normleist       = 1.0,
                mittelwert      = 0.0,
                varianz = 0.0,
                max             = 0.0;
double static   normfaktor      = 1.0,
                mulfaktor       = 1.0,
                summand = 0.0;
long            filelength,
                sampanz,
                i;
```

Anhang C: Listing des Quellcodes von SPV (SPVFUNC.C)

```
int             ret,
                mode,
                samplength;
BOOL            newfile = FALSE,
                error   = FALSE;

          fp = fopen (filename,"rb");
          if (fp == NULL)
                {
                beep ();
                showerror (1);
                return (FALSE);
                }
          fseek (fp, 0L, SEEK_END);
          filelength = ftell (fp);
          samplength = getsamplength (fp);
          if (samplength == 0)
                {
                fclose (fp);
                beep ();
                showerror (3);
                return (FALSE);
                }
          sampanz = (long) (filelength / samplength);
          if (sampanz == 0L)
                {
                fclose (fp);
                beep ();
                showerror (3);
                return (FALSE);
                }

          ret = option (OPTX, YREL, 6, "File bearbeiten:",
          "Mittelwert einstellen",
          "Sigma einstellen",
          "Amplitude normieren",
          "Multiplizieren",
          "Filter",
          "Filter (Sigma konstant)");

          if (ret == 0)
                {
                fclose (fp);
                return (FALSE);
                }
          else
                {
                mode = ret;
                }

          switch (mode)
                {
                case 1:
```

```
        ret = inputnumber ("Neuer Mittelwert:", &summand);
        if (ret == 50)
                {
                fclose (fp);
                return (FALSE);
                }
        break;

        case 2:
        ret = inputnumber ("Neues Sigma:", &normfaktor);
        if ((ret == 50) || (normfaktor <= 0.0))
                {
                fclose (fp);
                return (FALSE);
                }
        break;

        case 3:
        ret = inputnumber ("Normieren auf:", &normfaktor);
        if ((ret == 50) || (normfaktor <= 0.0))
                {
                fclose (fp);
                return (FALSE);
                }
        break;

        case 4:
        ret = inputnumber ("Multiplizieren mit:", &mulfaktor);
        if ((ret == 50) || (mulfaktor == 0.0))
                {
                fclose (fp);
                return (FALSE);
                }
        break;

        case 6:
        if ((sirpfil.bs >= 1.0) || (sirpfil.bs < 0.0) || (sirpfil.as == 0.0))
                {
                showerror (9);
                }
        else
                {
                normleist = sqrt (1.0 - (sirpfil.bs * sirpfil.bs)) / sirpfil.as;
                }
        break;

        default:
        break;
        }

    ret = inputfilename ("DAT-Filename:", "DAT", filename, out_file);
    if (ret == 50)
        {
```

```
        fclose (fp);
        return (FALSE);
        }
if (out_file[0] == 0x00)
        {
        strcpy (out_file, filename);
        }

showpercent (0, 1);

for (i = 0L; i < sampanz; i++)
        {
        if ((i % 100L) == 0L)  /* 0% - 33% */
                {
                showpercent ((int) (i * 33L / sampanz), 0);
                }

        if(!fread (&num[0],samplength,1,fp))
                {
                break;
                }
        wert = 0.0;
        wert = strtod (&num[0],&endptr);
        if ((wert == 0.0) && ((*endptr != 0x0D)  && (*endptr != 0x20)))
                {
                error = TRUE;
                break;
                }
        mittelwert += wert;
        varianz    += wert * wert;

        if (fabs(wert) > max)
                {
                max = fabs(wert);
                }
        }

switch (mode)
        {
        case 1:
        mittelwert /= (double) sampanz;
        summand    -= mittelwert;
        break;

        case 2:
        mittelwert /= (double) sampanz;
        varianz    /= (double) sampanz;
        varianz    -= mittelwert * mittelwert;
        max = sqrt (varianz) / normfaktor;
        if (max <= 0.0)
                {
                error = TRUE;
                }
```

```
            break;

            case 3:
            max /= normfaktor;
            if (max <= 0.0)
                    {
                    error = TRUE;
                    }
            break;

            default:
            break;
            }

      if (error)
            {
            fclose (fp);
            showpercent (0, 2);
            showerror (3);
            return (FALSE);
            }

      rewind (fp);

      if (stricmp (filename, out_file) == 0)
            {
            tp = fopen (TEMPDATEI, "wb");
            newfile = FALSE;
            }
      else
            {
            tp = fopen (out_file, "wb");
            newfile = TRUE;
            }

      if (tp == NULL)
            {
            fclose (fp);
            showpercent (0, 2);
            beep ();
            showerror (2);
            return (FALSE);
            }

      for (i = 0L; i < sampanz; i++)
            {
            if ((i % 100L) == 0L)  /* 33% - 100% */
                    {
                    showpercent ((int) (i * 67L / sampanz + 33L), 0);
                    }

            fread (&num[0],samplength,1,fp);
            wert = atof (num);
```

```
                switch (mode)
                        {
                case 1:
                        wert += summand;
                        break;

                        case 2:
                        case 3:
                        wert /= max;
                        break;

                        case 4:
                        wert *= mulfaktor;
                        break;

                        default:
                        wert   = sirpfil.as * wert + sirpfil.bs * wert_1;
                        wert_1 = wert;
                        if (mode == 6)
                                {
                                wert *= normleist;
                                }
                        break;
                        }

                fprintf (tp, "%16.7e", wert);
                fputs ("\x0D\x0A", tp);
                }

          fclose (fp);
          fclose (tp);

          if (!newfile)
                {
                remove (filename);
                rename (TEMPDATEI, filename);
                }

          showpercent (0, 2);
          shortbeep ();
          if (newfile)
                {
                displaydata (filename, out_file, 0);
                }
          else
                {
                displaydata (filename, "", 0);
                }
return (TRUE);
}
```

SPVEDIT.C

Anhang C: Listing des Quellcodes von SPV (SPVEDIT.C)

```
/*********************************************************************/
/* SPV                                                               */
/* Autor: Harald Zottmann                                            */
/* Matr.: 787378                                                     */
/* Studienarbeit 1992/93                                             */
/* Technische-Hochschule-Darmstadt                                   */
/*********************************************************************/

#define  FALSE 0
#define  TRUE         1

#define  BOOL         unsigned char


#include <dos.h>
#include <conio.h>
#include <string.h>
#include <graphics.h>

extern   BOOL         nobeep;

/*********************************************************************/
/* String auf Dezimal testen                                         */
/*********************************************************************/

BOOL     strdez (str)
char     *str;

{
int i;

         for (i = 0; i < strlen(str); i++)
               {
               if ( (*(str+i) < 0x30)  || (*(str+i) > 0x39) )
                      {
                      return (FALSE);
                      }
               }
return (TRUE);
}

/*********************************************************************/
/* String auf Fließkommazahl testen                                  */
/*********************************************************************/

BOOL     strfloat(str)
char     *str;

{
int i;

         for (i = 0; i < strlen(str); i++)
               {
```

```
                if ( ((*(str+i) < 0x30)  || (*(str+i) > 0x39)) && (*(str+i) != '.') )
                        {
                        return (FALSE);
                        }
                }
return (TRUE);
}

/*********************************************************************/
/*                        Standard-Eingaberoutine                    */
/*                                                                   */
/* Rückgabewerte:  0  RETURN OHNE EINGABE                            */
/*                 1  RETURN NACH EINGABE oder CURSOR RUNTER         */
/*                -1  CURSOR HOCH                                    */
/*                 2  BILD RUNTER  /  -2  BILD HOCH                  */
/*                50  ESCAPE                                         */
/*********************************************************************/

int      input(int x,int y,int maxlaenge,char *str,BOOL insert)
{
void     beep ();

int             laenge,
                pos = 0,
                i;
unsigned char   c;


          laenge = strlen (str);
          for (i = laenge; i <= maxlaenge; i++)
                {
                *(str+i) = 0x00;
                }

          setviewport (x*8, y*10, (x+maxlaenge)*8, y*10+8, 1);
          clearviewport ();
          setlinestyle (SOLID_LINE, 0xFFFF, THICK_WIDTH);
          setwritemode (XOR_PUT);
          setcolor (WHITE);

          do{
                clearviewport ();
                outtextxy (0, 0, str);
                line (pos*8, 0, pos*8, 8);
                line (pos*8+3, 0, pos*8+3, 8);
          line (pos*8+6, 0, pos*8+6, 8);
                switch (c = getch ())
                        {
                        case 0x00:
                        switch (getch ())
                                {
          /* Cursor Down */    case 'P':
                                clearviewport ();
```

```
                        setcolor (DARKGRAY);
                        outtextxy (0, 0, str);
                        return (1);

/* Cursor Up */                case 'H':
                        clearviewport ();
                        setcolor (DARKGRAY);
                        outtextxy (0, 0, str);
                        return (-1);

/* Cursor Left */       case 'K':
                        if (pos > 0)
                                {
                                pos--;
                                }
                        break;

/* Cursor Right */      case 'M':
                        if ((pos < maxlaenge) && (pos < strlen(str)) )
                                {
                                pos++;
                                }
                        break;

/* Bild Up */           case 'I':
                        clearviewport ();
                        setcolor (DARKGRAY);
                        outtextxy (0, 0, str);
                        return (-2);

/* Bild Down */                case 'Q':
                        clearviewport ();
                        setcolor (DARKGRAY);
                        outtextxy (0, 0, str);
                        return (2);

/* Einfügen */                 case 'R':
                        insert = !insert;
                        break;

/* Delete */            case 'S':
                        if (pos >= maxlaenge)
                                {
                                break;
                                }
                        for (i = pos; i < maxlaenge; i++)
                                {
                                *(str+i) = *(str+i+1);
                                }
                        laenge = strlen (str);
                        break;

/* F10 */               case 0x44:
```

```
                                clearviewport ();
                                setcolor (DARKGRAY);
                                outtextxy (0, 0, str);
                                return (20);

                                default:
                                continue;
                                }
                        break;

/* Backspace */case 0x08:
                        if (pos)
                                {
                                for (i = pos; i <= maxlaenge; i++)
                                *(str+i-1) = *(str+i);
                                laenge = strlen(str);
                                pos--;
                                }
                        break;

/* Escape */            case 0x1B:
                        clearviewport ();
                        setcolor (DARKGRAY);
                        outtextxy (0, 0, str);
                        return(50);

/* Return */            case 0x0D:
                        clearviewport ();
                        setcolor (DARKGRAY);
                        outtextxy (0, 0, str);
                        if (strlen(str))
                                {
                                return (1);
                                }
                        else
                                {
                                return (0);
                                }

                        default:
                        if ( ((c < 32) || (c > 155)) && (c != 225) )
                                {
                                beep();
                                break;  /* Nicht darstellbare Zeichen */
                                }
                        if (insert)
                                {
                                if (strlen(str) < maxlaenge)
                                        {
                                        for (i = maxlaenge-1; i >= pos; i--)
                                        *(str+i+1) = *(str+i);
                                        *(str+pos) = c;
                                        pos++;
```

```
                                        }
                                if      (pos > maxlaenge)
                                        {
                                        pos--;
                                        }
                                }
                        else if (pos < maxlaenge)
                                {
                                *(str+pos) = c;
                                pos++;
                                }
                        }
        }while (TRUE);
}

void    beep ()
{
        if (nobeep)
              {
              return;
               }
        sound (440);
        delay (100);
        nosound ();
return;
}

void    shortbeep ()
{
        if (nobeep)
              {
              return;
              }
        sound (880);
        delay (50);
        nosound ();
return;
}
```

SPVSIRP.C

Anhang C: Listing des Quellcodes von SPV (SPVSIRP.C)

```
/*********************************************************************/
/* SPV                                                               */
/* Autor: Harald Zottmann                                            */
/* Matr.: 787378                                                     */
/* Studienarbeit 1992/93                                             */
/* Technische-Hochschule-Darmstadt                                   */
/*********************************************************************/

#include "SPVINC.H"

#include "const.h"
#include "fft.h"
#include "heap.h"
#include "sirp.h"

#include <time.h>

extern   SIRPFILTER   sirpfil;
extern   VERTEILUNG   verteilung;

double         *akf1;
int            anzakf1;
filkof         tp      = {NULL,NULL,0,0},
               ff1     = {NULL,NULL,0,0};
sirpdatensirp1         = {NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL};

/*********************************************************************/
/* Sirp erzeugen                                                     */
/*********************************************************************/

BOOL     makesirp ()
{
FILE           *fp;
char           c,
               num            [20] = "",
               defaultname    [13] = "",
               filename       [13] = "";
int            ret;
long           length         = 2048L,
               k;
double         *e1,
               rd1, rd2,
               wert           = 0.0,
               wert_1         = 0.0,
               normleist      = 1.0;
BOOL           filter_on      = FALSE;


         ret = option (OPTX, YREL, 8, "Verteilung:", "Gleichverteilung  (FIL)", "Gleichverteilung",
                                                     "Laplaceverteilung (FIL)", "Laplaceverteilung",
                                                     "Gaußverteilung    (FIL)", "Gaußverteilung",
                                                     "Gaußverteilung    (SIRP)",
                                                     "Ko-Verteilung     (SIRP)");
```

```
switch (ret)
      {
      case 1:
      c = 'W';
      filter_on = TRUE;
      strcpy (defaultname, "GLEICH_F.xxx");
      break;

      case 2:
      c = 'W';
      strcpy (defaultname, "GLEICH.xxx");
      break;

      case 3:
      c = 'L';
      filter_on = TRUE;
      strcpy (defaultname, "LAPLAC_F.xxx");
      break;

      case 4:
      c = 'L';
      strcpy (defaultname, "LAPLAC.xxx");
      break;

      case 5:
      c = 'N';
      filter_on = TRUE;
      strcpy (defaultname, "GAUSS_F.xxx");
      break;

      case 6:
      c = 'N';
      strcpy (defaultname, "GAUSS.xxx");
      break;

      case 7:
      c = 'G';
      strcpy (defaultname, "GAUSS.xxx");
      break;

      case 8:
      c = 'K';
      strcpy (defaultname, "KO.xxx");
      break;

      default:
      return (FALSE);
      }

if (filter_on)
      {
      if ((sirpfil.bs >= 1.0) || (sirpfil.bs < 0.0) || (sirpfil.as == 0.0))
             {
```

```
                showerror (9);
                }
        else
                {
                normleist = sqrt (1.0 - (sirpfil.bs * sirpfil.bs)) / sirpfil.as;
                }
        }

ret = option (OPTX, YREL, 7, "Filelänge:","2048","4096","8192","16384","32768","65536","Andere Länge");
    switch (ret)
        {
        case 1: length = 2048L;  break;
        case 2:        length = 4096L;  break;
        case 3: length = 8192L;  break;
        case 4: length = 16384L; break;
        case 5:        length = 32768L; break;
        case 6:        length = 65536L; break;

        case 7:
        setviewport (0, 6*10, 26*8, 9*10, 1);
        setcolor (BLUE);
        rectangle (0, 0, 26*8, 3*10);
        setcolor (YELLOW);
        outtextxy (3*8, 1*10, "Filelänge:");
        sprintf (num, "%ld", length);
        ret = input(3, 8, 18, num, 1);

        if ((ret == 0) || (ret == 50))
                {
                return (FALSE);
                }
        else
                {
                length = atol (num);
                }
        if ((length < 128L) || (length >  1048576L))
                {
                beep ();
                showerror (10);
                return (FALSE);
                }
        setviewport (0, 6*10, 26*8, 9*10, 1);
        clearviewport ();
        break;

        default: return (FALSE);
        }

    if ((c == 'G') || (c == 'K'))
        {
        ret = inputfilename ("SRP-Filename:", "SRP", defaultname, filename);
        if (ret == 50)
                {
```

```
                return (FALSE);
                }
        }
else
        {
        ret = inputfilename ("DAT-Filename:", "DAT", defaultname, filename);
        if (ret == 50)
                {
                return (FALSE);
                }

         }

fp = fopen (filename, "wb");
if (fp == NULL)
        {
        beep ();
        showerror (2);
        return (FALSE);
        }

if ((c == 'G') || (c == 'K'))
        {
        busy (TRUE);

        e1   = dvek_alloc (8);
        akf1 = dvek_alloc (anzakf1 = 6);
        akf1[0] = 1.0;  akf1[1] = 0.7; akf1[2] = 0.3;
        akf1[3] = 0.2;  akf1[4] = 0.1; akf1[5] = 0.05;
        tp.a = dvek_alloc (tp.anz_a = 5);
        tp.a[0] =  1.0;
        tp.a[1] = -3.979476688;
        tp.a[2] =  5.9386404051;
        tp.a[3] = -3.9388494855;
        tp.a[4] =  0.97968577219;
        tp.b = dvek_alloc (tp.anz_b = 5);
        tp.b[0] =  4.647337628e-9;
        tp.b[1] =  1.858935051e-8;
        tp.b[2] =  2.788402577e-8;
        tp.b[3] =  tp.b[1];
        tp.b[4] =  tp.b[0];

        bl_filter (127, akf1, anzakf1, &ff1);
        sirp_init (&sirp1, ff1, tp);

        busy (FALSE);
        showpercent (0, 1);

        for (k = 0; k < length; k++)
                {
                if ((k % 100L) == 0L)
                        {
                        showpercent ((int) (k * 100L / length), 0);
```

```
                        }
                wert = sirp (e1, &sirp1, ff1, tp, c, NEUERWERT);
                fprintf (fp, "%16.7e", wert);
                fputs ("\x0D\x0A", fp);
                }
        sirp_free (&sirp1);
        }
    else
        {
        randomize ();

        showpercent (0, 1);

        for (k = 0; k < length; k++)
                {
                if ((k % 100L) == 0L)
                        {
                        showpercent ((int) (k * 100L / length), 0);
                        }
                if      (c == 'L')
                        {
                        wert = (double) rand () / 16383.5 - 1.0;
                        if (wert < 0)
                                {
                                wert =  1.0 / verteilung.ld * log (-1.0 * wert);
                                }
                        else
                                {
                                wert = -1.0 / verteilung.ld * log (wert);
                                }
                        }
                else if (c == 'N')
                        {
                        rd1 = (double) rand () / RAND_MAX;
                        rd2 = (double) rand () / RAND_MAX;

                        if (rd1 != 0.0)
                                {
                                wert = verteilung.sg * sqrt (-2.0*log (rd1))*cos (PI - PI2 * rd2);
                                }
                        }
                else
                        {
                        wert = ((double) rand () / 16383.5 - 1.0) * verteilung.a;
                        }

                if (filter_on)
                        {
                        wert   = sirpfil.as * wert + sirpfil.bs * wert_1;
                        wert_1 = wert;
                        wert  *= normleist;
                        }
                fprintf (fp, "%16.7e", wert);
```

```
                        fputs ("\x0D\x0A", fp);
                        }
                }

          fclose (fp);

          showpercent (0, 2);

          histogramm (filename);

return (TRUE);
}
```

FILEREQ.C

Anhang C: Listing des Quellcodes von SPV (FILEREQ.C)

```
/*********************************************************************/
/* SPV                                                               */
/* Autor: Harald Zottmann                                            */
/* Matr.: 787378                                                     */
/* Studienarbeit 1992/93                                             */
/* Technische-Hochschule-Darmstadt                                   */
/*********************************************************************/

#define  FALSE        0
#define  TRUE                 1

#define  BOOL                 unsigned char

#define  XREL                 0
#define  YREL                 6
#define  SHOWMAX              20

/*********************************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <dir.h>
#include <dos.h>
#include <mem.h>
#include <alloc.h>
#include <graphics.h>

BOOL     savegraph ();

/*********************************************************************/
/* Filerequester                                                     */
/*********************************************************************/

char     *getfilename ()
{
BOOL     getdirectory  ();
void     getfilereqpos ();


static   char          filename       [13] = "",
                       pathname       [13] = "*.*";
struct   ffblk ffblk,
                       *ptr_ffblk;

char                   alt_dir [MAXDIR],
                       neu_dir [MAXDIR],
                       num            [20],
                       str            [50];

char                   *fn_ptr = NULL;
unsigned int           fn_size,
```

```
                        ff_size = sizeof (struct ffblk),
                        drive;
static   int   art            = 0,    /* 0-FILES / 1-DIRECTORIES */
               pos            = 0,
                        foffset = 0,
                        doffset        = 0;

int                     mode    = 0,    /* 0-SELECT / 1-OPEN / 2-ESC */
                        danz           = 0,    /* Directoryeinträge */
                        da,
                        fanz           = 0,    /* Fileeinträge */
                        fa,
                        selected       = 0,    /* Ausgewähltes File / Directory */
                        i;

          ptr_ffblk = &ffblk;

          getdirectory (&fanz, &danz, &drive, &alt_dir[0], &pathname[0],
                      &ffblk, &fn_size, ff_size, &fn_ptr);

          setviewport (XREL*8, YREL*10, (XREL+48)*8, (YREL+SHOWMAX+8)*10, 1);
          clearviewport ();

          while (TRUE)
                {
                setcolor (BLUE);
                setlinestyle (SOLID_LINE, 0xFFFF, THICK_WIDTH);
                setwritemode (COPY_PUT);
                rectangle ( 0*8, 0*10, 47*8, (SHOWMAX+7)*10);
                rectangle ( 1*8, 5*10, 23*8, (SHOWMAX+6)*10);
                rectangle (26*8, 5*10, 46*8, (SHOWMAX+6)*10);
                setcolor (YELLOW);
                outtextxy ( 2*8, 1*10,"DRIVE:");
                setcolor (RED);
                sprintf (num, "%c:", drive + 'A');
                outtextxy (13*8, 1*10, num);
                setcolor (YELLOW);
                outtextxy ( 2*8, 2*10, "DIRECTORY:");
                setcolor (RED);
                outtextxy (13*8, 2*10, alt_dir);
                setcolor (YELLOW);
                outtextxy (2*8, 4*10, "FILES:");
                setcolor (RED);
                sprintf (num, "%d", fanz);
                outtextxy (13*8, 4*10, num);
                setcolor (YELLOW);
                outtextxy (27*8, 4*10, "DIRECTORIES:");
                setcolor (RED);
                sprintf (num, "%d", danz);
                outtextxy (40*8, 4*10, num);

                fa = da = 0;
                for (i = 0; i < fanz+danz; i++)
```

```
                {
                memcpy (&ffblk, fn_ptr+i*ff_size, ff_size);

                if (ptr_ffblk->ff_attrib != FA_DIREC)
                        {
                        if ((fa-foffset >= SHOWMAX) || (fa < foffset))
                                {
                                fa++;
                                continue;
                                }
                        if ((pos == fa) && (art == 0))
                                {
                                selected = i;
                                setcolor (WHITE);
                                }
                        else
                                {
                                setcolor (DARKGRAY);
                                }
                        sprintf (str, "%-12s", ptr_ffblk->ff_name);
                        sprintf (num, "%-7ld", ptr_ffblk->ff_fsize);
                        strcat (str," ");
                        strcat (str, num);
                        outtextxy (2*8, (6+fa-foffset)*10, str);
                        fa++;
                        }
                else
                        {
                        if ((da-doffset >= SHOWMAX) || (da < doffset))
                                {
                                da++;
                                continue;
                                }
                        if ((pos == da) && (art == 1))
                                {
                                selected = i;
                                setcolor (WHITE);
                                }
                        else
                                {
                                setcolor (DARKGRAY);
                                }
                        sprintf (str, "%-12s", ptr_ffblk->ff_name);
                        strcat (str, " <DIR>");
                        outtextxy (27*8, (6+da-doffset)*10, str);
                        da++;
                        }
                }
        getfilereqpos (&pos, &foffset, &doffset, &mode, &art, fanz, danz);
        if (mode == 1)                /* open / chdir */
                {
                if (art == 0)         /* Files */
                        {
```

```
                                memcpy (&ffblk, fn_ptr+selected*ff_size, ff_size);
                                strcpy (&filename[0], ptr_ffblk->ff_name);
                                free (fn_ptr);
                                clearviewport ();
                                return (&filename[0]);
                                }
                        else
                                {
                                memcpy (&ffblk, fn_ptr+selected*ff_size, ff_size);
                                strcpy (&neu_dir[0], ptr_ffblk->ff_name);
                                free (fn_ptr);
                                chdir (&neu_dir[0]);
                                getdirectory (&fanz, &danz, &drive, &alt_dir[0], &pathname[0],
                                              &ffblk, &fn_size, ff_size, &fn_ptr);
                                pos = foffset = doffset = 0;
                                clearviewport ();
                                }
                        }
                if (mode == 2)                /* Esc */
                        {
                        free (fn_ptr);
                        clearviewport ();
                        memset (&filename[0],0x00,13);
                        return (&filename[0]);
                        }
                }
}

/*********************************************************************/

BOOL     getdirectory (int *fanz, int *danz, unsigned int *drive, char *alt_dir, char *pathname,
                    struct ffblk *ffblk, unsigned int *fn_size, unsigned int ff_size, char **fn_ptr)
{
void     sortdirectory ();

         *fanz = 0;
         *danz = 0;

         *drive = getdisk ();
         getcurdir (0, alt_dir);

         if (!findfirst (pathname,ffblk, FA_RDONLY|FA_DIREC))
               {
               *fn_size = ff_size;
               *fn_ptr = malloc (*fn_size);
               if (*fn_ptr == NULL)
                       {
                       return (FALSE);
                       }
               memcpy (*fn_ptr, ffblk, ff_size);
               if (ffblk->ff_attrib == FA_DIREC)     (*danz)++;
               else                                  (*fanz)++;
               }
```

```
        while   (!findnext (ffblk))
                {
                *fn_size += ff_size;
                *fn_ptr = realloc (*fn_ptr, *fn_size);
                if (*fn_ptr == NULL)
                        {
                        return (FALSE);
                        }
                memcpy ((*fn_ptr)+(*fn_size)-ff_size, ffblk, ff_size);
                if (ffblk->ff_attrib == FA_DIREC)     (*danz)++;
                else                                  (*fanz)++;
                }

        sortdirectory (*fn_ptr, *fn_size, ff_size);
return (TRUE);
}

/*********************************************************************/

void    sortdirectory (char *fn_ptr, unsigned int fn_size, unsigned int ff_size)
{
struct          ffblk           ffblk           [2],
                                *ffblk_ptr      [2];
unsigned int    i, anz;
BOOL            ok = TRUE;;

        ffblk_ptr[0] = &ffblk[0];
        ffblk_ptr[1] = &ffblk[1];

        anz = fn_size / ff_size;

        while (ok)
                {
                ok = FALSE;
                for (i = 1; i < anz; i++)
                        {
                        memcpy (ffblk_ptr[0], fn_ptr+(i-1)*ff_size, ff_size);
                        memcpy (ffblk_ptr[1], fn_ptr+(i  )*ff_size, ff_size);
                        if (strcmp (ffblk_ptr[0]->ff_name, ffblk_ptr[1]->ff_name) > 0)
                                {
                                memcpy (fn_ptr+(i-1)*ff_size, ffblk_ptr[1], ff_size);
                                memcpy (fn_ptr+(i  )*ff_size, ffblk_ptr[0], ff_size);
                                ok = TRUE;
                                }
                        }
                }
return;
}

/*********************************************************************/

void    getfilereqpos (int *pos, int *foffset, int *doffset,
```

```
                        int *mode, int *art, int fanz, int danz)
{
char      c;

          *mode = 0;

          c = getch ();
          switch (c)
                {
                case '\x0':
                switch (getch ())
                        {
                        case 'P':      /* Cursor Down */
                        (*pos)++;
                        if (*art == 0)
                                {
                                if (*pos >= fanz)
                                        {
                                        (*pos)--;
                                        }
                                if (*pos >= *foffset+SHOWMAX)
                                        {
                        (*foffset)++;
                        setviewport ( (XREL+1)*8, (YREL+5)*10, (XREL+23)*8, (YREL+SHOWMAX+6)*10, 1);
                        clearviewport ();
                        setviewport (XREL*8, YREL*10, (XREL+48)*8, (YREL+SHOWMAX+8)*10, 1);
                                        }
                                }
                        else
                                {
                                 if (*pos >= danz)
                                        {
                                        (*pos)--;
                                        }
                                if (*pos >= *doffset+SHOWMAX)
                                        {
                        (*doffset)++;
                        setviewport ((XREL+26)*8, (YREL+5)*10, (XREL+46)*8, (YREL+SHOWMAX+6)*10, 1);
                        clearviewport ();
                        setviewport (XREL*8, YREL*10, (XREL+48)*8, (YREL+SHOWMAX+8)*10, 1);
                                        }
                                }
                        break;

                        case 'H':      /* Cursor Up */
                        (*pos)--;
                        if (*art == 0)
                                {
                                if (*pos < 0)
                                        {
                                        (*pos)++;
                                        }
                                if (*pos < *foffset)
```

```
                        {
                (*foffset)--;
                setviewport ( (XREL+1)*8, (YREL+5)*10, (XREL+23)*8, (YREL+SHOWMAX+6)*10, 1);
                clearviewport ();
                setviewport (XREL*8, YREL*10, (XREL+48)*8, (YREL+SHOWMAX+8)*10, 1);
                        }
                }
                else
                        {
                         if (*pos < 0)
                                {
                                (*pos)++;
                                }
                        if (*pos < *doffset)
                                {
                (*doffset)--;
                setviewport ((XREL+26)*8, (YREL+5)*10, (XREL+46)*8, (YREL+SHOWMAX+6)*10, 1);
                clearviewport ();
                setviewport (XREL*8, YREL*10, (XREL+48)*8, (YREL+SHOWMAX+8)*10, 1);
                                }
                        }
                break;

                default:
                break;
                }
        break;

        case 0x67:              /* G */
        savegraph ((unsigned int) 349);
        *mode = 2;
        break;

        case 0x1B:              /* Escape */
        *mode = 2;
        break;

        case 0x09:              /* Tabulator */
        if (*art == 0)
                {
                *art = 1;       /* DIRS */
                *pos = (*pos) - (*foffset) + (*doffset);
                if (*pos >= danz)
                        {
                        *pos = danz - 1;
                        }

                }
        else
                {
                if (fanz == 0) /* Directory ist leer */
                        {
                        break;
```

```
                        }
                *art = 0;       /* FILES */
                *pos = (*pos) - (*doffset) + (*foffset);
                if (*pos >= fanz)
                        {
                        *pos = fanz - 1;
                        }
                }
        break;

        case 0x0D:              /* Return */
        *mode = 1;
        break;

        default:
        break;
        }
return;
}
```

Anhang D

Listing des Quellcodes von SPVGEN

Anhang D: Listing des Quellcodes von SPVGEN

```
/*********************************************************************/
/* SPV                                                               */
/* Autor: Harald Zottmann                                            */
/* Matr.: 787378                                                     */
/* Studienarbeit 1992/93                                             */
/* Technische-Hochschule-Darmstadt                                   */
/*********************************************************************/

#define  FALSE 0
#define  TRUE         1
#define  PI           3.14159265359

/*********************************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <dir.h>
#include <dos.h>
#include <math.h>

/*********************************************************************/
/*********************************************************************/

void     main (int argc, char *argv[])
{
FILE           *fp            = NULL;
char           *ptr           = NULL,
               fname [128]    = "SPVGEN.DAT";        /* Defaultfilename */
double         wert           = 0.0,
               step1          = 0.0,                 /* Steigungen für RAMP */
               step2          = 0.0,
               offset  = 0.0,                 /* Summand für alle Werte */
               alpha          = 0.01;        /* Dämpfungsfaktor für E1|E2 */
int            siwert         = 0;                   /* Laufvariable für SI|E1|E2 */
unsigned int   i              = 0,
               x              = 0,
               y              = 0,
               mode           = 0,                   /* Signalart */
               tast1          = 10,                  /* Tastverhältnis */
               tast2          = 10,                  /* Tastverhältnis */
               sigfreq = 100,                 /* Signalfrequenz */
               sampfreq       = 24000,               /* Samplefrequenz */
               sampanz = 8192;        /* Anzahl der Werte */

         while (argc > 1)
               {
               if      (stricmp (argv[argc-1], "SIN") == 0)
                       {
                       mode = 1;
                       }
               else if (stricmp (argv[argc-1], "COS") == 0)
                       {
```

```
        mode = 2;
        }
else if (stricmp (argv[argc-1], "SI") == 0)
        {
        mode = 3;
        }
else if (stricmp (argv[argc-1], "S01") == 0)
        {
        mode = 4;
        }
else if (stricmp (argv[argc-1], "S10") == 0)
        {
        mode = 5;
        }
else if (stricmp (argv[argc-1], "RAMP") == 0)
        {
        mode = 6;
        }
else if (stricmp (argv[argc-1], "RECT") == 0)
        {
        mode = 7;
        }
else if (stricmp (argv[argc-1], "E1") == 0)
        {
        mode = 8;
        }
else if (stricmp (argv[argc-1], "E2") == 0)
        {
        mode = 9;
        }
else if (strnicmp (argv[argc-1], "SGFREQ:",7) == 0)
        {
        ptr = (char *) argv[argc-1];
        ptr += 7;
        sigfreq = (unsigned int) atol (ptr);
        }
else if (strnicmp (argv[argc-1], "SPFREQ:",7) == 0)
        {
        ptr = (char *) argv[argc-1];
        ptr += 7;
        sampfreq = (unsigned int) atol (ptr);
        }
else if (strnicmp (argv[argc-1], "T1:",3) == 0)
        {
        ptr = (char *) argv[argc-1];
        ptr += 3;
        tast1 = (unsigned int) atol (ptr);
        }
else if (strnicmp (argv[argc-1], "T2:",3) == 0)
        {
        ptr = (char *) argv[argc-1];
        ptr += 3;
        tast2 = (unsigned int) atol (ptr);
```

```
                }
        else if (strnicmp (argv[argc-1], "ALPHA:",6) == 0)
                {
                ptr = (char *) argv[argc-1];
                ptr += 6;
                alpha = (double) atof (ptr);
                alpha = -1.0 * fabs (alpha);
                }
        else if (strnicmp (argv[argc-1], "OFFSET:",7) == 0)
                {
                ptr = (char *) argv[argc-1];
                ptr += 7;
                offset = (double) atof (ptr);
                }
        else if (strnicmp (argv[argc-1], "LENGTH:",7) == 0)
                {
                ptr = (char *) argv[argc-1];
                ptr += 7;
                sampanz = (unsigned int) atol (ptr);
                }
        else if (strnicmp (argv[argc-1], "NAME:",5) == 0)
                {
                ptr = (char *) argv[argc-1];
                ptr += 5;
                strcpy (fname, ptr);
                }
        else
                {
printf ("\r\n\nSPVGEN V1.0  -  Autor: Harald Zottmann  -  Datum: 1993\r\n\n");
printf ("SPVGEN [SIN|COS]  [SGFREQ:] [SPFREQ:] [OFFSET:] [LENGTH:] [NAME:]\r\n");
printf ("       [SI]       [SGFREQ:] [SPFREQ:] [OFFSET:] [LENGTH:] [NAME:]\r\n");
printf ("       [S01|S10]  [T1: T2:]           [OFFSET:]           [NAME:]\r\n");
printf ("       [RAMP]     [T1: T2:]           [OFFSET:] [LENGTH:] [NAME:]\r\n");
printf ("       [RECT]     [T1: T2:]           [OFFSET:] [LENGTH:] [NAME:]\r\n");
printf ("       [E1|E2]    [ALPHA:]            [OFFSET:] [LENGTH:] [NAME:]\r\n\n");
printf ("SIN|COS            Sinusfunktion oder Cosinusfunktion\r\n");
printf ("SI                 SI-Funktion\r\n");
printf ("S01|S10            Sprungfunktion 0 -> 1 oder 1 -> 0\r\n");
printf ("RAMP               Rampenfunktion\r\n");
printf ("RECT               Rechteckfolge\r\n");
printf ("E1|E2              Normale oder quadratische Exponentialfuntion\r\n\n");
printf ("SGFREQ             Signalfrequenz\r\n");
printf ("SPFREQ             Samplefrequenz\r\n");
printf ("ALPHA              Dämpfungsfaktor\r\n");
printf ("T1/T2              Tastverhältnis\r\n");
printf ("OFFSET             Summand für alle Werte\r\n");
printf ("LENGTH             Anzahl der Werte in der Datei\r\n");
printf ("NAME               Dateiname\r\n");
getch ();
printf ("\r\n");
exit (0);
                }
        argc--;
```

```
            }

        fp = fopen (fname,"wb");
        if (fp == NULL)
            {
            printf ("\r\n\nKann Datei nicht öffnen !\r\n\n");
            exit (0);
            }
        printf ("\r\n");

        if  (   (mode == 3)                         /* SI|E1|E2 */
             || (mode == 8) || (mode == 9) )
            {
            siwert = sampanz / 2;
            siwert *= -1;
            }
        if ((mode == 4) || (mode == 5))             /* S01|S10 */
            {
            sampanz = tast1+tast2;
            }
        if  (mode == 6)                             /* RAMP */
            {
            if (tast1 == 0) tast1 = 1;
            if (tast2 == 0) tast2 = 1;
            step1 = (double) (2.0 / (double) tast1);
            step2 = (double) (2.0 / (double) tast2);
            wert  = -1.0;
            }

        for (i = 0; i < sampanz; i++)
            {
            printf ("\rFertig: %d \%",(long) (i * 100L) / sampanz);

/*********************************************************************/

        switch (mode)
            {
            case 1:        /* SIN */
            wert = sin ( (double) (2*PI / sampfreq * sigfreq) * (double) i);
            break;

            case 2:        /* COS */
            wert = cos ( (double) (2*PI / sampfreq * sigfreq) * (double) i);
            break;

            case 3:        /* SI */
            wert = (double) (2*PI / sampfreq * sigfreq) * (double) siwert;
            if (wert == 0.0)
                    {
                    wert = 1.0;
                    }
            else
                    {
```

```
        wert = sin (wert) / wert;
        }
siwert++;
break;

case 4:         /* S01 */
if (i < tast1)
        {
        wert = 0.0;
        }
else
        {
        wert = 1.0;
        }
break;

case 5:         /* S10 */
if (i < tast1)
        {
        wert = 1.0;
        }
else
        {
        wert = 0.0;
        }
break;

case 6:         /* RAMP */
if (x == tast1)
        {
        y++;
        wert -= step2;
        if (y == tast2)
                {
                x    = 0;
                y    = 0;
                }
        }
else
        {
        x++;
        wert += step1;
        }
break;

case 7:         /* RECT */
if (x == tast1)
        {
        y++;
        wert = -1.0;
        if (y == tast2)
                {
                x = 0;
```

```
                                y = 0;
                                }
                        }
                else
                        {
                        x++;
                        wert = 1.0;
                        }
                break;

                case 8:         /* E1 */
                wert = alpha * fabs ((double) siwert);
                wert = exp (wert);
                siwert++;
                break;

                case 9:         /* E2 */
                wert = alpha * (double) siwert * (double) siwert;
                wert = exp (wert);
                siwert++;
                break;

                default:
                wert = 0.0;
                break;
                }

/*********************************************************************/

                fprintf (fp, "%16.7e", wert + offset);
                fputs ("\x0D\x0A", fp);
                }
          fclose (fp);

          printf("\rFertig !      \n\n\r");

exit (0);
}
```

Anhang E

Inhalt der Diskette

Anhang E: Inhalt der Diskette

Files von SPV:

-	SPVINC.H	Includedatei von SPV
-	SPV.C	Hauptmodul von SPV
-	SPVFUNC.C	Funktionen wie FFT, AKF, KKF usw.
-	SPVEDIT.C	Eingaberoutine und diverse Stringroutinen
-	SPVSIRP.C	Erzeugen der verschiedenen Eingangssignale
-	FILEREQ.C	Filerequester von SPV
-	SPV.OBJ	Objektdatei von SPV.C
-	SPVFUNC.OBJ	Objektdatei von SPVFUNC.C
-	SPVEDIT.OBJ	Objektdatei von SPVEDIT.C
-	SPVSIRP.OBJ	Objektdatei von SPVSIRP.C
-	FILEREQ.OBJ	Objektdatei von FILEREQ.C
-	EGAVGA.OBJ	Grafiktreiber
-	GRAPHICS.OBJ	Grafikfunktionen
-	SPV.DSK	Desktopdatei von Turbo C++ für SPV
-	SPV.PRJ	Projektdatei von Turbo C++ für SPV
-	SPV.EXE	Programmdatei von SPV

Includedateien zur Sirpgenerierung [5]:

- CONST.H
- FFT.H
- HEAP.H
- SIRP.H

Files von SPVGEN:

-	SPVGEN.C	Quellcode von SPVGEN
-	SPVGEN.OBJ	Objektdatei von SPVGEN.C
-	SPVGEN.DSK	Desktopdatei von Turbo C++ für SPVGEN
-	SPVGEN.PRJ	Projektdatei von Turbo C++ für SPVGEN
-	SPVGEN.EXE	Programmdatei von SPVGEN

Anhang F

Tabelle der Korrekturfaktoren

Anhang F: Tabelle der Korrekturfaktoren

$\tau_{\alpha r}$ [ms]	$\tau_{\alpha f}$ [ms]	Y_{cg}	Y_{cl}	Y_{cn}
1.00000	1.00000	1.00000	1.00000	1.00000
1.00000	2.00000	0.85758	0.78825	0.82491
1.00000	5.00000	0.72770	0.58861	0.65708
1.00000	10.00000	0.66138	0.48140	0.56506
1.00000	20.00000	0.61424	0.40062	0.49431
1.00000	50.00000	0.57230	0.32227	0.42393
1.00000	100.00000	0.55114	0.27810	0.38315
2.00000	1.00000	1.19915	1.29050	1.23753
2.00000	2.00000	1.00000	1.00000	1.00000
2.00000	5.00000	0.81839	0.72894	0.77541
2.00000	10.00000	0.72565	0.58537	0.65433
2.00000	20.00000	0.65974	0.47868	0.56270
2.00000	50.00000	0.60110	0.37696	0.47328
2.00000	100.00000	0.57150	0.32068	0.42248
5.00000	1.00000	1.59793	1.86401	1.70197
5.00000	2.00000	1.28519	1.41488	1.33865
5.00000	5.00000	1.00000	1.00000	1.00000
5.00000	10.00000	0.85436	0.78341	0.82087
5.00000	20.00000	0.75086	0.62496	0.68792
5.00000	50.00000	0.65877	0.47705	0.56129
5.00000	100.00000	0.61229	0.39715	0.49124
10.00000	1.00000	2.04917	2.50774	2.21997
10.00000	2.00000	1.60789	1.87825	1.71346
10.00000	5.00000	1.20549	1.29968	1.24501
10.00000	10.00000	1.00000	1.00000	1.00000
10.00000	20.00000	0.85396	0.78280	0.82037
10.00000	50.00000	0.72402	0.58278	0.65212
10.00000	100.00000	0.65844	0.47651	0.56082
20.00000	1.00000	2.68835	3.41602	2.94828
20.00000	2.00000	2.06501	2.53028	2.23807
20.00000	5.00000	1.49658	1.71884	1.58478
20.00000	10.00000	1.20630	1.30085	1.24596
20.00000	20.00000	1.00000	1.00000	1.00000
20.00000	50.00000	0.81645	0.72597	0.77292
20.00000	100.00000	0.72381	0.58246	0.65184
50.00000	1.00000	3.95771	5.21533	4.38759
50.00000	2.00000	2.97279	3.81955	3.27135
50.00000	5.00000	2.07465	2.54400	2.24909
50.00000	10.00000	1.61599	1.88984	1.72280
50.00000	20.00000	1.29003	1.42186	1.34431
50.00000	50.00000	1.00000	1.00000	1.00000
50.00000	100.00000	0.85363	0.78231	0.81996

Anhang F: Tabelle der Korrekturfaktoren

$\tau_{\alpha r}$ [ms]	$\tau_{\alpha f}$ [ms]	Y_{cg}	Y_{cl}	Y_{cn}
100.00000	1.00000	5.38884	7.24146	6.00625
100.00000	2.00000	3.99626	5.26992	4.43122
100.00000	5.00000	2.72638	3.46999	2.99150
100.00000	10.00000	2.07789	2.54862	2.25279
100.00000	20.00000	1.61701	1.89130	1.72398
100.00000	50.00000	1.20695	1.30178	1.24672
100.00000	100.00000	1.00000	1.00000	1.00000