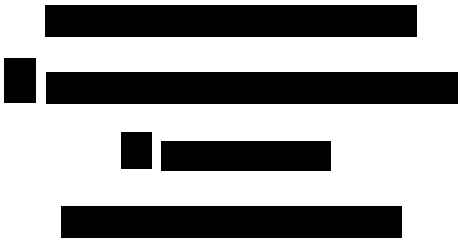
Technische Universität Berlin
Fakultät I
Institut für Sprache und Kommunikation

Fachgebiet Audiokommunikation

Erweiterung einer bestehenden Hörversuchsumgebung durch Implementierung des ZEST-Verfahrens

Magisterarbeit im Fach Kommunikationswissenschaft

Matthias Herder



6. Juni 2010

Betreuer: Prof. Dr. Stefan Weinzierl,
Dr. Hans-Joachim Maempel

Die selbstständige Anfertigung dieser Arbeit versichere ich an Eides statt.

Berlin, 7. Juni 2010 .....

MATTHIAS HERDER

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Wahrnehmungsexperimente 1

1.1.1 Historisch wissenschaftlicher Ursprung 1

1.1.2 Hörversuche heute: Anwendung und Nutzen 2

1.2 Hörversuchsumgebung Whisper – Funktionen und Möglichkeiten . 4

1.3 Aufgabenstellung dieser Arbeit 7

1.3.1 Implementierung des adaptiven Schwellwertverfahrens ZEST 7

1.3.2 Optimierung der Arbeitsabläufe – Stimulusorganisation . . 11

1.3.3 Optimierung der Arbeitsabläufe – Testkonfiguration 14

1.3.4 Portierung der Software auf unixartige Betriebssysteme . . 15

1.3.5 Technische Dokumentation 17

2 Das ZEST-Verfahren **19**

2.1 Einführung 19

2.2 Bayes-Statistik und Bayes-Theorem 20

2.3 *A-priori*-Wahrscheinlichkeitsdichtefunktion 24

2.4 Psychometrische Funktion und Likelihood 26

2.4.1 *Epsilon* und *ideal sweat factor* 29

2.5 Trialplatzierung, abschließende Schwellwertschätzung 30

3 Implementierungen **32**

3.1 Programmiertechniken in Matlab 32

3.1.1 GUIDE 32

3.1.2 Callbacks 36

3.1.3 Callbacks in .m-files, Handles 37
3.1.4 Spezialfälle 39
3.2 Implementierung von ZEST 41
3.3 Optimierung der Stimulus-Konfiguration 48
3.4 Standardwerte-Preset-System 51
3.5 Linux-Portierung 56
4 Abschlussdiskussion und Selbstkritik **61**
Anhang **65**
5.1 Literaturverzeichnis 65
5.2 Abbildungsverzeichnis 67
5.3 Matlab Quellcode 69

1 Einleitung

1.1 Wahrnehmungsexperimente

1.1.1 Historisch wissenschaftlicher Ursprung

„Psychophysik“ nannte ihr Vater Gustav Theodor Fechner seine Fortentwicklung der Vorarbeiten Ernst Heinrich Webers in einem Teilgebiet der empirischen Wissenschaften während der frühen Mitte des 19. Jahrhundert, innerhalb der er, wie es der Name suggeriert, *Psychisches* mit *Physischem* verknüpfte:

Im 1860 zweiteilig veröffentlichten „Elemente der Psychophysik“[1] stellt er eine Formel vor, anhand welcher er einen durch Zahlen ausdrückbaren Zusammenhang zwischen einem psychischen Vorgang oder Zustand und einem physischen Vorgang oder Zustand konstatiert, wobei *psychisch* und *physisch* bei Fechner für einen je nur nach Betrachter unterschiedlichen Blickwinkel auf ein und denselben Vorgang oder Zustand stehen - die von ihm *psychophysikalisches Grundgesetz* oder *Fundamentalformel* genannte Gleichung erlaubt nun nach ihm das Umrechnen der Werte einer dieser Erscheinungsformen in die jeweils andere.

Den empirischen Nachweis der Richtigkeit seiner Entdeckung muss Fechner auf indirektem Weg führen: Wäre man unter Umständen noch gewillt, als Korrelat eines *psychischen Zustandes* bestimmte neuronale Aktivitäten der betreffenden Versuchsperson anzuerkennen, so standen weder der Physiologie noch der Neurologie jener Zeit die Mittel und Wege zu deren direkter Untersuchung oder gar Messung

[1] Fechner, 1860

zur Verfügung. Da Fechner jedoch genug Grund zu der Annahme sah, dass auch zwischen den durch äußere Reize verursachten Empfindungen und den aus deren Weiterleitung resultierenden Hirnwerten Proportionalität bestände, genügte es ihm nun zum Nachweis der Richtigkeit seiner Formel, eben diese äußeren Reize sowie von ihnen hervorgerufene Sinnesempfindungen in ihrer Größe zu bestimmen und zu vergleichen.[2]

Ungeachtet nun der auch heute noch bestehenden Unklarheit, inwiefern sich selbst ein gegenwärtig zunehmend präziser bestimmbarer Gehirnzustand zu einem *Bewusstseinsinhalt* der entsprechenden Versuchsperson - und damit nämlich letztendlich wirklich zum nach Fechner postulierten von der Außenperspektive unterscheidenden *psychischen Zustand* verhält, zeigen sich im eben skizzierten indirekten empirischen Beweisverfahren Fechners die Ursprünge von jenen Experimenten, die darauf abzielen, die Stärke von einer Versuchsperson dargebotenen Reizen in Beziehung zu setzen mit der Stärke der von dieser Versuchsperson daraufhin wahrgenommenen Sinnesempfindung - den sogenannten *Wahrnehmungsexperimenten.*

1.1.2 Hörversuche heute: Anwendung und Nutzen

Wahrnehmungsexperimente lassen sich nach den mithilfe von ihnen untersuchten Sinnesmodalitäten unterscheiden: Es gibt Versuche zur visuellen Wahrnehmung, zur auditiven, zum Tast- sowie zum Geruchssinn. Die *Psychoakustik* als Teilgebiet der *Psychophysik* beschäftigt sich mit dem Hörsinn, die von ihr verwendeten Wahrnehmungsexperimente sind die Hörversuche.

Hörversuche werden in vielen Wissenschafts- wie auch praktischen Bereichen mit unterschiedlicher Zielsetzung durchgeführt – man findet sie in der (experimentellen) Psychologie und Physiologie, (wie bereits erwähnt) ebenfalls in der

[2] Gundlach, 1993, S. 2 ff.

Psychoakustik sowie im Weiteren angrenzend auch bei musikwissenschaftlichen und soziologischen Fragestellungen, und schließlich darüberhinaus auch in allen über das reine wissenschaftliche Erkenntnisinteresse hinausgehenden Bereichen, in denen es um die Beurteilung der Qualität und der Eigenschaften einer großen Bandbreite von „Entwicklungen“ geht – diese reichen von Tonerzeugungs- über -wiedergabe- und - aufnahmeapparate bis hin zu reinen Algorithmen z. B. der digitalen Audiokompression.

Mit Hörversuchen ist man nun in jedem dieser Bereiche in der Lage, grundsätzlich eine der folgenden vier Untersuchungen durchzuführen:

- Wie (in der interessierenden Hinsicht) stark muss ein bestimmter akustischer Reiz sein, damit er (überhaupt) wahrgenommen wird? Untersucht wird so die *Wahrnehmungsschwelle*.

- Wie groß muss der Unterschied zwischen zwei dargebotenen Reizen (in der interessierenden Hinsicht) sein, damit sie auch als unterschiedlich wahrgenommen werden? Die Untersuchung gilt hier der *Reizunterscheidung* bzw. der *Reizunterschiedsschwelle*.

- Unter welchen Bedingungen können identifizierende Aussagen über einen dargebotenen Reiz gemacht und dieser nicht bloß als vorhanden angegeben werden? Dies ist eine Frage der *Reizerkennung*.

- Schließlich: Unter welchen Bedingungen kann zutreffend etwas über unterschiedliche Stärken eines dargebotenen Reizes ausgesagt werden? Hierbei beschäftigt man sich mit den Fähigkeiten zur *Reizskalierung*.

Wird hierbei nicht der Mensch, sondern das dargebotene Testmaterial untersucht, lassen sich im Umkehrschluss mit Hörversuchen Antworten auf (unter anderem) folgende Fragen finden:[3]

[3] Bech und Zacharov, 2006, S. 4.

- Sind zwei Stimuli perzeptiv gar nicht zu unterscheiden?
- Ist ein bestimmter Stimulus in Bezug auf Klangqualität qualitativ besser als ein bestimmter anderer?
- Um wieviel ist ein bestimmter Stimulus in Bezug auf Klangqualität besser als ein bestimmter anderer?
- Welches „Audiosystem“[4] wird bevorzugt?
- Genügt ein „Audiosystem“ den Anforderungen einer bestimmten, vorher festgelegten Aufgabe?

Die Möglichkeit, Fragen in beiderlei Hinsicht mithilfe von Hörversuchen beantworten zu können, macht die Wichtigkeit von Hörversuchen als Werkzeug aller oben aufgeführten Disziplinen und Betätigungsfelder überdeutlich.

1.2 Hörversuchsumgebung Whisper – Funktionen und Möglichkeiten

Am Fachbereich Audiokommunikation der Technischen Universität Berlin wurde darum im Rahmen der Magisterabschlussarbeiten von Simon Ciba und André Wlodarski ein Computerprogramm entworfen und implementiert, welches eine Umgebung zur Durchführung von Hörversuchen zur Verfügung stellt.

„Whisper“[5] ermöglicht die Ausführung aller während eines Hörversuchs anfallender Schritte innerhalb einer einheitlichen Oberfläche:

[4] Bech und Zacharov füllen den Begriff nicht weiter aus – denkbar sind hierunter außer (den offensichtlichen) z. B. Stereo- oder Heimkinoanlagen aber auch verschiedene Telefonieanlagen, Hörgeräte, unterschiedliche Abmischungen von Filmtonspuren,ja, in gewisser Hinsicht selbst Audiocodecs, etc.

[5] Ciba, 2008, Wlodarski, 2008a.

1. Anlegen, Organisieren und Konfigurieren von Stimuli.

   Whisper ermöglicht hier nicht nur die auch sonst oft übliche Nutzung von lokal, d.h. auf dem Whisper ausführenden Rechner vorliegenden Audiodateien im „wav"-Format, unterstützt werden darüberhinaus als Stimulus die Steuerung von via Netzwerk erreichbaren Gegenstellen per *open sound control* (*OSC*) Protokoll. Hierbei ist auch die simultane Nutzung beider Arten von Stimuli möglich.[6]

   Erleichtert wird von Whisper auch die Organisation der zu verwendenden Stimuli – Benennung, Eingabe der OSC-Parameter und -Daten sowie gesammeltes Speichern in einem extra hierfür reservierten Ordner gehören zu den dieses gewährleistenden Funktionen.

2. Auswahl, Organisation und Konfiguration von Versuchsverfahren.

   Grundsätzlich haben sich im Laufe der wissenschaftlichen Arbeit an den weiter oben skizzierten durch Wahrnehmungsversuche beantwortbaren Fragestellungen spezifische hierauf besonders abgestimmte Testverfahren herausgebildet. Zu Beginn der vorliegenden Arbeit bot Whisper *semantisches Differential*, die *repertory grid technique*, *ABX-Trials* sowie *adaptive Verfahren* zur Auswahl, wobei letztere mit *nAFC*-Paradigma ($n \leq 4$) und *Staircase*, *Best PEST* und einer Reihe von *maximum likelihood*-basierten bzw. auf der *Bayes-Statistik* beruhenden Verfahren verfügbar sind. Die adaptiven Verfahren erlauben hierbei eine Untersuchung der *Reizschwelle*, ABX-Trials ermöglichen Auskünfte zur *Reizunterscheidung*, semantisches Differential

[6] Mit seiner OSC-Fähigkeit durchbricht Whisper sogar eigentlich den primär intendierten Nutzungsrahmen als (bloße) Hörversuchsumgebung – denn da per OSC nahezu beliebige „Aktionen" kontrolliert und initiiert werden können, stünde (zumindest in dieser Hinsicht) einer Nutzung von Whisper als Umgebung auch für wenigstens visuelle Wahrnehmungsexperimente tatsächlich nichts entgegen.

und Repertory Grid Technique sind als psychologische Verfahren geeignet, Fragen wie obige nach der Bevorzugung eines bestimmten „Audiosystems“ zu beantworten (sie sind hierauf jedoch nicht beschränkt).[7]

Whisper ermöglicht die Konfiguration aller relevanten Parameter der ausgewählten Verfahren über je eine verfahrensspezifische graphische Oberfläche (und erleichtert dies Aufgabe an vielen Stellen durch sinnvolle Vorgabe von Standardwerten), die gewählten Verfahren können zudem im zeitlichen Ablauf eines komplexeren Versuchsdurchlaufs kombiniert und organisiert werden. Die speichernde Sicherung vorgenommener Konfiguration ist selbstverständlich auch gewährleistet.

3. Durchführung und Ablaufsteuerung des eigentlichen Hörversuchs.
   Whisper vereint hierbei sowohl die Bedienelemente und -oberflächen, die einem Versuchsleiter vor und während eines Hörversuchs zur Verfügung stehen müssen, als auch die den jeweiligen Versuchsteilnehmern dargebotenen Oberflächen und Bedienelemente in sich. Dem Versuchsleiter werden hier organisatorische Mittel (zur Identifikation der Versuchsperson z. B.) zur Hand gegeben, desweiteren hat er die Möglichkeit, bestimmte vorher festgelegte und zusammengefasste Parameter der Versuchsverfahren auch jetzt noch zu variieren, dem Versuchsteilnehmer werden hingegen Versuchserkärungen und -instruktionen sowie natürlich graphische Bedienelemente zur Abgabe seiner Reaktionen auf die Stimuli dargeboten.

4. Sammlung der Ergebnisse und Unterstützung bei deren Weiterverarbeitung.
   Whisper fertigt eine extensive Dokumentation des gesamten Hörversuchs an. Dazu gehören die Protokollierung der verwendeten Stimuli, aller benutzten Verfahren inklusive sämtlicher zu diesen gehörenden Einstellungen,

[7] Zu dieser Einteilung siehe z. B. Maempel, 2006, S. 7 ff.

Protokollierung des Versuchsablaufs und selbstverständlich die Sammlung aller empirischer Daten (d. h. der Eingaben der Versuchsperson) sowie deren graphische Darstellung und für die spätere (z. B. statistische) Weiterbearbeitung und/oder Auswertung optimierte Daten- und Dateiformate.

1.3 Aufgabenstellung dieser Arbeit

In Kapitel 1.1 habe ich versucht, die Herkunft von Hörversuchen nachzuzeichnen sowie ihre heutige Bedeutung in den empirischen Wissenschaften anhand ihrer Einsatzmöglichkeiten zu skizzieren, Kapitel 1.2 stellte die am Fachbereich Audiokommunikation durchgeführte Implementierung der Hörversuchsumgebung *Whisper* als eine Antwort auf den daraus erwachsenden Bedarf vor.

Obwohl nun die von Whisper zur Verfügung gestellten Möglichkeiten sehr umfangreich sind und weite Bereiche aller im Zusammenhang mit Hörversuchen anfallenden Aufgaben abdecken, stellte sich in Anschluss an die ursprüngliche Implementierung heraus, dass aus unterschiedlichen Gründen an verschiedenen Stellen der Software der Wunsch nach einer Erweiterung oder Ergänzung des Programms bestand.

1.3.1 Implementierung des adaptiven Schwellwertverfahrens ZEST

Wie in Kapitel 1.2 dargelegt, implementiert Whisper verschiedene Testverfahren, die sich primär nach der Art der von ihnen untersuchten Fragestellung unterscheiden – zur Ermittlung von *Schwellwerten* bietet Whisper (wie gesagt) bereits eine Reihe *adaptiver Verfahren*.

Zu Beginn des wissenschaftlichen Gebrauchs von Hörversuchen wurden Schwellwerte üblicherweise mithilfe eines der als „klassisch“ bezeichneten drei Verfahren

Herstellungsverfahren, *Grenzverfahren*, *Konstanzverfahren* bestimmt.[8]

In ersterem regelt die Versuchsperson die zu untersuchende Intensität eines in eben dieser Hinsicht interessierenden Reizes solange selbst, bis sie glaubt, eine vor Versuchsbeginn vorgegebene Wahrnehmung in Bezug auf den oder die Reize zu haben.

Im Grenzverfahren geschieht dies durch den Versuchsleiter, wobei dieser sich der Schwelle sowohl von oben als auch von unten nähert und die zwei ermittelten Werte durch arithmetische Durchschnittsbildung in die gesuchte Schwelle überführt.

Das Konstanzverfahren verwendet einen Satz vorher festgelegter Stimuli, diese sollen gleichabständig und möglichst um die zu ermittelnde Schwelle herum verteilt sein. Sei werden der Versuchsperson in zufälliger Reihenfolge dargeboten, diese wird nach dem Vorhandensein einer bestimmten Sinneswahrnehmung gefragt, die pro Stimulus notierten Antworthäufigkeiten werden als eine sogenannte *psychometrische Funktion* aufgetragen, wobei deren 50 %-Punkt die gesuchte Schwelle markiert.

Alle drei klassischen Verfahren weisen Probleme auf: Im Rahmen der ersten zwei genannten ist die Versuchsperson gezwungen, selbst zu entscheiden, wann sie eine Schwelle für erreicht hält und wann nicht. Verschiedene Versuchspersonen werden hier zwangsläufig ein unterschiedlich stark ausgeprägtes *Kriterium* für diese Entscheidung setzen, das Untersuchungsergebnis wird auf diese Weise letztendlich verfälscht. Außerdem zeigen sich in beiden Verfahren systematisch das Ergebnis verfälschende Störungen dadurch, dass der Versuchsperson die Richtung der Reizintensitätsänderung bekannt ist – sie antwortet entweder immer zu früh (*Antizipation*) oder immer zu spät (*Habituation*).

Das Konstanzverfahren teilt diese Probleme nicht, doch es ist zum einen auf-

[8] Maempel, 2006, S. 4 f.

grund der relativ großen zur zuverlässigen Bestimmung der psychometrischen Funktion benötigten Anzahl an Durchläufen sehr zeitaufwendig. Dies ist ein Problem, das weniger auf vermeintliche akademische Arbeitsscheue verweist, als vielmehr auf die sich ganz praktisch tatsächlich immer schwierig gestaltende Suche nach zur Teilnahme an Hörversuchen bereiten Freiwilligen – oft unentgeltlich, möglichst noch in großer Zahl.[9] In Aussicht gestellte lange Testdurchläufe sind nicht geeignet, dieser Schwierigkeit sinnvoll zu begegnen.

Zum anderen besteht ein Dilemma des Konstanzverfahrens darin, dass die verwendeten Stimuli möglichst um die zu messende Schwelle herum verteilt sein sollten – diese jedoch ist natürlich vor Beginn des Wahrnehmungsversuchs noch gar nicht bekannt.

Die auch in Whisper bereits in Auswahl vorliegenden das nAFC-Paradigma verwendenden adaptiven Testverfahren nun vermögen es, effektiv die oben genannten Beschränkungen zu beseitigen: Als *forced-choice*-Verfahren nötigen sie die Versuchsperson bei jeder Stimuluspräsentation zu einer Antwort und entbinden sie so von der Notwendigkeit, ein Kriterium für eine Entscheidung zu setzen und zu nutzen.

Adaptiv heißen die entsprechenden Verfahren, da sie die Auswahl des jeweils folgenden Stimulus in Abhängigkeit von der vorhergehenden Antwort der Versuchsperson treffen – ob es einem Verfahren gelingt, Antizipations- und Habituationsfehler zu vermeiden, hängt darum davon ab, ob die Versuchsperson den Adaptionsmechanismus zu durchschauen und die Richtung der Platzierung des folgenden Stimulus vorherzusagen im Stande ist. Alle in Whisper vorhandenen adaptiven Messverfahren sind von sich aus hinreichend komplex in der Stimulusselektion (Best PEST, maximum likelihood/Bayes-Verfahren) oder erlauben

[9] Das Problem zu zeitintensiver Hörtests besteht in sogar verschärfter Form bei Hörversuchen mit Kleinkindern.

dieses gewährleistende Einstellungen (Staircase) dahingehend, dass ein Durchschauen der Adaptionsregel durch die Versuchsperson praktisch unmöglich ist und Antizipations- und Habituationsfehler in Messungen ausgeschlossen werden können.

Für ihren Einsatz als Schwellwertmessverfahren relevante Unterschiede typischer Vertreter der „modernen“ adaptiven Verfahren arbeitete eine ebenfalls am Fachbereich Audiokommunikation der Technischen Universität Berlin verfasste Arbeit jüngeren Datums deutlich heraus:[10] Zwar ist der *Aufwand* zum Einsatz des Staircase-Verfahrens relativ gering – es ist nicht schwierig zu verstehen oder zu implementieren und benötigt bezüglich der Parameter keine besonderen Vorkenntnisse, im Gegensatz zu Best-PEST und den maximum likelihood/Bayes-basierten Verfahren; die von diesen genutzte psychometrische Funktion sollte schon vor Versuchsbeginn möglichst genau bestimmt sein.

In Bezug auf mit der Gültigkeit des ermittelten Ergebnisses in Zusammenhang stehende Aspekte jedoch sind diese Verfahren dem Staircase-Verfahren durchwegs überlegen: Sie weisen praktisch[11] keinen systematischen, durch das Verfahren bedingten Fehler (*Bias*) auf, während dieser bei Staircase direkt von der Wahl der Parameter abhängt, die Genauigkeit der durch sie ermittelten Ergebnisse ist zudem sehr hoch (geringe *Varianz*), und es ist möglich, durch sie eine noch frühere Konvergenz des Testverlaufs zu erreichen, als dies das Staircase-Verfahren zu leisten vermag. Mit dem so vorteilhafteren Verhältnis der Anzahl benötigter Trials zur Varianz der Messung geht als *Effizienz* der dritte der den *Nutzen* beschreibenden Merkmale in die unterscheidende Evaluation der besagten Verfahren ein.[12]

Als schließlich optimal stellte sich das ZEST-Verfahren heraus: Für alle Paradigmen ohne systematischen Messfehler, zeigten die Messwerte für alle Paradig-

[10] Otto, 2008. Evaluiert wurden: Staircase-Verfahren, PEST, Best-PEST, ZEST.
[11] Das Best-PEST-Verfahren tatsächlich nur bei geeigneter Wahl des Paradigmas.
[12] Otto, 2008, S. 6, S. 61 ff.

men durchweg sehr geringe Varianz, und das Verfahren konvergierte als schnellstes von allen Untersuchten schon nach nur zwanzig Trials – ein Umstand, der wie kein anderer geeignet ist, den weiter oben beschriebenen praktischen Anforderungen an Hörversuchsdurchführungen gerecht zu werden.

All dies führte zu dem Wunsch nach einer Implementierung des bisher nicht vollständig in Whisper zur Verfügung stehenden ZEST-Verfahrens.

1.3.2 Optimierung der Arbeitsabläufe – Stimulusorganisation

Das Programm Whisper verwaltet Stimuli für Hörversuche global, d. h., dass Reize nicht erst auf der Ebene einzelner Hörversuche, nicht erst in Zusammenhang mit dem Festlegen und Konfigurieren von bestimmten Messverfahren, sondern unabhängig von diesen und ihnen tatsächlich vorgeordnet angelegt, bearbeitet und verwaltet werden. Der offensichtliche Vorteil dieses Vorgehens ist, dass ein einmal dem Programm „bekannt gemachter“ Stimulus in *jedem* Testverfahren zur Verfügung steht; und in jeder Instanz eines Testverfahrens: So ist z. B. auch die Aneinanderreihung nur leichter Variationen von ein und demselben Testverfahren in Hinblick auf die gleichen zu verwendenden Stimuli sehr einfach zu realisieren.

Whisper bietet zum Zweck der Stimulusorganisation und -bearbeitung den sogenannten *Stimulus Pool*. Erreicht werden kann er über den gleichnamigen *Push-button* im Hauptfenster der Software, die sich dem Benutzer direkt nach dem Programmstart zeigt. Abbildung 1.1 zeigt die Oberfläche des *Stimulus pool*.

Im zentralen Fenster werden die bereits angelegten Stimuli aufgelistet (mindestens einer findet sich hier selbst nach Anlegen eines gerade ganz neuen Projekts), und per Button lassen sich entweder je ein Stimulus bearbeiten, jeweils ein neuer Stimulus anlegen, der letzte Stimulus der Liste löschen (ausgenommen, es befindet sich sowieso nur noch ein einziger Stimulus im Pool), oder ein (konfigurierter) Stimulus zur Probe abspielen (im Falle einer lokalen „wav“-Datei wird diese über

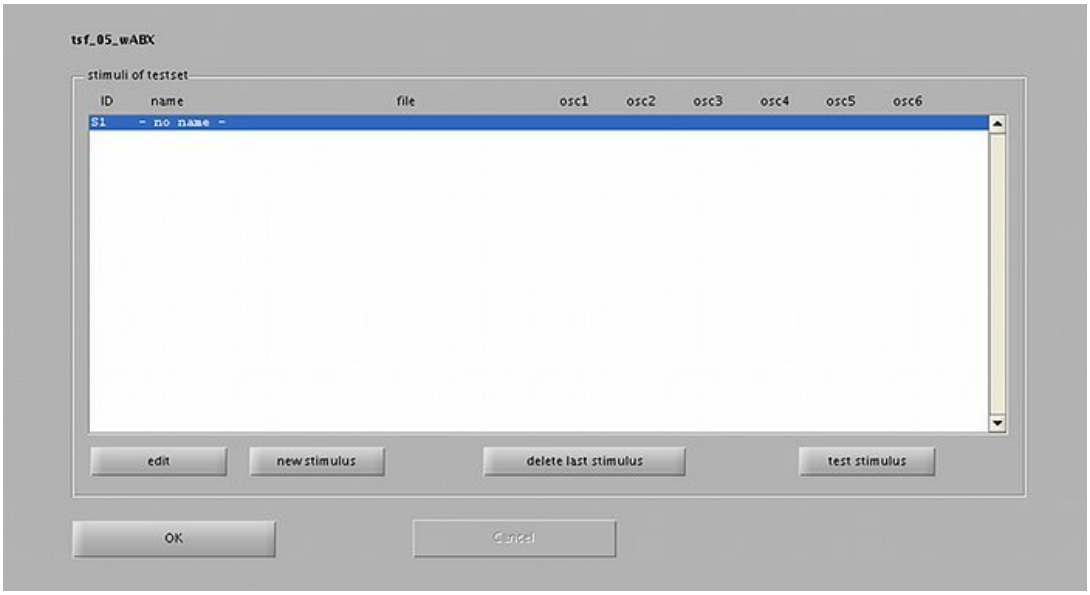


Abbildung 1.1: Screenshot des *Stimulus Pool* von Whisper (Version 1.05 [in Zukunft: „alte Version“]).

das Audiosystem des Rechners wiedergegeben, im Falle von OSC-Daten werden diese gemäß den globalen Netzwerkeinstellungen des Programms verschickt).

Der normale Weg, einen Stimulus anzulegen und für die Nutzung durch ein Testverfahren zur Verfügung zu stellen, besteht also darin, zuerst per Klick auf [new stimulus] einen neuen Eintrag in der Liste zu erzeugen und diesen nach einem Klick auf [edit] im sich daraufhin öffnenden Fenster „edit stimulus“ zu konfigurieren, siehe Abbildung 1.2. Um eine irgendwie brauchbaren Stimulus zu erhalten, muss nun mindestens entweder per [browse] -Button und anschließender Auswahl im sich öffnenden Dateiauswahldialog eine „wav“-Datei gesucht und ausgewählt werden, oder in mindestens einem der Server-Felder für OSC-Befehle die gewünschten Parameter eingetragen werden; sollen nicht alle Stimuli den gleichen Standardeintrag in ihrem Namensfeld („- no name -“) tragen und später z. B. bei der Zuordnung zu Intensitätsstufen im Rahmen der Testkonfiguration irgendwie

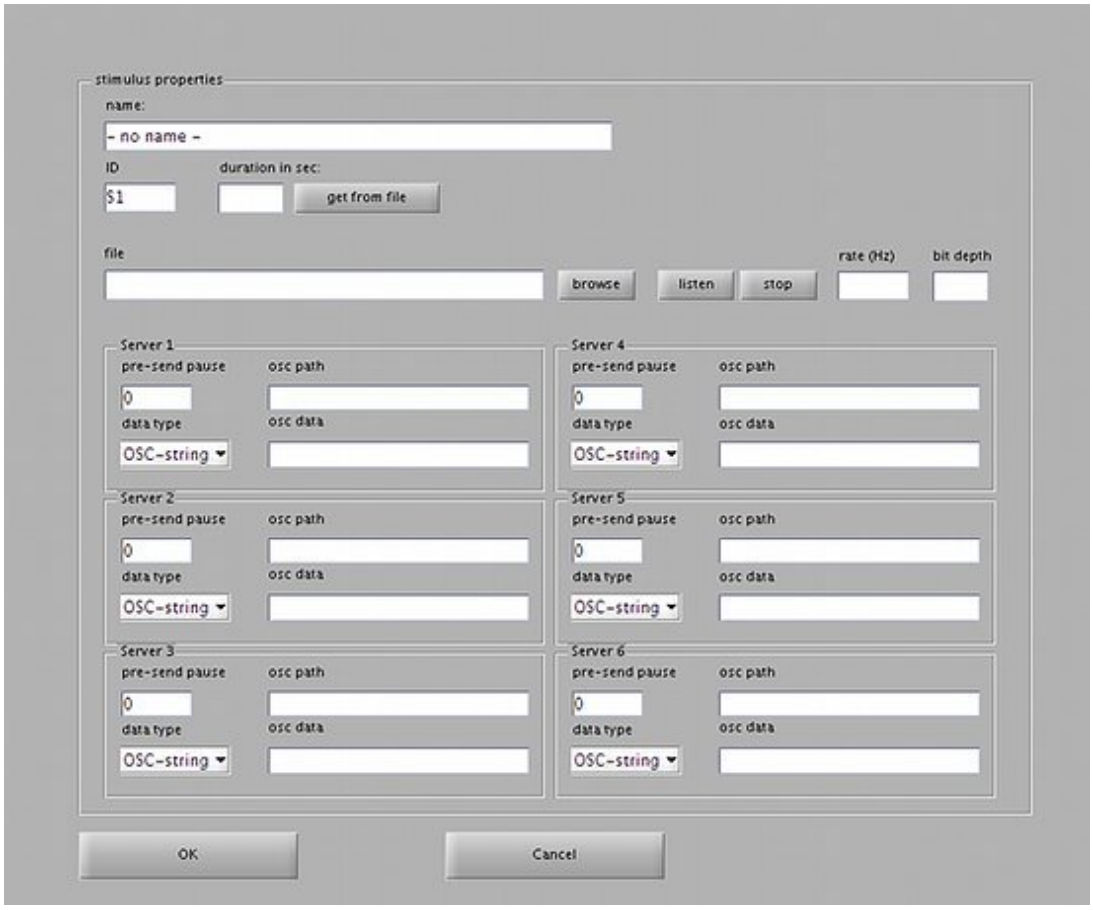


Abbildung 1.2: Screenshot des Dialogs „edit stimulus“ von Whisper.

identifizierbar sein, ist darüberhinaus das Vergeben eines aussagekräftigen Stimulusnamens in dafür vorgesehenen Feld vonnöten. Abschließend muss der „edit stimulus“-Dialog noch durch einen Klick auf den OK-Button geschlossen und gleichzeitig damit die getroffenen Einstellungen für den betreffenden Stimulus abgespeichert werden.

Dies heißt, dass pro anzulegenden Stimulus allerwenigstens sechs Mausklicks auszuführen sind, um auch nur eine Sounddatei mit diesem Stimulus zu assoziieren, eine individuelle Bezeichnung hat er dann allerdings auch noch nicht; die Zahl der nötigen Schritte erhöht sich darüberhinaus bei nicht im aktuellen Verzeichnis

befindlichen Dateien und ebenso ganz erheblich bei gewünschter Nutzung eines Stimulus als OSC-Datenquelle.

Zu allem Übel kommt zu diesem misslichen Umstand hinzu, dass bei adaptiven Hörtestverfahren üblicherweise relativ große Mengen von Stimuli benötigt werden und der Konfigurationsaufwand sich somit potenziert. Schon den ursprünglichen Autoren des Programms war das Problem nicht unbekannt, sie schlugen eine möglicherweise halbautomatische Hilfe zur Bewältigung der Konfigurationsaufgaben vor.[13]

Deren Konzeption und Implementierung stellt die zweite Teilaufgabe dieser Arbeit dar.

1.3.3 Optimierung der Arbeitsabläufe – Testkonfiguration

Innerhalb des Programms Whisper erfolgt die Konfiguration aller zur Verfügung stehenden Parameter – seien es jene der Stimuli, jene der Testverfahren, den Testverfahren übergeordnete, die Versuchsdurchführung betreffende, wie auch grundlegende „Systemeinstellungen“ (Netzwerk, Plotting) – über graphische Benutzeroberflächen und Dialoge. Wie man sich leicht vorstellen kann, ist die Zahl der Konfigurationsmöglichkeiten immens, ihre Verteilung auf verschiedene Teilbereiche und Oberflächen folgt einer an der Benutzung der Software orientierten Strukturierung.

Da es z. B. keinen Sinn hätte, ein Testverfahren überhaupt nur auszuwählen, bevor nicht bereits ein Hörversuchsprojekt („test series“) angelegt wurde, ist erstere Konfigurationsmöglichkeit tatsächlich erst *nach* dem Ausführen des zweitgenannten Arbeitsschrittes auch nur verfügbar. Diese Aufteilung dient ganz klar der Verständlichkeit der dargebotenen Optionen – da sie immer nur am für sie relevanten Orte angeboten werden, ist immer deutlich, worauf sie sich beziehen;

[13] Ciba, 2008, S. 91.

Dialoge wirken aufgeräumt und unüberfrachtet.

Die Kehrseite dieser Medaille ist jedoch, dass sich die Komplexität der Zusammenhänge und gegenseitigen Abhängigkeiten der einzelnen Teilbereiche von Whisper nun in der beträchtlichen Tiefe der Verschachtelung der Dialoge und deren Anzahl abbildet und niederschlägt: Die mit dem Programm ausgelieferte technische Dokumentation listet eine Verschachtelungstiefe von nicht weniger als fünf Ebenen und schon alleine neunundzwanzig Dialogoberflächen nur auf der ersten Seite des drei DIN A4-Seiten überspannenden Diagramms, das sich bemüht, Whispers gesamte Oberfläche darzustellen.[14]

So vorteilhaft die oben skizzierte Benutzerführung sich bei Unkenntnis des Programms und der zur Verfügung stehenden Konfigurationsmöglichkeiten zeigt, so schwierig macht sie eben deren Konfiguration, wenn zwar die Option, nicht aber ihr genauer „Ort“ im Programm bekannt ist. Und selbst, wenn dieser bekannt sein sollte – um z. B nur eine Veränderung an einer Einstellung im Staircase-Testverfahren vorzunehmen – oder diese Einstellung auch nur zu kontrollieren – ist es nötig, sich durch vier Dialoge hin und anschließend wieder zurückzuklicken.

Es erwies sich also für den praktischen Umgang mit Whisper als äußerst wünschenswert, über die bestehende Verfügbarmachung der Konfigurationsmöglichkeiten hinaus zusätzlich eine direkter zugängliche Alternative zu schaffen.

Dies zu entwerfen und umzusetzen ist damit die dritte Teilaufgabe dieser Arbeit.

1.3.4 Portierung der Software auf unixartige Betriebssysteme

Whisper wurde mithilfe der Programmiersprache und Programmentwicklungsumgebung Matlab unter dem Betriebssystem Microsoft Windows Vista 32 bit entwickelt. Matlab ist jedoch keineswegs nur exklusiv für dieses Betreibssystem

[14] Wlodarski, 2008b, S. 14 ff.

verfügbar – neben einer Version für Apples OS X sind Versionen für das traditionell und auch heute immer noch an universitären Instituten der Informatik weitverbreitete kommerzielle Unix der Firma Sun Microsystems namens „Solaris“, sowie für die quelloffenen und frei entwickelten und verfügbaren Linux-Varianten verfügbar.

Trotz prinzipieller Verfügbarkeit der Umgebung und Sprache, in der Whisper realisiert wurde, auf anderen Platformen als Microsoft Windows, läuft Whisper *nicht* ohne weiteres Zutun auf diesen, der Grund hierfür ist jedoch nicht (primär) in der Programmiersprache zu suchen, sondern an jenen Bereichen, an denen Whisper sozusagen über die reinen Grenzen der Programmiersprache hinausgeht und auf andere Ressourcen des lokalen Rechners zuzugreifen versucht – namentlich geht es um Dateizugriffe und jede andere Art von Dateisystemoperationen, ferner muss die Möglichkeit für die betreibssystemspezifisch unterschiedliche Art, auf die Audiohardware des verwendeten Rechners zuzugreifen sichergestellt sein, und drittens ist dafür zu sorgen, dass Whispers OSC-Datenkommunikationsfähigkeiten auch unter einer neu zu unterstützenden Plattform vorhanden ist bzw. ermöglicht werden kann.

Das Fachgebiet Audiokommunikation betreibt selbst einige Rechner mit Linux-Betreibssystem, da jedoch die oben genannten Punkte nicht Teil der Konzeption waren oder im Fokus der Aufmerksamkeit der ursprünglichen Whisper-Implementation standen und so schon ein prinzipieller Einsatz auf diesen oder anderen Rechner mit Linux-Betreibssystem vereitelt wurde, sich die Bearbeitung der dies verhindernden Punkte aber auch als nicht eben *en passent* zu erledigen herausgestellt hatten, ist dieses Unterfangen die vierte Teilaufgabe dieser Arbeit.

1.3.5 Technische Dokumentation

Die Komplexität von Whisper wurde bereits in früheren Abschnitten angedeutet, sei es anhand der Beschreibung des Funktionsumfangs, sei es anhand der Beschreibung des Konfigurationsumfangs und der Andeutung der Verschachtelungen der graphischen Oberflächen. Es dürfte so wenig erstaunen, zu erfahren, dass die Programmierung, der Programmcode, die Art und Weise, wie Funktionalität hergestellt und operationalisiert wird, sich als kein bisschen weniger komplex darstellt, als die „äußeren“, „nur“ die Nutzung des Programmes betreffenden Aspekte.

Diese komplexen Zusammenhänge laufen dabei zudem zwischen graphischen Benutzerinterface und dem darunterliegenden Programmcode keineswegs parallel: Bietet das graphische Benutzerinterface z. B. eine Möglichkeit, die Option „Zufällige Reihenfolge aller Tracks“ anzuwählen, so erfährt man hierdurch weder, wo,[15] wie,[16] oder wann[17] diese Auswahl gespeichert wird, noch ob sie vielleicht sogar überhaupt nicht gespeichert und statt dessen weiterverarbeitet wird.[18]

Daraus ergibt sich, dass der nächste wesentliche Schritt zur programmierenden Erweiterung von Whisper nach dem initialen Studium des Programmkonzepts

[15] D. h. z. B.: In welcher Variable?

[16] „Nur“ als Variable? Oder doch mit Abspeicherung in eine Datei? Und wenn ja – welche denn? Etc. ...

[17] D. h. z. B. durch welche Benutzeraktion – in Whisper werden beispielsweise – alles andere als offensichtlich – *fast nie* Eingaben dadurch oder zu dem Zeitpunkt gespeichert, zu dem ein anderer Wert vom Benutzer eingegeben, eine Häkchen entfernt oder gesetzt, ein anderer Menüpunkt ausgewählt wurde etc.!

[18] Z. B. so: In Whisper existieren häufiger Paare von Bedienelementen – z. B. die alternativen sog. *radio buttons* „all objects in random order“ und „manual selection“ im Konfigurationsdialog des semantischen Differentials – wählt man einen von ihnen an, wird der andere automatisch abgewählt. Gespeichert werden nun nicht Variablen wie etwa `all_obj_random=no` und `manual_select=yes`, sondern die Stellungen *beider* Bedienelemente werden verrechnet und im Konfigurationsparameter `use_multisession=yes` festgehalten!

und der Programmbedienung und -benutzung das Studium der Struktur der Programmierung ist. Genau diese ist nun bedauerlicherweise ganz im Gegensatz zum Programmkonzept und der Programmbedienung fast völlig undokumentiert.[19]

Da ich nun schon mindestens der zweite Autor bin, der eine Erweiterung von Whisper vornimmt und auch in Zukunft nicht davon auszugehen ist, dass die Weiterentwicklung oder auch nur die Wartung grundsätzlich eingestellt werden wird, jeder Autor aber wieder zwangsweise der unnötigen und auch völlig unökonomischen Aufgabe unterworfen ist, sich selbst eine (zumindest mentale) Dokumentation zu erarbeiten, diese jedoch ohne dass sie festgehalten wird, gerade nach Abschluss der jeweiligen Arbeit wieder einfach verloren ist und sich das Problem fehlender Dokumentation bei nun hinzugekommenen, ebenfalls undokumentierten Programmteilen sogar noch verschärft – da aber im Falle auch nur partieller oder ergänzender Dokumentation *alle* zukünftigen und gegenwärtigen Autoren von dieser profitieren, darum soll ein weiterer Teil dieser Arbeit der begleitenden Dokumentation als Hilfestellung zu einer Erweiterung von Whisper gewidmet sein.

[19] Die bereits erwähnte *Technische Dokumentation* gibt nur teils sehr lapidar auf gerade mal zweieinhalb Seiten Auskunft über die zukünftige Implementation von Testverfahren – wichtige Datenstrukturen wie etwa die alle Vorgabewerte für alle Testverfahren enthaltende *PC* bleiben leider komplett undokumentiert, vieles andere enttäuschend vage. Siehe Wlodarski, 2008b, S. 21-23.

2 Das ZEST-Verfahren

2.1 Einführung

Bei *ZEST*[1] handelt es sich um ein adaptives psychometrisches Schwellwertmessverfahren. *ZEST* stellt eine nur geringe (wiewohl entscheidende) Modifikation des *QUEST*-Verfahrens[2] in dessen Kriterium für die Platzierung des jeweils nächsten Reizes für folgende Trials dar (hierzu später mehr), in allen anderen Bereichen die Theorie und Praxis des Verfahrens betreffend stimmen *ZEST* wie *QUEST* nahezu völlig überein. Wird hier nun im Folgenden aus historischen Gründen teils zunächst das QUEST-Verfahren beschrieben, so gelten alle hierüber getroffenen Aussagen gleichermaßen auch für ZEST, auf dessen Abweichungen wird selbstverständlich explizit hingewiesen und diese kenntlich gemacht. Das gleiche gilt auch in umgekehrter Weise.

Adaptive Schwellwertmessverfahren richten die Platzierung von Reizen nach den vorangehenden Reaktionen der Versuchsperson. Dies hat jedoch nicht nur den in Kapitel 1.3.1 aufgezeigten Vorteil, dass sie (potentiell) in der Lage sind, Habituations- und Antizipationsfehler von Messungen fernzuhalten.

Sie weisen üblicherweise auch eine höhere *Effektivität* (im ebenfalls in Kapitel 1.3.1 genannten Sinn) auf: Antworten der Versuchsperson zu präsentierten Reizen, die nahe der zu untersuchenden Schwelle liegen, haben im Allgemeinen

[1] King-Smith u. a., 1994. Das Akronym steht für „Zippy estimation by sequential testing“ (ebd., S. 895).

[2] „Quick estimation by sequential testing“: Watson und Pelli, 1983.

einen höheren Informationsgehalt als solche zu weit entfernt liegenden; In einem „Ja/Nein“-Experiment z. B. besagt eine gemessene „Ja“-Wahrscheinlichkeit von 100 % Prozent nur, dass der Schwellwert *irgendwo* eher deutlich unterhalb der präsentierten Stimulusintensität liegt. Beträgt die gemessene Antwortwahrscheinlichkeit jedoch 50 %, und ist damit die gesuchte Schwelle in unmittelbarer Nähe der präsentierten Stimulusintensität zu suchen, so wird hier ein sehr viel kleiner Bereich für die Lage der tatsächlichen Schwelle spezifiziert – präziser spezifiziert. Da adaptive Verfahren nun genau darauf abzielen und dahingehend optimiert werden, Stimuli möglichst in der Nähe der zu messenden Schwelle zu platzieren, sind sie üblicherweise sehr effizient.[3]

Die Art und Weise, wie verschiedene adaptive Verfahren eine Entscheidung über die Platzierung des nächsten Stimulus treffen, variiert. QUEST und ZEST berechnen hierzu aus den früheren Antworten der Versuchsperson und weiteren Informationen über den Schwellwert den wahrscheinlichsten Ort der Schwelle mithilfe von Methoden der Bayes-Statistik.

2.2 Bayes-Statistik und Bayes-Theorem

Traditionelle Statistik behandelt als ihren Gegenstand die Wahrscheinlichkeit von zufälligen Ereignissen, wie sie aus Zufallsexperimenten hervorgehen, ihr Begriff von „Wahrscheinlichkeit“ kann als relative Häufigkeit gedeutet werden. Die Bayes-Statistik hingegen beschränkt sich nicht nur auf zufällige Ereignisse, sie zielt darauf ab, grundsätzlich für beliebige *Aussagen* mithilfe von Wahrscheinlichkeitsverteilungen der in diesen vorkommenden unbekannten Parameter Bereichsschätzungen, Parameterschätzungen oder Hypothesenprüfungen zu ermöglichen.

Sind unbekannte Parameter hier das Ziel des Interesses, so sind die von der

[3] King-Smith u. a., 1994, S. 886.

Bayes-Statistik untersuchten „Aussagen“ solche über die Werte der unbekannten Parameter. Die ermittelte Wahrscheinlichkeit bezüglich einer Aussage repräsentiert den Zustand des Wissens über diese Aussage, und so ist diese Wahrscheinlichkeit in der Bayes-Statistik nicht bloß als relative Häufigkeit zu sehen, sondern als Auskunft über die *Plausibilität einer Aussage*.[4]

Um diese Wahrscheinlichkeiten nun angeben zu können, wird zunächst der Begriff der *bedingten Wahrscheinlichkeit*

$$P\left(A|B\right) \tag{2.1}$$

eingeführt. Sie liefert ein Maß für die Plausibilität der Aussage, dass A wahr ist unter der Vorbedingung, dass B wahr ist.[5] Mithilfe der *bedingten Wahrscheinlichkeit* lässt sich empirisches Wissen ausdrücken – B steht für vorhandenes Wissen, $P\left(A|B\right)$ steht für die Wahrscheinlichkeit, mit der vorhandenes Wissen für weiteres Wissen relevant ist.

Quantitative Gesetzte für die bedingte Wahrscheinlichkeit lassen sich allein durch logisches und konsistentes Schließen aus der klassischen Aussagenalgebra herleiten, und es ergibt sich, dass die Wahrscheinlichkeit auch für die Bayes-Statistik durch reelle Zahlen ausgedrückt wird und zwischen Null und Eins liegt.[6]

Desweiteren lässt sich durch verschiedene Ableitungen und Umformungen das sehr grundlegende Produktgesetz der Wahrscheinlichkeit der Bayes-Statistik ge-

[4] Koch, 2000, S. 1 ff.

[5] Ein Beispiel: Die Plausibilität der Aussage, dass ein Einbruch stattgefunden hat unter der Bedingung, dass die Alarmanlage heulte. Deutlich wird an dieser Stelle auch der Zusammenhang mit der klassischen Statistik: Deren „Wahrscheinlichkeit, eine 3 zu würfeln“ wird im Rahmen der Bayes-Statistik als bedingte Wahrscheinlichkeit formuliert: Die Wahrscheinlichkeit, eine 3 zu würfeln unter der Bedingung, dass der Würfel symmetrisch ist. Siehe Koch, 2000, S. 5 f.

[6] Koch, 2000, S. 7.

winnen,

$$P(AB|C) = P(A|C)\,P(B|AC) = P(B|C)\,P(A|BC) \tag{2.2}$$

welches eine Beziehung herstellt zwischen der Plausibilität der Konjunktion AB der beiden Aussagen[7] und der Plausibilität der Aussage A und der Plausibilität der Aussage B unter der Bedingung, dass Aussage C wahr ist.

Wird Gleichung 2.2 durch Auflösung nach $P(A|BC)$ umgeformt, ergibt sich das *Bayes-Theorem*:

$$P(A|BC) = \frac{P(A|C)\,P(B|AC)}{P(B|C)} \tag{2.3}$$

Hierbei steht A üblicherweise für die Aussage über ein unbekanntes Phänomen (den interessierenden Untersuchungsgegenstand), B für Aussagen, die Informationen über A enthalten (z. B. empirisches Wissen) und C für (allgemeines, grundlegendes) Hintergrundwissen.

$P(A|C)$ wird sodann als *apriori-Wahrscheinlichkeit* bezeichnet, $P(A|BC)$ als *aposteriori-Wahrscheinlichkeit*, $P(B|AC)$ schließlich als *Likelihood*.[8] [9]

Um die wahre Bedeutung und die bemerkenswerten Möglichkeiten des Bayes-Theorems zu verstehen, sei hier zuerst kurz auf den Unterschied zwischen deduktiven und induktiven Schließen verwiesen: Geht ersteres nach den einfach beschreibbaren und zu operationalisierenden Regeln der Aussagenlogik von vorgefundenen

[7] Die *Konjunktion* AB ist zu lesen als: „Sowohl A als auch B ist wahr“, sowie entsprechend $P(AB|C)$: „Die Wahrscheinlichkeit, dass sowohl A als auch B wahr ist unter der Bedingung, dass C wahr ist“.

[8] Die Begriffe *apriori* und *aposteriori* gehen eigentlich auf Immanuel Kant zurück, im Zusammenhang mit der Bayes-Statistik ist eine gekürzte Nomenklatur (*priori, posteriori*) ebenfalls nicht unüblich.

[9] Die Wahrscheinlichkeit $P(B|C)$ im Nenner des Bayes-Theorems trägt keine gesonderte Bezeichnung, tatsächlich kann sie rein rechnerisch als Normierungskonstante betrachtet und häufig einfach vernachlässigt werden.

Ursachen durch Schlüsse zu daraus resultierenden Wirkungen über, so stellt induktives Schließen den Versuch dar, von (beobachteten) Auswirkungen auf deren mögliche Ursachen zu schließen. Der unschätzbare Wert des Bayes-Theorems besteht nun darin, genau für diese Aufgabe ein mathematisch-wissenschaftliches Verfahren zur Verfügung zu stellen. Denn umgangssprachlich ausformuliert beantwortet das Bayes-Theorem die Frage: „Wie wahrscheinlich ist es, gegeben meine grundlegenden Kenntnisse C über einen allgemeinen Zusammenhang, bei gleichzeitigem Vorliegen meiner Beobachtungen B, dass A hierfür als Ursache in Frage kommt?“ mit der Antwort: “Das hängt zum einen davon ab, wie wahrscheinlich das Vorkommen der vermeintlichen Ursache A im Zusammenhang mit dem grundlegenden Zusammenhang C überhaupt ist und zum anderen von der Wahrscheinlichkeit, die Daten B sammeln zu können, wenn sowohl die „Ursache“ A als auch der grundlegende Zusammenhang C gegeben ist.“

Diese drei Terme werden nun von QUEST und ZEST wie folgt besetzt:

- Die apriori-Wahrscheinlichkeit $P(A|C)$ entspricht dem (allgemeinen) Wissen um die Größe des Schwellwerts im Rahmen der aktuellen Versuchsbedingungen. QUEST nutzt hierzu eine Gaussverteilung (und schlägt alternativ eine Rechteckverteilung vor), ZEST verwendet eine modifizierte hyperbolische Sekante, modelliert nach Verteilungen, welche in früheren Experimenten gewonnenen wurden. Siehe Kapitel 2.3.

- Als Likelihood $P(B|AC)$ verwenden beide Verfahren das Ergebnis der Modifikation einer psychometrischen Funktion durch die bisherigen Antworten der Versuchsperson während des aktuellen Hörversuchs. Eine *psychometrische Funktion* beschreibt den Zusammenhang zwischen Stimulusintensität und der Wahrscheinlichkeit einer bestimmten[10] Reaktion einer (beliebigen)

[10] Im Falle eines „Ja/Nein“-Paradigmas z. B. wäre dies eine positive Antwort.

Versuchsperson. Als psychometrische Funktion verwenden beide Verfahren (in leicht unterschiedlicher Ausprägung) die Weibull-Funktion. Siehe Kapitel 2.4.

- Die ermittelte a-posteriori-Wahrscheinlichkeit $P(A|BC)$ schließlich gibt die Verteilung wieder, die die Wahrscheinlichkeit für das tatsächliche Vorliegen der Schwelle als Ursache für die gemessenen Daten beschreibt. Mithilfe dieser Verteilung entscheiden QUEST und ZEST, wo der nächste Stimulus zu platzieren ist.[11] QUEST berechnet hierzu ihr Maximum, ZEST ihr arithmetisches Mittel.

2.3 A-priori-Wahrscheinlichkeitsdichtefunktion

Wie bereits erwähnt, repräsentiert die A-priori-Wahrscheinlichkeitsdichtefunktion des Bayes-Theorems im Rahmen von QUEST und ZEST ein „allgemeines“ Wissen um Wahrscheinlichkeiten verschiedener Größen des Schwellwertes.

Ein sehr einfacher Weg, Informationen hierüber bereitzustellen, besteht darum in der Verwendung einer Rechteckverteilung – die Wahrscheinlichkeiten für alle Schwellwertgrößen wird damit als gleich angenommen. Es ist offensichtlich, das diese Angabe nur von geringem Informationsgehalt ist, ihre Verwendung als a-priori-pdf[12] im Rahmen von QUEST/ZEST widerspricht zwar nicht den Bedingungen des Verfahrens, lässt aber durchaus die berechtigte Frage zu, inwiefern durch sie tatsächlich Hintergrund*wissen* ausgedrückt wird und in das Verfahren mit einfließt.

Die Verwendung einer Gauss-Verteilung, wie sie von QUEST genutzt wird, hat dem gegenüber nun den Vorteil, mittlere Schwellwertgrößen graduell zu gewichten

[11] Nämlich eben möglichst nahe am „wirklichen“ Schwellwert, vergleiche Kapitel 2.1.
[12] „*P*robability *D*ensity *F*unction“, Wahrscheinlichkeitsdichtefunktion.

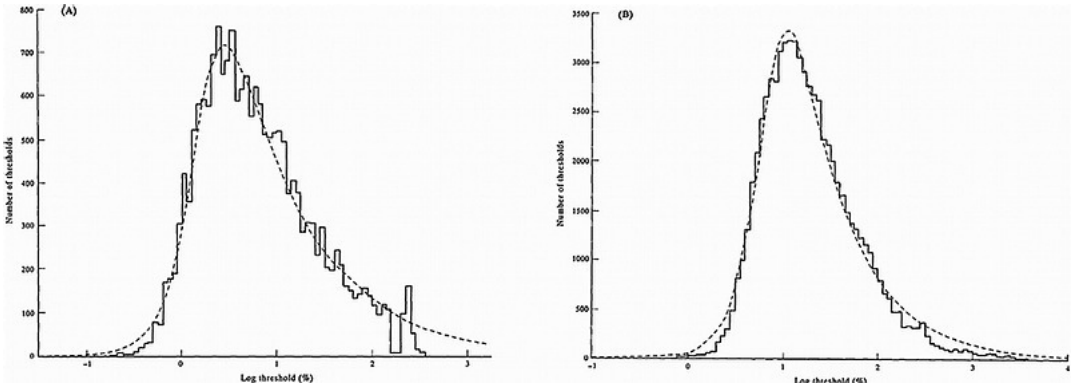


Abbildung 2.1: Zwei Schwellwerthistogramme vorangegangener Versuche mit angelegten Funktionen gemäß Gleichung 2.4.

und so spezifischere Informationen beizutragen, ihre Abhängigkeit, vom Versuchsleiter gemäß seiner Erfahrung erstellt werden zu müssen,[13] birgt potentiell jedoch ungünstigerweise praktische Schwierigkeiten. Nicht abschließend geklärt ist auch bei ihr die zwingende Eignung als Ausdruck von Wissen um Wahrscheinlichkeiten verschiedener Schwellwertgrößen, es bleibt ein Rest an Erklärungsbedarf ob der idealen Eignung als A-priori-Wahrscheinlichkeitsdichtefunktion.

ZEST nun gelingt es dieser Anforderung zu genügen, indem Histogramme von Schwellwertmessungen früherer Testdurchläufe und Versuche approximiert werden, und dies anhand der folgenden Funktion:

$$q_0(T) = A/[Be^{-C(T-t)} + Ce^{B(T-t)}] \tag{2.4}$$

mit T der logarithmierten Schwellwertgröße, A der Höhe der Kurve, B der Steigung der Kurve im Bereich kleiner Schwellwerte, C der Steigung der Kurve im Bereich großer Schwellwerte und t dem Maximum der Funktion, der Schwellwertgröße mit der höchsten Wahrscheinlichkeit.

Abbildung 2.1 zeigt zwei Histogramme von Schwellwertgrößen vorangegangener

[13] Siehe King-Smith u. a., 1994, S. 888.

Experimente nebst ihnen angepasster Funktion – der augenfällige Unterschied zu einer Gauss-Verteilung besteht in ihrer (potentiellen) Asymmetrie. Da diese Verteilungen empirisch aus tatsächlichen Tests hervorgegangen sind und solch asymmetrische Verteilungen von Schwellen also *tatsächlich* vorkommen, bestätigt sich hier die Skepsis gegenüber der bevorzugten Nutzung einer Gauss-Verteilung als apriori-pdf, die ein im Verhältnis zu niedrigen vermehrtes Vorkommen von großen Schwellwerten abzubilden nicht im Stande ist.

Die Autoren von ZEST weisen auf eine mögliche Allgemeingültigkeit ihrer angenäherten Apriori-pdf aufgrund der großen Ähnlichkeit der ermittelten zwei Kurven trotz ihrer Gewinnung aus ganz unterschiedlichen Tests hin, warnen vor der unkritischen Annahme, die konkreten Werte ihrer Kurve für jede mögliche psychometrische Messung für geeignet zu halten und machen abschließend deutlich kenntlich, dass die Benutzung einer an tatsächliche Schwellwertverteilungen angepassten Apriori-pdf die Effizienz des Verfahrens erhöht und dessen systematischen Fehler (*bias*) minimiert.

(Es gilt zu bedenken, dass im Umkehrschluss hieraus zwar das *Caveat!* folgt, dass die Wahl einer nicht auf die tatsächlichen Schwellwertverteilungen abgestimmten Apriori-Wahrscheinlichkeitsdichtefunktion den *bias* des Verfahrens zwar *vergrößert*, dies jedoch mit wachsender Zahl an Trials zunehmend geringer ins Gewicht fällt.)[14]

2.4 Psychometrische Funktion und Likelihood

QUEST und ZEST nutzen zur Bestimmung der *Likelihood* des von ihnen angewendeten Bayes-Theorems als psychometrische Funktion die Weibull-Funktion.[15]

[14] Siehe King-Smith u. a., 1994, S. 910.
[15] Weibull, 1951.

Gleichung 2.5 zeigt sie in ihrer von ZEST implementierten Fassung:

$$\Psi\left(x-T\right)=1-\lambda-\left(1-\gamma-\lambda\right)exp[-10^{\beta(x-T+\epsilon)}] \tag{2.5}$$

Dabei bestimmt der Parameter β die Steigung der psychometrischen Funktion.

Die sogenannte *guessing rate* oder auch *false positive rate* γ stellt die untere Asymptote dar – ihr Name rührt daher, dass bei Stimuluspräsentation deutlich unterhalb der gesuchten Schwelle bei erzwungener Antwort richtige Antworten der Versuchsperson als nur erraten gewertet werden. Der Wert der *guessing rate* ist fest mit dem jeweils gewählten Paradigma verbunden, bei forced-choice-Paradigmen berechnet sie sich aus dem Kehrwert der Alternativen, beim „Ja/Nein"-Paradigma wird sie als Rauschen interpretiert und auf einen eher niedrigen Wert wie z. B. 0,03 eingestellt.

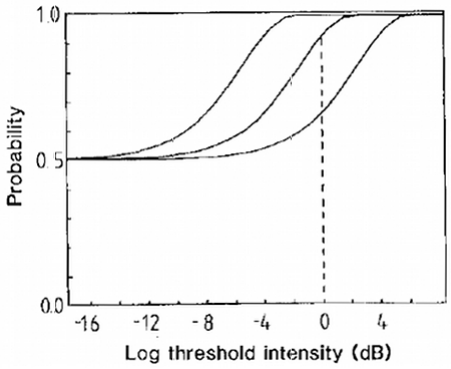


Abbildung 2.2: Die Weibull-Funktion, Variation von T.

Die *lapsing rate* oder *false negative rate* λ stellt die obere Asymptote der psychometrischen Funktion, sie erhält ihren Namen daher, dass fehlerhafte Antworten selbst bei hohen Stimulusintensitäten auf kurzzeitige Unaufmerksamkeiten der Versuchspersonen zurückgeführt werden. Sie ist unabhängig vom Paradigma, festgesetzt wird sie üblicherweise auf etwa 0,01.

T für *threshold* repräsentiert die (unbekannte) Schwelle, der Parameter ϵ ist in der Lage, diese zu verschieben und damit zu beeinflussen, als welches Kriterium die Schwelle der psychometrischen Funktion betrachtet wird. (Mehr hierzu weiter unten in Kapitel 2.4.1.)

Abbildung 2.2 ist eine Darstellung der Weibull-Funktion mit verschiedenen Werten für den Thresholdparameter T. (Deutlich zu sehen ist außerdem die un-

tere Asymptote – da sie bei 0,5 liegt, muss es sich um ein 2AFC-Paradigma handeln. Etwas weniger deutlich tritt am oberen Rand die *lapsing rate* in Erscheinung.) Die Abbildung deutet ebenfalls schon jene wichtige, fundamentale Eigenschaft der psychometrischen Funktion an, die sich sowohl QUEST als auch ZEST zunutze machen, um auf sehr ökonomische Weise nach jedem Trial eines Hörversuchs aus allen bisherigen Antworten des Versuchsteilnehmers in Verbindung mit der psychometrischen Funktion die Likelihood zur Ermittlung der Aposteriori-Wahrscheinlichkeit zu berechnen – denn für verschiedene (grundsätzlich logarithmisch skalierte) Schwellwerte unterscheidet sich die psychometrische Funktion niemals in Hinsicht auf ihre Form, sondern nur in Hinsicht auf ihre Position entlang der x-Achse.

Wie bereits oben eingeführt, versteht sich die Likelihood eines Trials als die Wahrscheinlichkeit, ein bestimmtes Datum zu beobachten (d. h. eine bestimmte Antwort von der Versuchsperson zu erhalten), gegeben die Tatsache, dass eine (von uns vermutete) Ursache (wirklich) vorlag (d. h., dass sich an der untersuchten Stelle auch eine Schwelle befand) und die Hintergrundinformationen zutreffen (d. h., die Angaben der psychometrischen Funktion zutreffend sind).

Diese gesuchte Wahrscheinlichkeit wird nun aber gerade genau von der psychometrischen Funktion geliefert, und zwar dann, wenn man sie an die Stelle T der dargebotenen Reizintensität verschiebt; Denn dann gibt sie – laut Definition – Auskunft über die Antwortwahrscheinlichkeiten bezüglich aller möglicher Reizintensitäten, vorausgesetzt, die Schwelle liegt bei T. Ist das Einbeziehen mehrerer vorhergehender Trials von Interesse, so ergibt sich die Likelihood einfach jeweils durch Multiplikation der verschobenen psychometrischen Funktion mit den Werten der ein Trial zurückliegenden Likelihood.[16]

[16] Der Übersichtlichkeit halber wurde hier nicht explizit differenziert zwischen positiven Antworten und negativen Antworten; Nur im ersten Fall wird einfach die psychometrische Funktion

2.4.1 Epsilon und ideal sweat factor

Wie bereits im vorigen Abschnitt eingeführt, ist die Variable ϵ der Weibull-Funktion geeignet, jenen Punkt zu verschieben, an dessen Lokation der sogenannte *threshold* der Funktion verortet wird. Ist man im Prinzip frei in der Wahl des threshold und werden allenthalben ohne Begründung das Überschreiten bestimmter, „üblicher“ Wahrscheinlichkeitswerte zu dessen Verortung verwendet, so nutzt im Gegensatz hierzu QUEST ein plausibles Kriterium für die spezifische Auswahl eines bestimmten threshold.

Die Autoren setzen ϵ so, dass der threshold mit jenem Punkt zusammen fällt, der den *ideal sweat factor*[17] der psychometrischen Funktion darstellt.

Der *sweat factor* gibt als ein Maß für die *Effizienz* Auskunft über das Verhältnis des betriebenen Aufwands zu einer durch diesen erzielten Genauigkeit. Bezüglich von Testversuchen und -verfahren wird er üblicherweise als Produkt von Trialanzahl und Varianz errechnet, für die Beurteilung des effizienten Einsatzes psychometrischer Funktionen errechnet sich der *sweat factor* als quadratisches Verhältnis der binomialen Standardabweichung zur Steigung der psychometrische Funktion an der fraglichen Stelle:[18]

$$K(X) = \frac{\Psi(X)[1-\Psi(X)]}{[d\Psi(X)/dX]^2} \qquad \text{mit} \qquad X = x - T \tag{2.6}$$

Der ideale *sweat factor* ist durch minimales $K(X)$ gegeben.

Durch geeignete Wahl von ϵ lässt sich nun herbeiführen, dass dieser ideale *sweat factor* genau an der Stelle T der psychometrischen Funktion zu liegen kommt. Und da sich die Varianz einer großen Anzahl Trials einer gleichbleibenden Stimulusin-

$\Psi(x-T)$ verwendet, im Fall einer negativen Antwort werden deren Wahrscheinlichkeiten – plausiblerweise – durch $(1-\Psi(x-T))$ beschrieben.

[17] Taylor, 1971

[18] Taylor, 1971, S. 506, Watson und Pelli, 1983, S. 114, King-Smith u. a., 1994, S. 890.

tensität mit $K(X)/M$ errechnet,[19] sorgt $K_{min}(X)$ für größtmögliche Präzision.[20]

Setzt man dieses Vorgehen ein, so interpretiert sich der Threshold T einfach als rein technisches Effizienzmerkmal.[21]

2.5 Trialplatzierung, abschließende Schwellwertschätzung

Wie dargelegt berechnen QUEST und ZEST prinzipiell auf gleiche Weise aus der apriori-pdf und der Likelihood eine aposteriori-pdf – unterschiedlich eruieren sie sowohl, wo der nächste Stimulus platziert werden soll als auch, wie die abschließende Berechnung der Schwelle erfolgt.

QUEST berechnet das Maximum der aposteriori-pdf und nutzt diesen Wert zur Auswahl des nächsten Stimulus. ZEST hingegen stützt sich hierzu auf das arithmetische Mittel der aposteriori-pdf. Vergleichende Evaluationen ließen schon die Autoren von ZEST zu dem Schluss kommen, dass die Wahl des arithmetischen Mittelwerts statt des Maximums unter allen untersuchten Paradigmen[22] zu größerer Präzision führt.

Zur finale Schätzung des Schwellwertes wird im Rahmen des QUEST-Verfahrens die abschließend ermittelte aposteriori-pdf durch die initiale Likelihood dividiert. Die Autoren wollen so sicherstellen, dass jegliche Annahmen bezüglich der initialen Likelihood von der Schätzung des Schwellwertes ausgenommen werden.

[19] Taylor, 1971, S. 507.

[20] Auch für nicht-gleichbleibende Stimulusintensitäten ermitteln die Autoren von ZEST empirisch, dass eine optimale Größe für ϵ auf der Minimierung von $K(X)$ beruht – die einzige Ausnahme bilden Hörversuche von nur 8 – 16 Trials; hier verhält sich $\epsilon = 0$ effizienter.

[21] Die Wahrscheinlichkeiten der psychometrischen Funktion an solch optimierter Stelle $T = K_{min}(X)$ differieren übrigens nicht wenig von den weiter oben kurz angesprochenen, sonst „üblichen“ z. B. 50 % bei „Ja/Nein“ und 75 % bei 2AFC: Die Autoren von QUEST nennen 0,796 und 0,92, jene von ZEST 0,78 und 0,897!

[22] Explizit waren dies „Ja/Nein“ und 2AFC. Siehe King-Smith u. a., 1994, S. 906 et passim.

Teils aus praktischen Gründen, teils unter der empirisch gewonnenen Erkenntnis, dass abschließende Schwellwertschätzungen bei „vernünftiger“ Auswahl einer ursprünglichen Wahrscheinlichkeitsdichtefunktion typischerweise wenig beeinflusst werden,[23] wird im Rahmen des ZEST-Verfahrens hiervon abgesehen.

[23] King-Smith u. a., 1994, S. 81 et passim.

3 Implementierungen

3.1 Programmiertechniken in Matlab

Die Hörversuchsumgebung Whisper ist in der Programmiersprache Matlab mithilfe der gleichnamigen Entwicklungsumgebung implementiert worden. Eine allgemeine Einführung in diese Programmiersprache ist nicht Aufgabe dieser Arbeit, zudem ist der Erwerb zumindest grundlegender, wenn nicht sogar weiterführender Programmierfähigkeiten in Matlab Teil der Lehrinhalte des Fachgebiets Audiokommunikation.

Die Programmierung komplex verknüpfter graphischer Benutzeroberflächen (sogenannter *GUIs*) sowie deren spezifischer Einsatz und ideosynkratischer Gebrauch von Whisper geht jedoch deutlich über diese Grundlagen hinaus. Eine Einführung in die wichtigsten der verwendeten Programmiertechniken ist so außer, dass sie Nutzen birgt für zukünftige an Whisper arbeitende Programmierer, wichtig für das Verständnis bestimmter Aspekte der hier dokumentierten Erweiterung.

3.1.1 GUIDE

„Herkömmliche“ Matlab-Skripte bestehen aus in sogenannten *m-files* – Dateien der Dateinamenserweiterung „.m“ – niedergeschriebenen Funktionen. Graphische Oberflächen bestehen in Matlab aus einem Pärchen von Dateien: Einer „.fig“-Datei (in einem binären Format, sie enthält alle relevanten Parameter des erstellten GUIs) sowie einer „.m“-Datei im üblichen Textformat, die alle Funktionalität

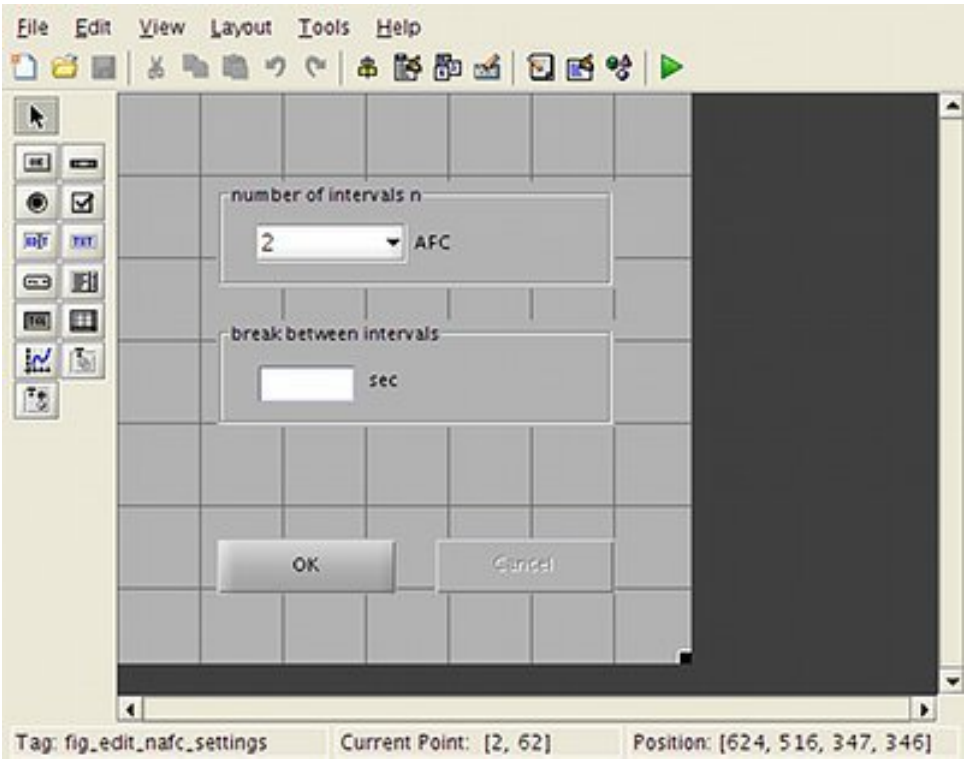


Abbildung 3.1: Der matlabeigene GUI-Editor *GUIDE*.

des besagten GUIs enthält und wie üblich mit dem matlabeigenen Texteditor bearbeitet werden kann.

Das Erzeugen, Verändern und mit Funktionalität Versehen von Bedienelementen gestaltet sich wie folgt:

· Geöffnet wird eine „.fig“-Datei nicht wie „.m“-Dateien durch eine Doppelklick im Dateifenster des Matlab-Workspace – sie muss selektiert und durch Rechtsklick mit der darauf erscheinenden Option `Open in GUIDE` in der dann erscheinenden GUIDE-Oberfläche (siehe Abbildung 3.1) zur Bearbeitung verfügbar gemacht werden. An dessen linkem Rand befinden sich die Buttons zur Auswahl der zur Erstellung zur Verfügung stehenden GUI-Elemente (*pushbutton, listbox, radiobutton, checkbox* etc.). Ein Klick auf einen von ihnen und ein anschließender Klick

an eine beliebige Stelle im mit grauem Raster hinterlegten GUI-Fenster legt dieses Bedienelement neu an, es kann anschließend bewegt und mithilfe der am oberen Rand der GUIDE- Oberfläche erreichbaren Werkzeuge graphisch ausgerichtet werden. Wichtig ist nun Folgendes: Jedes auch schon vorhandene Bedienelement kann durch einen Klick selektiert werden,[1] nach anschließendem Rechtsklick darauf und Auswahl des Menüpunktes `Property Inspector` erscheint das Fenster des *Property Inspector*, siehe Abbildung 3.2. Hier können Attribute des ausgewählten GUI-Elements verändert und überprüft werden. Essentiell sind für die Programmierung von Whisper:

`Tag` Dies ist der „Name“ des Bedienelements, über den später *alle* Zugriffe auf es erfolgen. Die von Matlab hier vergebene Standardbezeichnung bei neuerstellten GUI-Elementen muss geändert werden gemäß der in der *Technischen Dokumentation* von Whisper[2] vorgegebenen Namenskonventionen (*pb_mein_pushbutton* für den Pushbutton „Mein Pushbutton“, *ed_eingabe* für das Edit-field „Eingabe“ etc.), ein späteres einander Zuordnen wird andernfalls überaus schwierig, wenn nicht unmöglich.

`FontSize` Matlab-Vorgabewert ist 10, Whisper verwendet 9.

`String` Die im GUI sichtbare Bezeichnung des Bedienelements.

`Max` (Nur bei der Edit-Box: Ein Wert größer 1 schaltet die mehrzeilige Eingabe frei.)

Nach Veränderungen an den Bedienelementen sollte die betreffende „.fig“-Datei über die entsprechenden Optionen im GUIDE-Fenster abgespeichert werden, Matlab speichert auch die zugehörige „.m“-Datei (dazu gleich mehr).

[1] An dieser Stelle irrelevant, der Vollständigkeit halber dennoch: Auch der „Hintergrund“ des Fensters ist ein GUI-Element.

[2] Wlodarski, 2008b, S. 7 f.

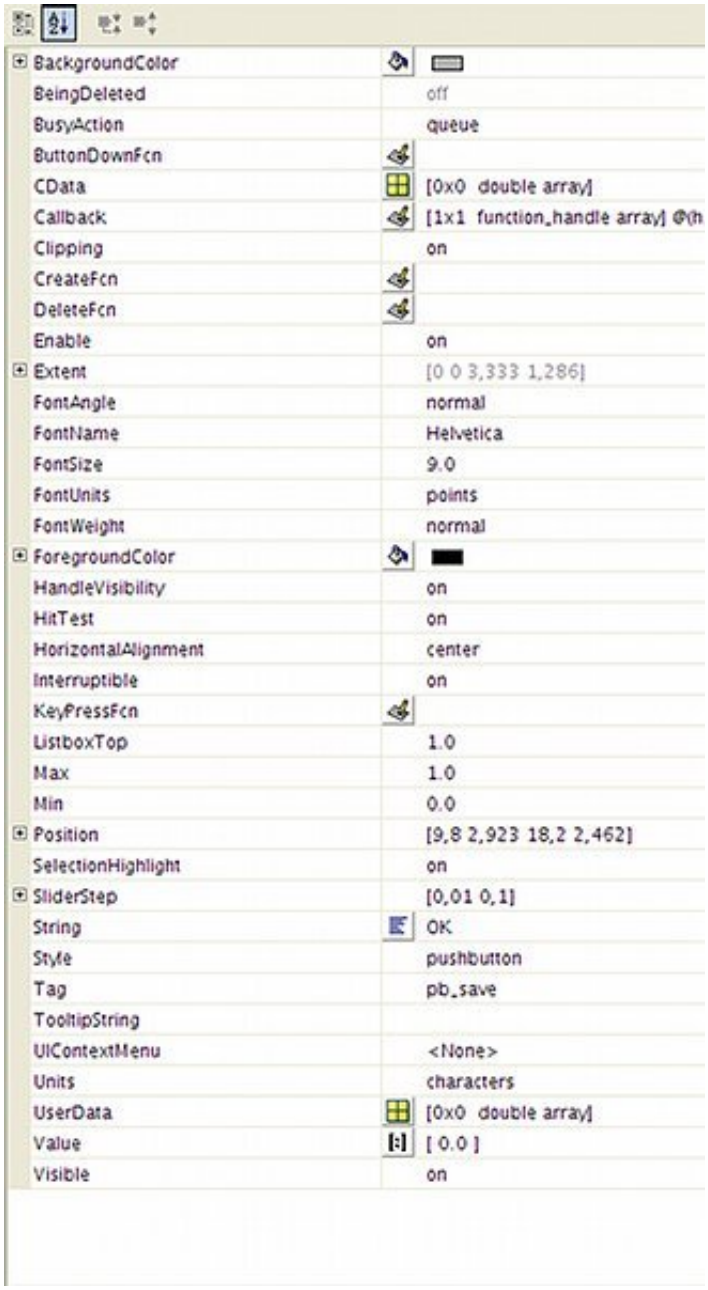


Abbildung 3.2: Der *Property Inspector* graphischer Bedienelemente, hier eines OK-Buttons.

Die gewünschte Funktionalität nun für ein Bedienelement wird über sogenannte *Callbacks* realisiert.

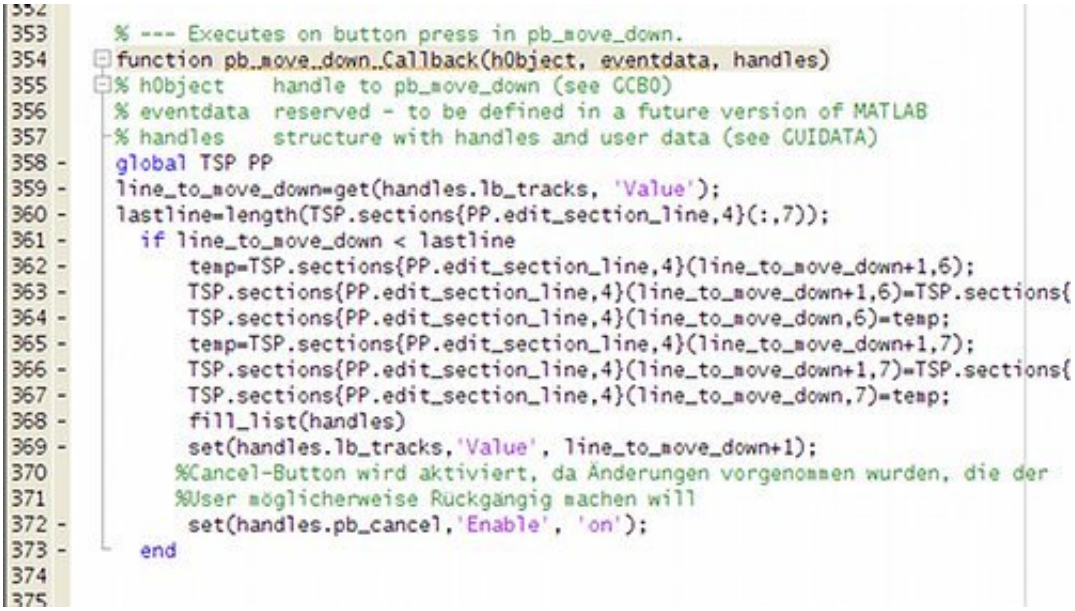
3.1.2 Callbacks

Callbacks sind Funktionen, die bei Betätigung von GUI-Elementen einer in Matlab selbsterstellten Oberfläche automatisch aufgerufen werden. Jedem GUI-Element sind Callbacks zugeordnet, und zwar mehrere. Alle zu einem Bedienelement gehörenden Callback-Funktionen unterscheiden sich darin, dass es unterschiedliche Benutzeraktionen sind, durch die sie ausgelöst werden.

Ein Rechtsklick im GUIDE-Fenster auf ein Bedienelement lässt den Menüpunkt `View Callbacks` sowie die Liste der zum ausgewählten Bedienelement gehörenden Callbacks erscheinen (bitte noch keinen auswählen).

Manche Namen dieser Callbacks verraten etwas über die sie auslösende Aktion: Der Programmcode der `KeyPressFcn` z. B. wird aufgerufen, sobald das entsprechende Bedienelement irgendeinen Tastendruck durch den Benutzer entgegennimmt, am wichtigsten ist jedoch die einfach nur mit `Callback` angezeigte Callback-Funktion – sie ist mit der spezifischen Betätigung ihres Bedienelements verknüpft. D. h. z. B., dass sie im Falle eines Buttons ausgelöst wird, sobald der Button betätigt wurde, bei einer *checkbox* (einem „Häkchen“), jedesmal, wenn es gesetzt oder entfernt wird, bei *edit-boxen* (Eingabefeldern), sobald man `Enter` gedrückt hat oder bei *Popup-Menüs*, sobald man eine Auswahl getroffen hat.

Alle Callbacks besitzen ganz normale `function` - Deklarationen, gerade so, wie man das für jede in Matlab selbst implementierte Funktion (in einem „.m-file“) gewohnt ist. Der `Callback`-Callback wird für jedes Bedienelement automatisch angelegt, sobald man dieses Bedienelement in GUIDE seiner Benutzeroberfläche hinzugefügt hat und die betreffende „.fig“-Datei speichert. (Alle anderen Callbacks werden von Matlab automatisch angelegt, sobald man das erste mal auf sie



```
353   % --- Executes on button press in pb_move_down.
354   function pb_move_down_Callback(hObject, eventdata, handles)
355   % hObject    handle to pb_move_down (see GCBO)
356   % eventdata  reserved - to be defined in a future version of MATLAB
357   % handles    structure with handles and user data (see GUIDATA)
358 - global TSP PP
359 - line_to_move_down=get(handles.lb_tracks, 'Value');
360 - lastline=length(TSP.sections{PP.edit_section_line,4}(:,7));
361 -   if line_to_move_down < lastline
362 -       temp=TSP.sections{PP.edit_section_line,4}(line_to_move_down+1,6);
363 -       TSP.sections{PP.edit_section_line,4}(line_to_move_down+1,6)=TSP.sections{
364 -       TSP.sections{PP.edit_section_line,4}(line_to_move_down,6)=temp;
365 -       temp=TSP.sections{PP.edit_section_line,4}(line_to_move_down+1,7);
366 -       TSP.sections{PP.edit_section_line,4}(line_to_move_down+1,7)=TSP.sections{
367 -       TSP.sections{PP.edit_section_line,4}(line_to_move_down,7)=temp;
368 -       fill_list(handles)
369 -       set(handles.lb_tracks,'Value', line_to_move_down+1);
370       %Cancel-Button wird aktiviert, da Änderungen vorgenommen wurden, die der
371       %User möglicherweise Rückgängig machen will
372 -     set(handles.pb_cancel,'Enable', 'on');
373 -   end
374
375
```

Abbildung 3.3: Automatisch gefundener, geöffneter und selektierter Callback im Editor.

zugreift, z. B. via des oben aufgezeigten Rechtsklickmenüs.)

Die `function` - Deklarationen der Callbacks befinden sich (dann) im zur „.fig“-Datei gehörigen „.m“-Datei, komfortablerweise ist es jedoch nicht nötig, sie „zu Fuß“ darin zu suchen: Der über das per Rechtsklick auf das gewünschte Bedienelement aufgerufene `View Callbacks`-Untermenü angewählte Callback wird von Matlab automatisch in der richtigen Datei gefunden, diese im Editor geöffnet und die zum gewählten Callback gehörende Zeile mit der `function` - Deklarationen selektiert, siehe Abbildung 3.3.

3.1.3 Callbacks in .m-files, Handles

Callbacks in „.m-files“ sind wie gesagt zunächst ganz normale Matlab-functions. Als solche können sie ganz normalen Matlab-Code enthalten, sowie in Funktionen ausgelagerte aber im gleichen „.m-file“ befindliche Hilfsfunktionen, etc., ganz, wie

bei „normalen“ „.m-files“ sonst auch.[3]

In Abbildung 3.3 kann man sehen, dass die Callbacks gemäß des *Tags* ihres zugehörigen Bedienelements benannt sind – einer der Gründe für eine klare Namensgebung. An der ursprünglich selektierten Zeile des Callbacks, der `function`-Deklaration, muss *nie* etwas verändert werden, aller eigener Programmcode folgt danach.

Auf die „Werte“ *aller* anderen Bedienelemente *des aktuellen GUI* lässt sich über deren *Handles* zugreifen:

```
get(handles.lb_tracks, 'Value')
```

beispielsweise greift auf die Nummer der aktuell selektierten Zeile (`'Value'`) der *Listbox* mit dem vorher vergebenen *Tag* `lb_tracks` zu und ermöglicht es, diesen Wert z. B. in einer Variable zu speichern.

– Die *handles*-Struktur enthält automatisch für jedes vorher definierte (und hoffentlich per *Tag* eindeutig umbenannte) Bedienelement einen der *Tag*-Bezeichnung entsprechenden Eintrag und sie ist in jedem Callback automatisch verfügbar. –

Der Zugriff auf `'Value'` würde bei einer *checkbox* oder einem *radio button* deren Zustand (gedrückt bzw. Häkchen gesetzt vs. nicht gedrückt oder nicht gesetzt) als „0“ oder „1“ zurückgeben.

Auf Einträge in *edit-boxen* würde man nicht über `'Value'` zugreifen, sondern über `'String'`.[4]

Auf alle eben genannten Attribute aller im aktuellen GUI befindlichen Be-

[3] Der Aufruf einer erstellten GUI funktioniert übrigens auch der „normalen“ selbstgeschriebenen Matlab-Funktion entsprechend – liegen „.fig“- und zugehörige „.m“-Datei im aktuellen Verzeichnis oder Pfad, so reicht zum Aufruf dieser GUI die Angabe ihres Namens (ohne Dateiendung).

[4] In den automatisch zusammen mit der function-Deklaration auch erzeugten Kommentaren weist Matlab in Stichworten auf die spezifische Zugriffsweise für eben den aktuellen Typ von Bedienungselement hin.

dienelemente kann man jedoch nicht nur lesend, sondern auch verändernd zugreifen, der Befehl hierzu heißt `set` und um automatisch die dritte Zeile einer Listbox auszuwählen, würde man folgenden Befehl in den Quellcode aufnehmen: `set(handles.lb_tracks, 'Value', 3)`.

Läuft die Kommunikation innerhalb der Bedienelemente eines GUI-Fensters wie eben beschrieben ab, so erfolgt der Zugriff auf Daten von „außerhalb“ des aktuellen Fensters in Whisper grundsätzlich per als *global deklarierter Variablen*, in Abbildung 3.3 sind die `TSP` und `PP` zu erkennen.[5]

Hingewiesen sei noch auf eine gewiss beim ersten Entdecken stutzig machende Besonderheit von Whisper bei der Nutzung von Callbacks in primär zur Konfiguration genutzten GUIs. Denn mitunter fehlt im zugehörigen „.m-file“ *jederr* Programmcode in sämtlichen Callbacks völlig – mit großer Ausnahme des `pb_ok_Callback`-Callback (oder auch `pb_save_Callback`); in diesem findet sich dann die Abfrage aller anderen Bedienelemente (über deren *handles*) sowie die Speicherung der abgefragten Werte. Es verwundert, aber auch seit der automatischen Erzeugung durch Matlab unveränderte und scheinbar ungenutzte Callbacks dürfen nicht gelöscht werden – der automatische Aufruf bei Betätigung des Bedienelementes würde sonst ins Leere laufen und einen Fehler produzieren.

3.1.4 Spezialfälle

Das GUI-Element der *Button-Group* erlaubt es, mehrere (auch verschiedenartige) innerhalb ihrer Ausdehnung im GUI angelegte Buttons komfortabel zusammenzufassen derart, dass immer nur einer von ihnen angewählt, „gedrückt“ sein kann, und der vormals „gedrückte“ von selbst „herausspringt“. Die Aktionen, die das Auslösen eines dieser Button in Gang setzen soll, werden nun nicht (individu-

[5] Alle global genutzten Variablen listet Wlodarski, 2008b, S. 9 ff.

ell) über den Callback des Buttons, sondern übergeordnet über die sogenannte *Selection_Change_Function* der *Button-Group* organisiert. Klare Beispiele hierzu finden sich in der in Matlab integrierten Hilfe.

Eine weitere, ganz spezielle Besonderheit findet sich in Verbindung mit der Nutzung einer externen Erweiterung zu Matlab der hier vorgestellten Erweiterung von Whisper: *Tabpanel* erlaubt das Anlegen von sogenannten *Tabs*, Karteireitern, in mit Matlab erstellten GUIs. Der große Vorteil dieser Erweiterung ist, dass sie weiterhin das Konfigurieren und Editieren von GUI-Elementen mittels GUIDE erlaubt und nur für das zusätzliche Kreieren von Tabs verantwortlich ist – auch zum Ausführen fertig programmierter mithilfe von ihr erstellter GUIs ist diese Erweiterung nicht nötig.

Der Prozess, mithilfe dessen die Tabs erzeugt und bevölkert werden, ist jedoch etwas aufwändiger und bietet speziell eine Fallschlinge, die zu vermeiden die folgende Beschreibung helfen soll.

Dieser Prozess ist zweistufig: Im ersten Schritt wird in GUIDE eine leere graphische Oberfläche erzeugt, in der ein ebenfalls leeres Textfeld als Platzhalter für die späteren Tabs eingefügt wird. GUIDE wird geschlossen, Tabpanel wird aufgerufen und öffnet seinerseits die eben erzeugte Oberfläche, es lassen sich in der Größe des vorher angelegten Platzhalters beliebig viele Tabs erzeugen. Um diese nun mit Bedienelementen zu füllen, wird das gewünschte Tab durch Klick fokussiert und von Tabpanel an GUIDE zur Bearbeitung als temporäre graphische Oberfläche (in der Größe nur des Tabs, selbstverständlich) weitergereicht. Dort editiert man wie gewohnt, speichert, schließt GUIDE und lässt anschließend die nun wieder an Tabpanel übergebene temporäre Datei von Tabpanel in die binäre „.fig“-Datei der ganz zu Anfang erzeugten, den Platzhalter für die Tabs und nach dieser Aktion auch die Tabs selbst enthaltenden graphischen Oberfläche

einrechnen.

Der Weg, die so erzeugte, Tags enthaltende GUI zu bearbeiten, führt nun immer über Tabpanel – öffnet man die die Tabs (und ursprünglich den Platzhalter) enthaltende GUI direkt in GUIDE, so zeigt sich so immer noch nur der Platzhalter; alle über das Editieren einzelner Tabs mittels Tabpanel erzeugten Bedienelemente sind auch nur dann zum graphischen Editieren verfügbar, wenn die Gesamt-GUI zuerst in Tabpanel, danach einzelne Tabs von Tabpanel an GUIDE übergeben werden.

Das hierbei auftretende Problem betrifft die Callbacks: Der Standard-Callback wird zwar für jedes Bedienelement auch der Tabs im „.m-file“ erzeugt, das Erzeugenlassen beispielsweise einer `KeyPressFcn` in der weiter oben beschriebenen Weise führt jedoch zu einer Fehlermeldung – GUIDE ist nicht in der Lage, die ihm übergebene, temporäre, nur den Inhalt eines Tabs der Gesamt-GUI enthaltene „.fig“-Datei mit dem „.m-file“ der „Mutterdatei“ zu assoziieren.

Gelöst wird dieses Dilemma doch durch Öffnen der „Mutterdatei“ in GUIDE (die Platzhalteroberfläche erscheint), das Aufrufen des *Object Browser*[6], Markieren des gewünschten GUI-Elements und Aufruf der altbekannten Funktion `View Callbacks` etc. *aus dem Fenstermenü des GUIDE heraus.* Viola – die gewünschten Callbacks werden richtig erzeugt und sind einsatzbereit.

3.2 Implementierung von ZEST

Die Implementierung von ZEST profitierte vom grundsätzlichen Vorhandensein Bayes-basierter adaptiver Testverfahren in Whisper. Ergänzt wurden diese dennoch an mehreren entscheidenden Stellen, um a) eine wirklich der ursprünglichen Veröffentlichung ganz entsprechende Nutzung von ZEST zu ermöglichen, und b)

[6] Dieser listet hierarchisch alle zu einer GUI gehörigen Objekte auf, im vorliegenden Falle auch die nicht sichtbaren Inhalte der Tabs.

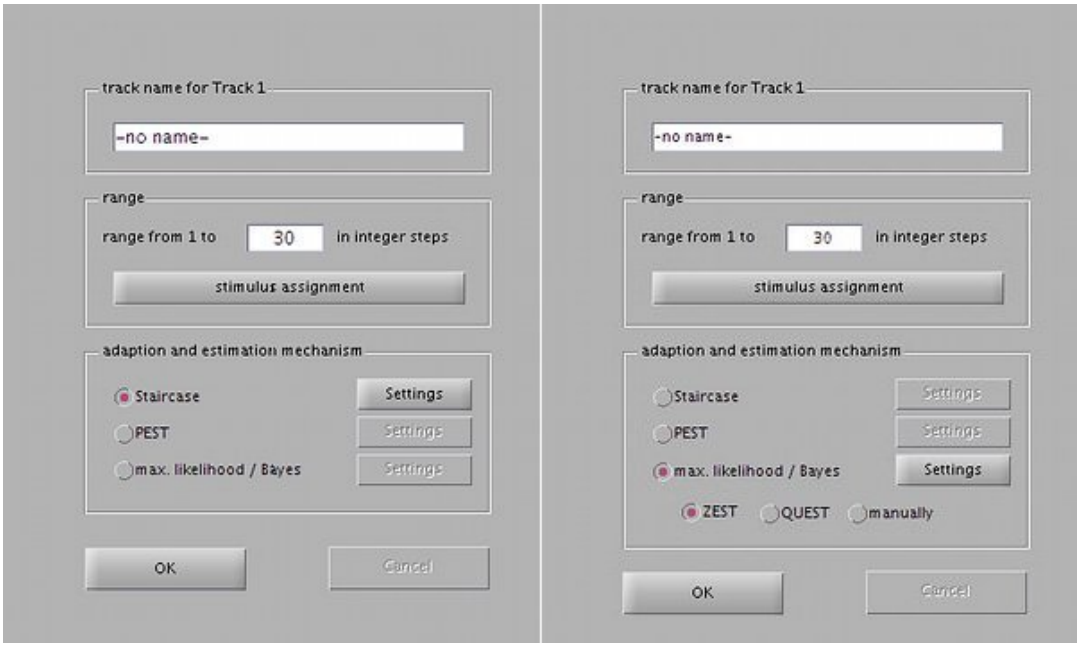


Abbildung 3.4: Alte (links) und neue Version von Whispers *Edit Track Settings* Dialog.

um die in Whisper bereits in dieser Richtung vorhandenen Möglichkeiten klarer herauszustellen und dem potentiellen Benutzer bei deren Beherrschung zur Seite zu stehen.

Der Ansatzpunkt in der graphischen Oberfläche von Whisper zu dieser Erweiterung ist die trackspezifische Konfiguration adaptiver Verfahren. Abbildung 3.4 zeigt links diesen Konfigurationsdialog vor der durchgeführten Änderung, rechts daneben danach. Ziel war, die hier bestehenden Möglichkeiten von Whisper nicht einzuschränken, jedoch trotzdem klarer als bisher diejenigen Einstellmöglichkeiten und Konfigurationen herauszustellen, die ein bestimmtes Verfahren charakterisieren – in erster Linie natürlich das ZEST-Verfahren, da die benötigten Änderungen jedoch durch Bereitstellung der weiteren Option QUEST nicht wesentlich aufwändiger wurden, findet sich auch hierfür nun eine eigene Vorwahlmöglichkeit.

Die neuen Auswahlmöglichkeiten „ZEST“, „QUEST“ und „manuell“ werden erst bei der übergeordneten Anwahl von „ML/Bayes“ aktiviert; Sie verändern die Optionen des nach ihrer Anwahl per Settings aufgerufenen ML/Bayes-Konfigurationsdialogs (diese sollten dort besprochen werden). „Manuell“ bietet die volle Konfigurationsfreiheit wie bisher, der Funktionsumfang von Whisper wurde nicht beschnitten. [7]

Nach vorgenommener Ausdifferenzierung der Wahl des Adaptionsmechanismus wurden die neuen Auswahlmöglichkeiten den Programmteilen bekannt gemacht, die die Wahl des Adaptionsmechanismus auswerten: Namentlich die Speicherung der Testseries-Informationen sowie die Durchführung eines adaptiven Testverfahrens. Für den Import alter, vor der Änderung angelegter Projekte wurde eine Kompatibilitätsfunktionalität implementiert, dazu später mehr.

Der erweiterte Konfigurationsdialog für das ZEST-Verfahren ist in Abbildung 3.6 zu sehen.

Dieser wurde gemäß den Anforderungen an einer möglichst vollständige Implementierung von ZEST erweitert und zeigt die hinzugekommenen Parameter.

Die vorgenommenen Änderungen im Detail:

Implementierung der Weibull-Funktion als psychometrische Funktion, Auswahlmöglichkeit zwischen logistischer und Weibull-Funktion. Die Auswahl von „manuell“ im vorherigen Auswahldialog für die adaptiven Mechanismen selektiert – wie bisher – automatisch „logistisch“, „QUEST“ und „ZEST“ selektieren „Weibull“ (wobei für jede der drei Optionen jeweils beide psychometrischen Funktionen selektierbar bleiben). Die Parameter β, γ, λ gemäß Kapitel 2.4 und Glei-

[7] Gespeichert wird die Wahl des Adaptionsmechanismus in `TSP.sections{n,4}{m,7}{2,1}`: Staircase(1), BestPEST(2), ALT: ML/Bayes(3), NEU: ZEST(31), QUEST(32), ML/BayesManuell(33).

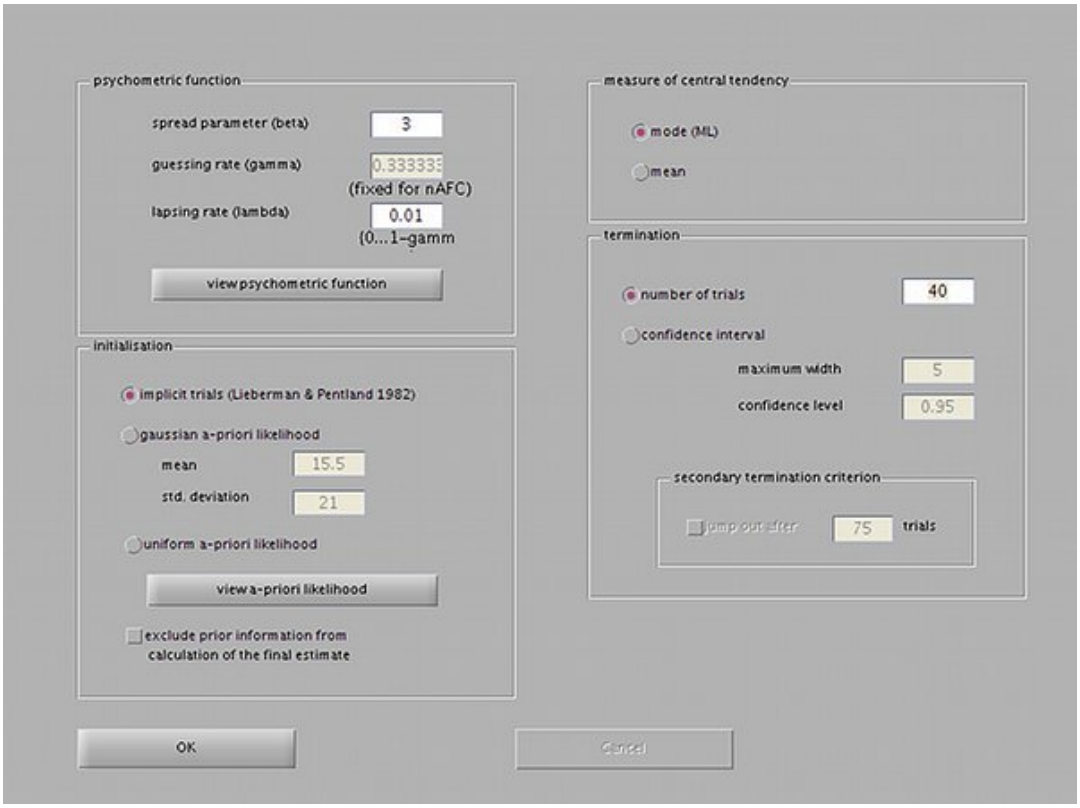


Abbildung 3.5: Alte Version von Whispers *Edit MLBayes Settings* Dialog.

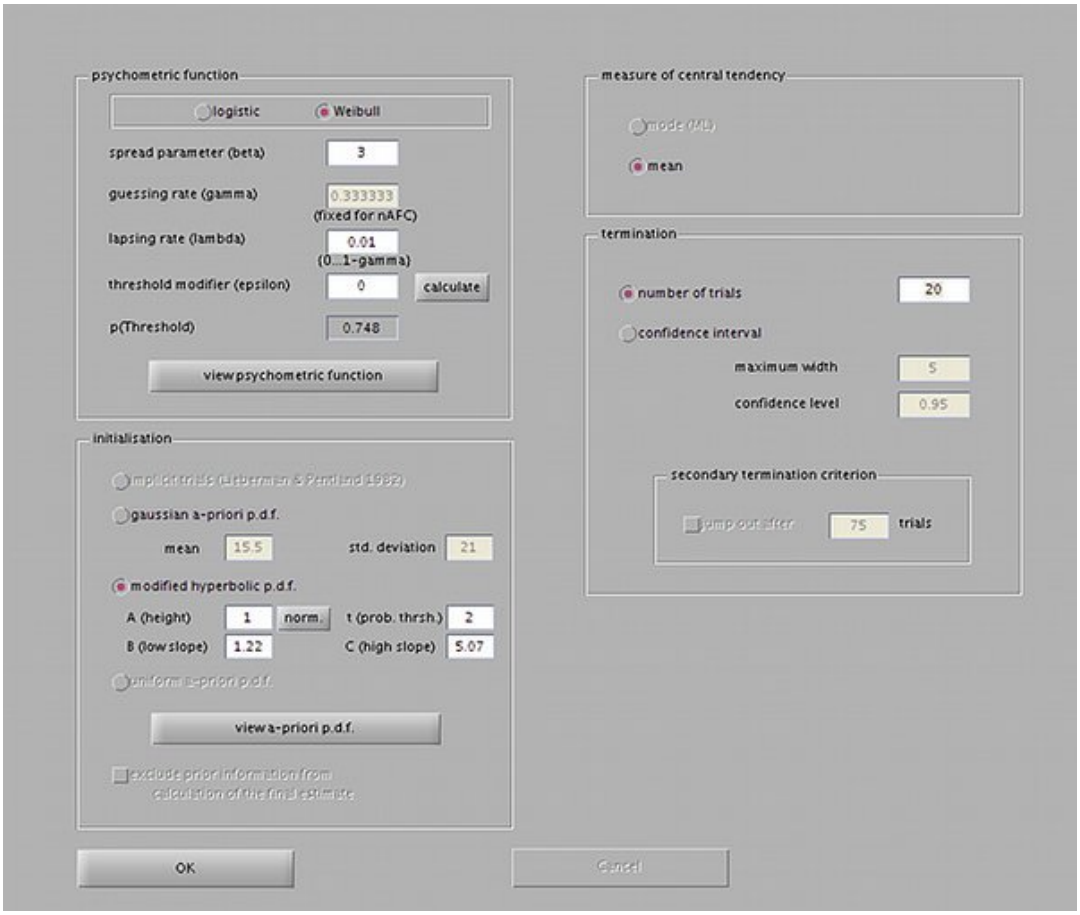


Abbildung 3.6: Neue Version von Whispers *Edit ZEST Settings* Dialog.

chung 2.5 verwendet auch die logistische Funktion, ϵ musste neu implementiert werden – dessen Eingabefeld ist bei Auswahl der logistischen Funktion deaktiviert. Auf diese Weise erklärt schon die Benutzeroberfläche die Zugehörigkeit dieses Parameters zur Weibull-Funktion. Der Pushbutton [calculate] rechts neben dem Eingabefeld für ϵ ruft bei Betätigung die neu implementierte Funktion `find_epsilon_for_ideal_sweat_factor` auf, die durch Iterierung von ϵ den *ideal sweat factor* an den Threshold der Weibull-Funktion verschiebt und das so ermittelte ϵ_{ideal} zurückgibt. Die Benutzung dieser Funktion ist optional. Ferner wurde für die psychometrische Funktion ein Informationsfenster implementiert, welches die Wahrscheinlichkeit der psychometrischen Funktion am aktuell eingestellten Threshold angibt. Diese Anzeige ist auch für die logistische Funktion aktiv und aktualisiert sich selbstständig, sollten neue Werte für β, γ, λ oder ϵ eingegeben werden. Die Plottingfunktion für die aktuelle psychometrische Funktion wurde um die Möglichkeit erweitert, auch die Weibull-Funktion mit den aktuell eingegebenen Parametern darstellen zu können.

Im Feld „Initialisation“ wurde die in Kapitel 2.3 vorgestellte Wahrscheinlichkeitsdichtefunktion implementiert und ihre Parameter gemäß Gleichung 2.4 zur Konfiguration bereitgestellt. Die Möglichkeit, die aktuell eingestellte pdf plotten lassen zu können, wurde um die Möglichkeit, auch die neue Funktion darstellen zu können, erweitert. Da hiernach auffiel, dass deren Höhe teils erheblich mit unterschiedlichen Werten für die Steigungen schwankt, wurde zusätzlich der Button [norm.] direkt rechts neben ihr implementiert. Seine Betätigung normiert das Maximum der Kurve auf 1 und gibt das entsprechende A zurück. Seine Benutzung ist optional. Im gesamten Feld „Initialisation“ wurde die der im Rahmen der Bayes-Statistik üblichen und in Kapitel 2.2 dargelegten Nutzung zuwiderlaufende Verwendung der Bezeichnung „Likelihood“ – „apriori-likelihood“ zumal

– allenthalben ersetzt durch das neutrale „p.d.f.“. Ferner selektiert die Auswahl von „ZEST“ im vorherigen Auswahldialog des Adaptionsmechanismus hier die hyperbolische Sekante als p.d.f. vor, „QUEST“ selektiert die Gaussverteilung, beide Optionen deaktivieren die übrigen Optionen. „ZEST“ deaktiviert zudem die Option, die apriori-Information von der abschließenden Ermittlung der Schwelle auszunehmen. Gemäß den Ergebnissen der Simulation bezüglich der frühen Konvergenz des Verfahrens,[8] wird die Anzahl der Trials mit „20“ vorgewählt.

Die Berechnung sowohl der Weibull-Funktion als auch der neu hinzugekommenen apriori-pdf wurden im die eigentliche Schätzung vornehmenden „.m-file“ MLBayes.m ergänzt. Die oben angeführten nun neu hinzugekommenen Parameter müssen zudem (dauerhaft) an einem bestimmten Ort gespeichert werden, dazu wurde der Teil der Datenstruktur `TSP.sections`, der auch schon bisher die MLBayes-Konfigurationsparameter aufgenommen hatte, erweitert.[9] Durch diese Erweiterung und die bei Initialisierung des Auswahldialoges erfolgenden Zugriffe auf die neuen Felder, sind alte Projekte, in deren `TSP` die neuen Felder nicht vorkommen, nicht mehr kompatibel zum Programm – der Versuch, auf die nicht vorhandenen Felder zuzugreifen, wird von Matlab mit einer Fehlermeldung abgebrochen. Aus diesem Grunde wurde eine Kompatibilitätsfunktion exklusiv für

[8] Otto, 2008

[9] Die neuen Felder sind: psyfunctyp : TSP.sections{n,4}{m,7}{1,4}{1,15} (neuer default: 1 = logistic (Wg. Kompatibilität));
epsilon : TSP.sections{n,4}{m,7}{1,4}{1,16} (neuer default: 0);
(p(T)) : TSP.sections{n,4}{m,7}{1,4}{1,17} (neuer default:);
A : TSP.sections{n,4}{m,7}{1,4}{1,18} (neuer default: 1);
B : TSP.sections{n,4}{m,7}{1,4}{1,19} (neuer default: 1.22);
t : TSP.sections{n,4}{m,7}{1,4}{1,20} (neuer default: 2);
C : TSP.sections{n,4}{m,7}{1,4}{1,21} (neuer default: 5.07).

das Laden bestehender Projekte implementiert. Wird ein Projekt geladen, wird absturzsicher auf das Vorhandensein des ersten neuen Konfigurationsfeldes getestet und bei Misserfolg die erforderlichen neuen Felder erzeugt und mit die Originalkonfiguration nicht kompromittierenden Werten aufgefüllt, der Parameter, der den Adaptionsmechanismus beschreibt, wird im Falle der nun obsoleten Auswahl „3“,“MLBayes“ in dessen aktuelle, voll kompatible Auswahl „33“,“MLBayes manuell“ umgeschrieben. Die Nutzung möglicherweise wichtiger, jedoch vor Erweiterung des Programmes angelegter Projekte ist damit weiter sichergestellt.

3.3 Optimierung der Stimulus-Konfiguration

In Kapitel 1.3.2 wurden die Unannehmlichkeiten in Verbindung mit der bei adaptiven Messverfahren häufig vorkommenden großen Menge an zu konfigurierenden Stimuli beschrieben, Abbildung 1.1 zeigte den *Stimulus pool* vor seiner Überholung, Abbildung 3.7 zeigt ihn danach; die erste Optimierung in dieser Richtung jedoch betraf bereits den *Edit stimulus*-Dialog, Abbildung 1.2: Genau wie bereits schon immer die Felder für die Dauer einer geladenen Stimulusdatei automatisch aus den eingelesenen Daten ermittelt ausgefüllt werden, wird nun bei jedem Laden eines Stimulus dessen Namensfeld automatisch mit dem Namen der Stimulusdatei abzüglich der Dateiendung ausgefüllt – vorausgesetzt, das Namensfeld weicht nicht vom Standardeintrag ab, d. h. es wurde bisher nicht vom Benutzer geändert; Selbsteingegebene Namen werden selbstverständlich respektiert und keinesfalls überschrieben.

Den Stimulusnamen aus dem Dateinamen zu generieren schien vernünftig, tragen Audiodateien doch nicht selten als Dateinamen eine Bezeichnung ihres Inhalts.

Der neue Stimulus-Pool, Abbildung 3.7, enthält neue Bedienelemente, und die

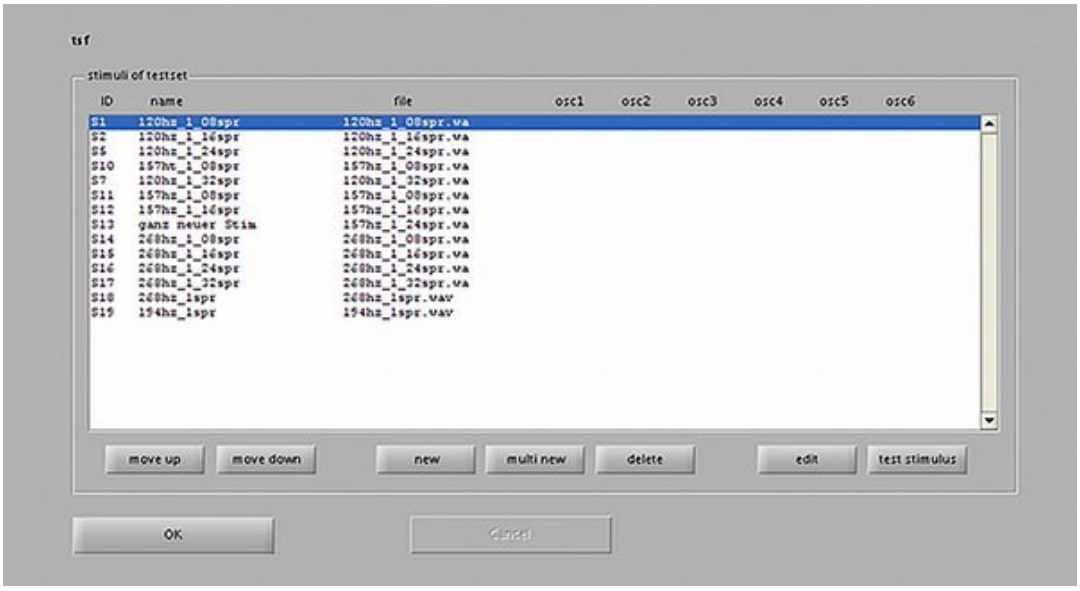


Abbildung 3.7: Whispers *Stimulus pool*, neue Version.

alten wurden teils überarbeitet.

Die links befindlichen move up und move down-Buttons sind nicht zwingend nötig – spielt die Reihenfolge der Stimuli im Pool doch tatsächlich für jedweden Hörtest gar keine Rolle, die Stimuluszuordnung erfolgt immer erst in einem weiteren manuellen Schritt. Dennoch tauchen die Stimuli in jener Reihenfolge, in der sie im Pool vorliegen, auch in den Zuordnungs- und Zuweisungsdialogen auf – die Möglichkeit, die Stimuli nach eigenem Gutdünken zu sortieren, wirkt so also zumindest indirekt auf einem möglicherweise flüssigen Arbeitsablauf. Der Sinn der Buttons ist jedoch noch ein anderer: Denn war bis jetzt nur das Löschen des jeweils letzten Stimulus des Stimulus Pools möglich und so eine bestimmte Reihenfolge eher fest vorgegeben, so erlaubt der neue delete-Button das Löschen jedes beliebigen Stimulus,[10] nicht nur vom Ende der Liste her. Diese drei Buttons erlauben nun also das einfache Löschen eines Stimulus an beliebiger Stelle im

[10] Unter der einzigen Einschränkung, dass ein einziger verbleibender Stimulus konzeptionsbedingt nicht aus dem Pool gelöscht werden kann.

Pool, das Einfügen eines neuen (geschieht immer am Ende der Liste), sowie das anschließende Bewegen an die Position des Gelöschten – mit anderen Worten: Das Austauschen von Stimuli wird ermöglicht, in der früheren Version hätte dies das Löschen aller nach dem auszutauschenden Stimulus befindlichen Stimuli und deren anschließendes Neuanlegen bedeutet.

Die weitere große Erweiterung des Stimulus-Pool stellt die mit dem multi new-Button verknüpfte Funktionalität dar. Anders als der Button new, der einen Stimulus anlegt, dessen Selektion und anschließendes Editieren dahingehend verlangt, dass ihm erst in diesem Schritt erst Audiodateien oder OSC-Befehle zugeordnet werden, ist der multi new-Button darauf ausgelegt, schnell eine große Zahl von Stimuli anzulegen und mit Audiodateien zu verknüpfen.

Bei Betätigung öffnet sich ein Dateiauswahldialogfenster. In diesem ist es nun möglich, eine Mehrfachauswahl an (wav-)Dateien zu treffen.[11] Für jede der selektierten Dateien wird daraufhin ein leerer Stimulus angelegt, die entsprechende Datei wird mit ihm verknüpft und das Namensfeld wird ausgefüllt auf Grundlage des Dateinamens (siehe oben). Ist der erste und einzige vorher im Pool vorliegende Stimulus zudem komplett unbearbeitet, d. h., hat der Benutzer bisher noch keine Veränderungen an ihm vorgenommen, so wird er mit einbezogen in die neu zu erstellenden Stimuli.

(Da für den multi new-Button zudem auch die Auswahl nur einer Audiodatei implementiert wurde, kann er als Abkürzung bezüglich des *Edit Stimulus*-Dialogs genutzt werden in Fällen, in denen OSC nicht von Interesse ist.)

Die OSC-Konfiguration wurde durch all diese Schritte noch in keiner Weise vereinfacht, hierzu kann das im nächsten Kapitel vorgestellte *Defaults'-Preset-System* beitragen.

[11] Durch Mausklicks bei gleichzeitigem Halten von Shift oder Steuerung z. B.

3.4 Standardwerte-Preset-System

Kapitel 1.3.3 beschreibt Szenarien, in denen eine direktere Konfiguration der Testverfahren wünschenswert erscheinen kann. Bei der Konzeption einer Antwort auf diesen Bedarf stellte sich jedoch schnell heraus, dass mit einer einfachen Sammlung aller Testparameter unter einer Oberfläche auf der anderen Seite Whispers Stärken flexibler und verschachtelter Versuchsverknüpfung preisgibt, lassen diese sich doch sehr schlecht quasi „statisch“ im Rahmen der gewünschten Einheitskonfigurationsoberfläche abbilden.

Entwickelt wurde darum eine andere Lösung, die auf der einen Hand sehr wohl die übersichtliche Konfiguration quasi aller Parameter eines Testverfahrens erlaubt (und mehr), andererseits aber trotzdem die Flexibilität der Hörversuchsanlage nicht unterminiert.

Um diese Aufgabe bewältigen zu können, gibt das entwickelte Konfigurationssystem dem Benutzer *nicht* die Konfigurationsmöglichkeiten notwendig einzelner „Instanzen“ von Hörversuchsdurchläufen an die Hand, stattdessen stellt es ihm den Zugriff auf die *default-Werte* aller Parameter zur Verfügung.

Abbildung 3.8 zeigt die Oberfläche des über das `edit`-Menü und den Menüpunkt `configure procedure's defaults` zu erreichenden *Defaults' Presets Systems*.

Zur Funktion:

Jedesmal, wenn man in Whisper zur Liste der in einem Hörversuch zu benutzenden ein Verfahren hinzufügt – sei es nun ein adaptives, das SD, RGT, ABX oder zukünftig erst zu implementierende – jedesmal, wenn so ein Verfahren angelegt und einer *test series* hinzugefügt wird, werden dessen später zur Konfiguration bereitstehende Parameter mit bestimmten Standardwerten aufgefüllt, welche der Autor des Verfahrens – der, der es in Whisper implementiert hat – vorher ange-

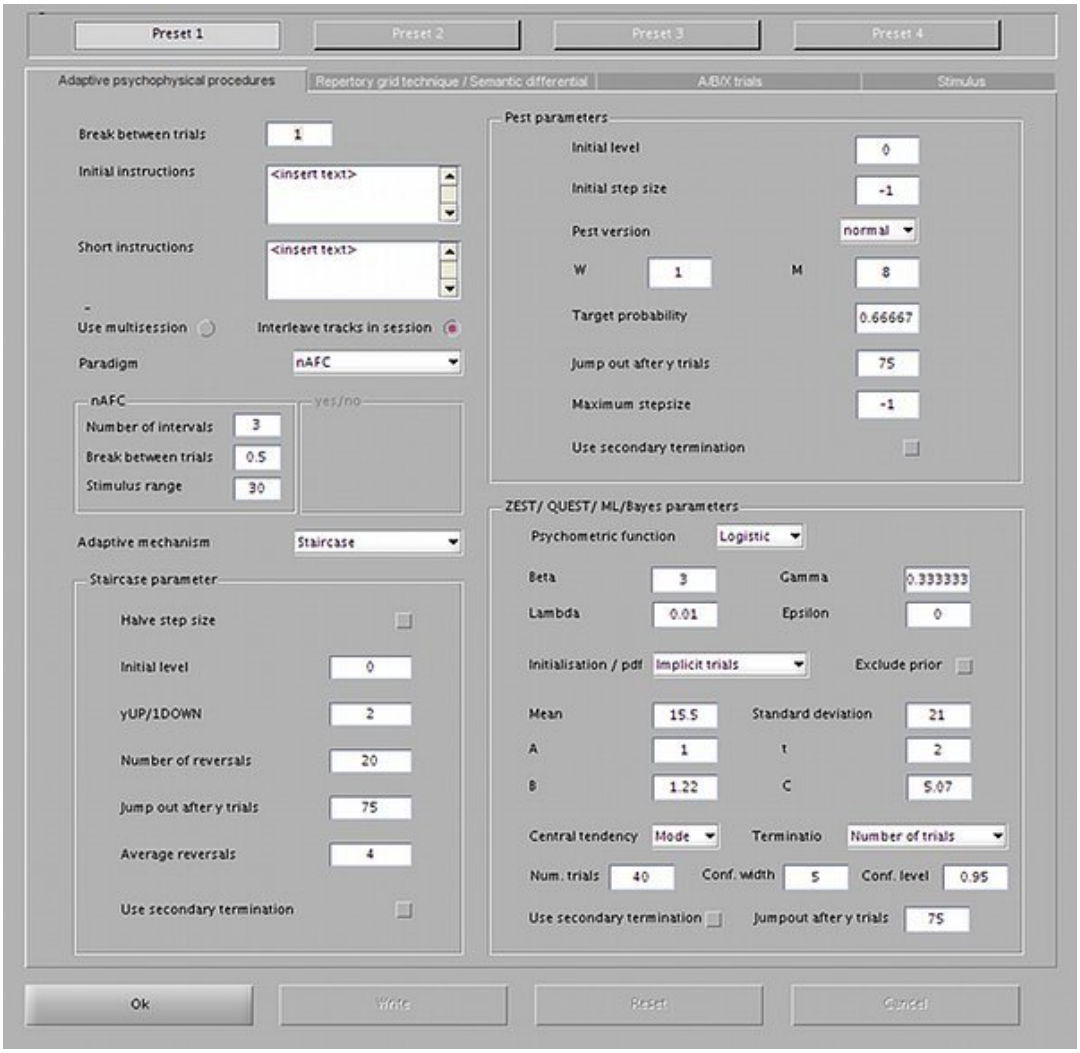


Abbildung 3.8: Whispers *Defaults' Presets Systems*.

legt hat in der Datei namens `PC.mat`.[12] Selbst *wenn* nun die Tatsache gegeben wäre, dass per Konfigurationsoberfläche alle Parameter eines Testverfahrens zur Konfiguration direkt zugänglich greifbar wären – wird ein Verfahren hinzugefügt, wird es zwingend mit den besagten Standardwerten aufgefüllt, *bevor* man es dann eventuell selbst konfigurieren kann.

Auf die `PC.mat` erfolgt in Whisper ausschließlich lesender Zugriff – bis jetzt. Denn das *Defaults' Presets Systems* gewährt Zugang zu den Initialwerten, mit denen neue Verfahren überhaupt erst angelegt werden.

Im Folgenden sollen knapp die Funktionsweise dieses Systems sowie Nutzungstechniken und Eigenheiten erläutert werden.

Wie in Abbildung 3.8 zu erkennen ist, besteht das Konfigurationswerkzeug aus einer graphischen Oberfläche mit (zur Zeit) vier *Tabs* – eines für die Parameter der adaptiven Prozeduren, eins, was gemeinsam die Parameter von SD und RGT zur Konfiguration bietet, eines für ABX-Trials und eines unter der Überschrift „Stimulus“. Diese vier Tabs bieten alle Parameter dieser Prozeduren (plus Konfigurationsmöglichkeiten für Stimuli-Defaults, dazu später mehr), zumindest all jene, für die es überhaupt sinnvoll ist, sie vor dem konkreten Anlegen eines Testverfahrens überhaupt festzulegen. Zuordnungen bestimmter Stimuli zu Stimulusintensitäten beispielsweise lassen sich hier nicht treffen. (Dies ist jedoch keine Einschränkung – bedenkt man, dass das Treffen einer solchen Zuordnung bedeuten würde, dass jedes neu angelegte Verfahren eben schon diese bestimmten Stimuli mit ihrer Zuordnung enthalten würde, so erkennt der Whispererfahrene möglicherweise leicht, dass ein solches Arrangement auch heute schon Bestandteil von Whisper ist und ganz leicht realisiert werden kann: Anlegen einer neuen *test series*, Hinzufügen, Konfigurieren und Zuordnen von Stimuli, Abspeichern. Kopiert man die resultierende `TSP.mat` und die Stimulusdateien an einen sicheren

[12] „*P*rogram *C*omponents“.

Ort, kann man deren Kopien als genau das oben vermeintlich auch von der neuen Konfigurationslösung nicht Gebotene immer wieder unverändert neu nutzen.)

Konfigurieren lassen sich jedoch außer mit konkreten Stimuli zusammenhängenden sonst fast alle Parameter, z. B. auch der parameterreichen adaptiven Prozeduren. Sind die Einstellungen hierzu einmal getroffen und gespeichert (per `ok` oder `write`-Button), sind sie sofort verfügbar. D. h. z. B., dass sich alle Parameter des ZEST-Verfahrens bzgl. psychometrischer Funktion, initialer pdf, Anzahl der Trials etc. komfortabel übersichtlich vorab einstellen lassen und ein neu der aktuellen *test series* hinzugefügtes AP-Verfahren mit ZEST-Adaptionsmechanismus automatisch schon die „richtigen“ Einstellungen enthält – Stimuli anlegen und zuordnen und schon ist man zum Hörversuch bereit.

Die Einsatzmöglichkeiten sind aber noch komplexer. Am oberen Rand lassen sich vier Buttons mit der Aufschrift „Preset 1“ bis „Preset 4“ erkennen. Im *Defaults' Presets Systems* entspricht nun ein Preset einem kompletten Satz an Vorgabeeinstellungen, alle Standardparameter aller Testverfahren (also alle Einstellmöglichkeiten aller Tabs) sind darin eingeschlossen. Beim Drücken eines anderen als des aktuellen Presets werden automatisch alle Einstellungen des gerade verlassenen Presets abgespeichert[13] und die Einstellungen des neuen Presets sind sofort im Arbeitsspeicher vorhanden und werden bei Hinzufügen eines Testverfahrens zu dessen Initialisierung genutzt.

Da das *Defaults' Presets Systems* mit Absicht so programmiert wurde, dass sein GUI-Fenster a) geöffnet bleiben kann, während andere Matlab-Whisper-GUI-Fenster aktiv sind und b) es beim Öffnen nicht das Schließen oder Verschwinden der anderen Fenster erzwingt, eröffnet sich die Möglichkeit, während

[13] Gespeichert wird je nach Presetnummer nach PC.1.mat bis PC.4.mat – das Original mit dem Standardwerten der Autoren wird nicht angerührt.

des Hinzufügens beispielsweise einer AP (natürlich automatisch mit den vorher festgelegten Initialwerten) das Presets-Fenster geöffnet zu lassen, nach dem gerade erfolgten Anlegen einer AP das Preset zu wechseln und nun beim Anlegen einer weiteren AP eine komplett andere Grundkonfiguration, ganz den eigenen Wünschen entsprechend, vorliegen zu haben.

Das *Defaults' Presets Systems* bietet in der jetzigen Inkarnation vier solcher Presets, die Auswahl eines bestimmten Presets bleibt über das Programmende hinaus bis zum nächsten Programmstart erhalten, Presets lassen sich leicht per `Reset`-Button wieder in den Urzustand der von den Implementatoren der Verfahren vorgesehenen Werte zurücksetzen,[14] der Zustand des `Reset`-Button und des `Write`-Button helfen zudem dabei, den Überblick über den Status des aktuellen Presets zu behalten: Ist der `Write`-Button aktiviert, liegen momentan noch ungespeicherte Änderungen vor, ist der `Reset`-Button aktiviert, entspricht das aktuelle Preset in mindestens einer Einstellungen nicht mehr dem bis jetzt unveränderlichen, ursprünglichen Default-Zustand.

Eine Erweiterung sogar über die Konfigurationsmöglichkeiten, die durch die „normalen“ Standardwerte geboten werden, zeigt der bis jetzt noch nicht besprochene Tab *Stimulus* auf, siehe Abbildung 3.9: Mit den hier vorgegebenen Feldern lassen sich nicht einfach ein Stimulus, sondern jeder neuangelegte Stimulus vorkonfigurieren. Besonders nützlich dürfte dies sicher bei OSC-Nutzung von Stimuli sein – variieren beispielsweise OSC-Pfade zwischen verschiedenen Stimuli eines Testvorgangs sicherich nur gering (wenn überhaupt), so können diese Felder an dieser Stelle vorausgefüllt werden, in der Konfiguration der tatsächlichen, ja leicht variierenden Stimuli bzw. OSC-Daten muss dann nur noch der jeweils

[14] Achtung! Konsistenzbedingt speichert eine Reset-Aktion die zurückgesetzten Parameter sofort in der ihr zugehörigen Presetdatei, eine Rücknahme dieser Aktion ist nicht möglich. Und wie erwähnt speichert auch der Druck auf einen anderen als den aktuellen Presetbutton das verlassene Preset sofort.

variierende Teil der Konfiguration nachgetragen werden.

Das *Defaults' Presets Systems* ist zudem fehlertolerant programmiert, was das Vorhandensein oder Fehlen von Presetspeicherdateien anbelangt – dies gilt selbst für aktuell als aktiv markierte. Fehlen Presetspeicherdateien, werden sie als Klon der ursprünglichen `PC.mat` neu angelegt.

Reichen vier Presets nicht aus, so können entweder Presetspeicherdateien an anderem Orte gelagert und bei Bedarf wiederverwendet werden, oder man erhöht die Zahl der Presets und Presetbuttons – bei der Implementierung wurde versucht darauf zu achten, die Beschränkung auf vier Presets nirgends im Programmcode „hart“ einzuarbeiten – was hier vier sind, kann als Variable mit relativ geringem Aufwand[15] im Prinzip beliebig erhöht werden.

3.5 Linux-Portierung

Die Linux-Tauglichkeit von Whisper hängt prinzipiell an drei Punkten:

1. Dateisystemzugriffe,
2. Zugriffe auf die Audiohardware,
3. OSC-Fähigkeit

Zu 1. Unixartige Betriebssysteme wie Linux verwenden als Trennzeichen in Dateisystempfaden den *forward slash* „/“, Microsoft Windows benutzt den *back slash*.

Da in Whispers Quellcode an vielen Stellen im Zusammenhang mit *allen* Dateioperationen (Öffnen, Speichern etc.) der Windowsbackslash verbatim im Quellcode Verwendung fand, waren alle diese Funktionen von Whisper unter Linux zum Scheitern verurteilt.

[15] Anpassung von Schliefenzählern z. B.

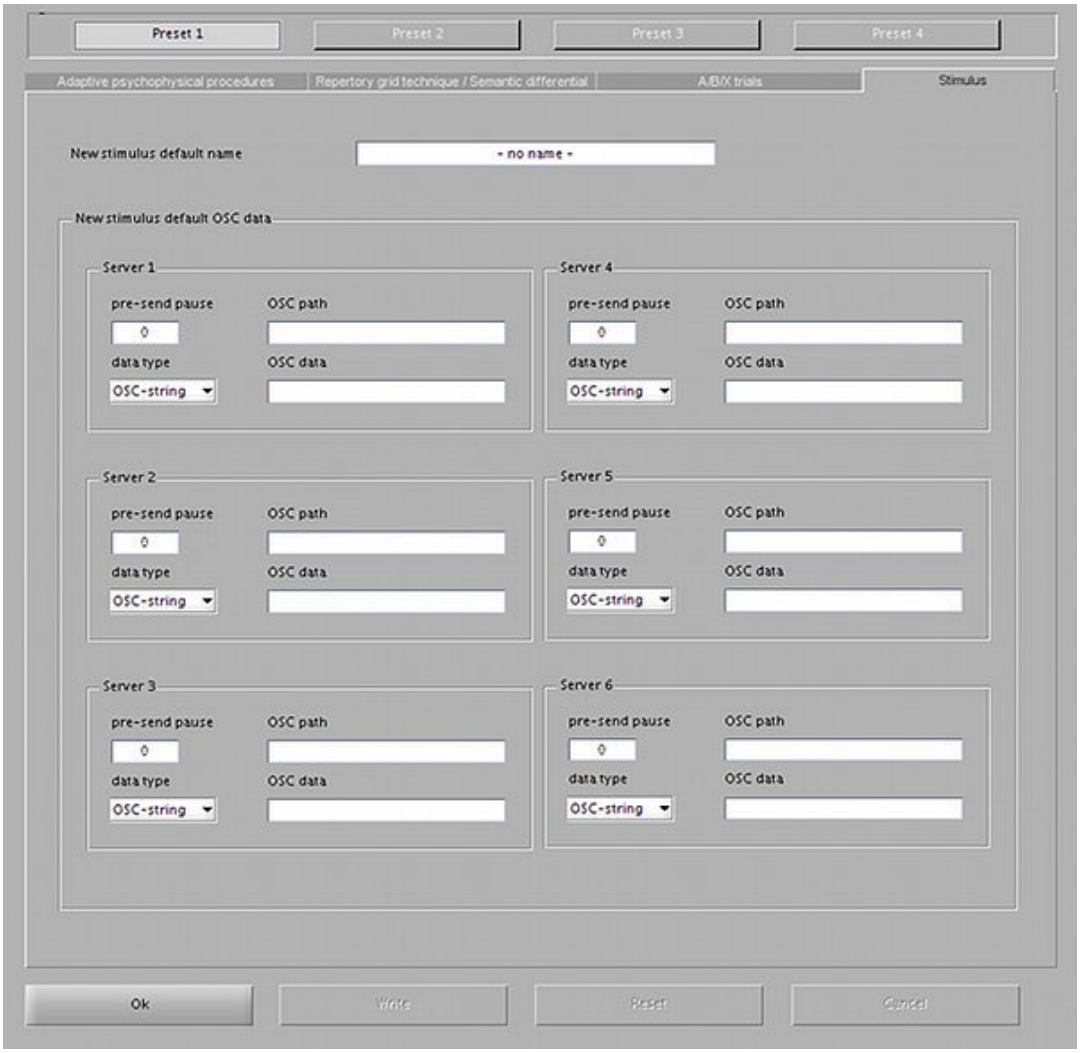


Abbildung 3.9: Whispers *Defaults' Presets Systems*, Tab „Stimulus“.

Abhilfe war erfreulicherweise jedoch relativ einfach: Matlab stellt eine Funktion zur bereit, die je nach Betriebssystem den betriebssystemspezifischen Trenner zurückgibt, sie lautet `filesep`.

Nach Ersetzen aller Vorkommen des Backslash also durch geeignete den Trenner `filesep` beinhaltende Konstruktionen, waren nur noch solche Dateioperationen gewissenhaft zu prüfen, die beispielsweise per Befehl einen ganzen Dateipfad in eine Variable laden – der Quellcode bekommt in solchen Fällen sozusagen den Trenner gar nicht zu Gesicht. Und tatsächlich gibt es solche Vorkommen – abermals glücklicherweise stellten sie sich aber als unproblematisch heraus: Denn den betreffenden betriebssystemspezifischen Trenner benutzen sie zwar zur Laufzeit des Programms – da stellt er aber tatsächlich gar kein Problem dar, denn er wird erst unter dem dann aktuellen Betreibssystem erzeugt. Gespeichert werden von Whisper aber keine Pfade, sondern nur Dateinamen. Diese sind in dieser Hinsicht gänzlich unproblematisch.

Zu 2. Reibungslos funktionierender Zugriff auf die Audiohardware unter Linux durch Matlab ist keine Selbstverständlichkeit – der `sound()`–Befehlt beispielsweise setzt unter Linux das Selbstkompilierten eines *.mexglx-Objektes voraus.

Das von Whisper benutzte `audioplayer`-Objekt legt diesbezüglich weitaus geringere Empfindlichkeit an den Tag. Dennoch verursacht es Probleme: Der *scope* des Objekts, d. h. der Rahmen, innerhalb der es existiert und auf Ressourcen zugreifen kann, ist lokal begrenzt, so dass die früheren Whisper-Autoren den `audioplayer` als globale Variable deklarierten, um überhaupt etwas zu hören. Das Problem hierbei ist, dass sie die Funktion zum Abspielen einer Testdatei aufriefen, den `audioplayer` dort global deklarierten, den `audioplayer` das Audiofile abspielen ließen, während im Hauptprogramm eine Pause der Länge der Audiodatei entsprechend mit Warten auf deren Abspielende verbracht wurde, um

danach dann die den `audioplayer` enthaltende Funktion um das Abspielen des nächsten Stimulus zu bitten. Soweit die Theorie. Praktisch ist es aber so, dass der verwendete Abspielbefehl `play` nicht blockierend ist, d. h. die Funktion, die ihn enthält, einfach weiter durchläuft, ohne darauf zu warten, dass er das Abspeilen abgeschlossen hat. Vor allem aber ist sie potentiell schon durchgelaufen, wenn der `play`-Befehl noch gar nicht mit dem Abspielen begonnen hat. In diesem Fall fängt der Zähler für die vermeintlich während des Abspielens zu wartende Pause *zu früh* an, hört zu früh auf, es wird um das Abspielen der nächsten Datei gebeten, während noch der vorige `play`-Befehl die Audiohardware blockiert. Und genau hier bricht das Programm in einem solchen Fall ab.

Da der `audioplayer` grundsätzlich unter Linux funktioniert, kann die Lösung des Problems z. B. darin bestehen, statt dem nicht-blockierenden `play` den die weitere Ausführung der ihn enthaltenden Funktion bis zum Ende des Abspielvorgangs sehr wohl blockierende `playblocking` zu verwenden – die wartende Pause im Hauptteil des Programms würde dann einfach entfallen.

Entfallen würde allerdings auch die den ursprünglichen Autoren wohl wichtige Funktionalität, angefangenes Abspielen unterbrechen zu können (beispielsweise beim Auditieren von Stimuli im Stimulus Pool). Eine mögliche Lösung hierzu wäre das Verwenden zweier verschiedener Abspielfunktionen – unterbrechbar fürs Auditieren, ununterbrechbar während des Hörtests.

All diese Änderungen sind nicht besonders aufwendig oder schwierig – da sie aber doch deutlich abseits bloßer Kompatibilitätsbemühungen in bestimmte bestehende Strukturen von Whisper eingreifen, soll hier zwar eine Antwort auf das Problem zu lesen sein, die Entscheidung damit umzugehen aber den Programmbetreuern gebühren.

Zu 3. Whisper verwendet eine Implementation von OSC für Matlab von Andy Schmeder, deren Windows-Version als kompilierte „.dll“-Dateien verfügbar sind

und deren als Quellcode verfügbare Variante die Kompilierung durch den Matlab-Compiler „mex“ mit einer Fehlermeldung abbricht.

Dieses Problem konnte gelöst werden: Verursacht wird das Problem durch zu strikte Compiler-Optionen, die mit der wirklich weitverbreiteten Verwendung des *inline*-Konstrukts in der *osc-mex* zugrundeliegenden *liblo* kollidieren. Zur Lösung ist zunächst der Matlab-Compiler mit mit der „-setup“ -Option aufzurufen, „mex -setup“, daraufhin eines (egal, welches) der angebotenen „mex-option-files“ auszuwählen, sich dessen Speicherort zu merken und anschließend dahingehend zu editieren, dass alle „-ANSI“ oder „-ansi“ -Optionen aus diese Datei gelöscht werden. Daraufhin läuft die Kompilierung von *osc-mex* glatt durch und liefert vier „.mexglx“-Dateien, die in den Programmordern von Whisper zu kopieren sind. Matlab verfügt damit voll über die gewünschte OSC-Funktionalität.

Unter dem Vorbehalt des Zögerns bezüglich Punkt zwei steht einer gleichwertigen Nutzung von Whisper auch unter Linux nichts mehr entgegen.

Alle Erweiterungen von Whisper innerhalb dieser Arbeit sind unter Debian/Linux mit der Linux-Variante von Matlab 7.9.0 erstellt worden.

4 Abschlussdiskussion und Selbstkritik

Eingebettet in ihren historischen Entstehungzusammenhang, ihren aktuellen Platz in den empirischen Wissenschaften, in den aus den spezifischen Bedürfnissen und Anforderungen erwachsenden Arbeitsauftrag dieser Arbeit, wurde ein adaptives psychometrisches Verfahren zur Schwellwertbestimmung ZEST vorgestellt, dessen Grundlagen expliziert, dessen Detailaspekte dargestellt, in der bestehenden am Fachbereich Audiokommunikation entwickelten Hörversuchsumgebung Whisper wurde seine Implementierung vorgenommen und es als Verfahren dem Benutzer zugängig gemacht. Die Implementierung erfolgte nicht ohne auch Detailaspekte zu bedenken und den Gebrauchsnutzen der Software so möglichst überall zu erhöhen.

Whisper wurde außerdem ergänzt um effizientere Mittel, mit der Konfiguration großer Mengen von Stimuli umgehen zu können, sowie mit einem Werkzeug zur souveränen Handhabung der sonst weitverzweigt und verschachtelt verteilten Konfiguration der Testprozeduren. Bei allen Erweiterungen und Ergänzungen wurde auf bleibende Kompatibilität in Berührungspunkten mit früheren Versionen der Software großer Wert gelegt.

Zur produktiven Arbeit und Weiterentwicklung an Whisper war die beträchtliche Hürde der Einarbeitung in die Software zu nehmen. Diese Hürde beruht nur bedingt auf der durchaus gegebenen Komplexität einer mit solch nicht gerade geringem Funktionsumfang ausgestatteten Software wie Whisper.

Als andere Ursachen geraten ins Blickfeld:

- Verbesserungsfähige technische Dokumentation.
- Lesbarkeit des Quellcode.

Der mit dieser Arbeit verbundene Lernprozess ließ eine Wertschätzung entstehen für zu ihrer effektiven Nutzung befähigende Werkzeuge.

Explikation der Kritikpunkte: Die Nutzung einiger wichtiger Variablen, die in Whisper zudem beträchtlich tief verschachtelt sind, ist nicht dokumentiert. Auch die vorliegende Arbeit ändert daran allerdings eher nicht viel. Programmieridiome, Strategien, Code-Funktionalitäts-Zusammenhänge sind so gut wie nicht dokumentiert, hier setzt die vorliegende Arbeit durchaus wenigstens etwas an und bietet ganz bestimmt Hilfe bei einem Einstieg.

Da sich alle Information zur produktiven Erweiterung der Software also fast nur durch Studium des Quellcodes erlangen lässt, fällt dessen Lesbarkeit in mehrfacher Hinsicht auf. Auf der untersten Ebene wäre eine Formatierungskonvention für den Quellcode wirklich wünschenswert. Die Programmierung in Rahmen der vorliegenden Arbeit machte einige Vorstöße in diese Richtung, siehe Abbildung 4.1. Der Unterschied ist deutlich, der Nutzen klar: Größere Übersichtlichkeit, kleinere Fehlerwahrscheinlichkeit, beschleunigte Fehlersuche, ermüdungsfreieres Lesen. Unter diesen Bedingungen produziert man auch einfach weniger Fehler. Der Wunsch nach Lesbarkeit setzt sich auch auf semantischer Ebene fort: Ausdrücke wie `if isempty(PP.edit_section_line) == 0` müssen erst entschlüsselt werden, der semantisch äquivalente Ausdruck `if ~isempty(PP.edit_section_line)` liest sich dagegen fast wie ein richtiger Satz.

Die Selbstkritik ist jedoch auch nicht stumm und muss feststellen, dass die zur Verfügung stehenden Mittel und Werkzeuge in ihren Stärken hätten mehr zum Vorteil gebraucht werden können, im ganz Speziellen wäre es im Rückblick ein Leichtes gewesen, vor allem das für die Quellcodeverwaltung genutzte Content

```
173   %termination
174 - if TSP.sections{PP.edit_section_line,4}{PP.edit_track_line,7}{1,4}{1,9}==2 %Voreinstellung der GUI ist Auswahl 1 (number of
175 -     set(handles.rad_confidence_interval,'Value', 1);
176 -     set(handles.ed_number_of_trials,'Enable', 'Off');
177 -     set(handles.ed_confidence_interval,'Enable', 'On');
178 -     set(handles.ed_confidence_level,'Enable', 'On');
179 -     set(handles.cb_secondary_termination,'Enable', 'on');
180 -     if TSP.sections{PP.edit_section_line,4}{PP.edit_track_line,7}{1,4}{1,13}==1
181 -        set(handles.ed_secondary_termination,'Enable', 'on');
182 -     end
183 - end
184 - set(handles.ed_number_of_trials, 'String', TSP.sections{PP.edit_section_line,4}{PP.edit_track_line,7}{1,4}{1,10});
185 - set(handles.ed_confidence_interval, 'String', TSP.sections{PP.edit_section_line,4}{PP.edit_track_line,7}{1,4}{1,11});
186 - set(handles.ed_confidence_level, 'String', TSP.sections{PP.edit_section_line,4}{PP.edit_track_line,7}{1,4}{1,12});
187
188   %secondary termination
189 - set(handles.cb_secondary_termination, 'Value', TSP.sections{PP.edit_section_line,4}{PP.edit_track_line,7}{1,4}{1,13});
190 - set(handles.ed_secondary_termination, 'String', TSP.sections{PP.edit_section_line,4}{PP.edit_track_line,7}{1,4}{1,14});
191
192   % pThreshold anzeigen
193 - update_pThresholdvalue(handles)
194
195
196   % --- Outputs from this function are returned to the command line.
```

```
1392  function fill_all_fields_with_saved_data(handles)
1393
1394 - global PC
1395
1396 - set(handles.ed_brk_btwn_trials,        'String', PC.testing_procedures{1,3}{1,1});
1397 - set(handles.ed_initial_instructions,   'String', PC.testing_procedures{1,3}{1,2});
1398 - set(handles.ed_short_instructions,     'String', PC.testing_procedures{1,3}{1,3});
1399 -     use_multisession = PC.testing_procedures{1,3}{1,4}{1,1} - 1; % da 2=ja, 1=nein.
1400 - set(handles.rad_use_multisession,      'Value',  use_multisession);
1401 - set(handles.rad_interleave,            'Value',  ~use_multisession);
1402 - set(handles.pop_paradigm_selection,    'Value',  PC.testing_procedures{1,3}{1,5});
1403 -     adaptive_mechanisms = [1 2 31 32 33];
1404 - set(handles.pop_adaptive_mechanism,    'Value',  find(adaptive_mechanisms==PC.testing_procedures{1,3}{1,7}{2,1}));
1405 - set(handles.ed_nAFC_num_intervals,     'String', PC.testing_procedures{1,3}{1,8}{1,1});
1406 - set(handles.ed_nAFC_brk_btwn_trials,   'String', PC.testing_procedures{1,3}{1,8}{1,2});
1407 - set(handles.ed_nAFC_range,             'String', PC.testing_procedures{1,3}{1,7}{1,1}{1,1});
1408 - set(handles.pop_psyfun,                'Value',  PC.testing_procedures{1,3}{1,7}{1,4}{1,15});
1409 - set(handles.ed_beta,                   'String', PC.testing_procedures{1,3}{1,7}{1,4}{1,1});
1410 - set(handles.ed_gamma,                  'String', PC.testing_procedures{1,3}{1,7}{1,4}{1,2});
1411 - set(handles.ed_lambda,                 'String', PC.testing_procedures{1,3}{1,7}{1,4}{1,3});
1412 - set(handles.ed_epsilon,                'String', PC.testing_procedures{1,3}{1,7}{1,4}{1,16});
1413 -     pdf_types = [1 2 4 3];   % muss man hier so machen, da die mod.hyp.pdf auch im GUI an 3. Stelle kommt, aus Kompatib
1414 - set(handles.pop_pdftyp,                'Value',  find(pdf_types==PC.testing_procedures{1,3}{1,7}{1,4}{1,5}));
1415 - set(handles.cb_exclude_prior,          'Value',  PC.testing_procedures{1,3}{1,7}{1,4}{1,4});
```

Abbildung 4.1: Auswahl von Whispers Quellcode. Oben unformatiert, unten formatiert.

Management Systems *svn* als implizites Mittel zur Dokumentation des sich in Teilen evolutionär darstellenden Entwicklungsprozesses voll auszunutzen; Kleine, häufige Beiträge mit kurzen prägnanten Änderungskommentaren wären hier sehr hilfreich gewesen. Doch ein Blick auf die ebenso fehlenden Kommentare aller vorherigen Entwickler zeugt hier von einer allgemeinen Brache.

Insgesamt hat Whisper trotzdem noch immer enormes Entwicklungspotenzial – der Quellcode bedarf ein wenig Pflege, als Erweiterung bietet sich unmittelbar natürlich die Realisierung des „Ja/Nein“-Paradigmas an, das ja schon an vielen Stellen als zumindest ursprünglich einmal vorgesehen deutlich ins Auge fällt – und dem schon die ZEST-Autoren eine durchweg höhere Effizienz als den nAFC-Paradigmen bescheinigen.

Anhang

Literatur

Bech, Søren und Nick Zacharov (2006). *Perceptual audio evaluation*. Chichester: John Wiley & Sons.

Ciba, Simon (2008). *Erstellung einer Softwarebibliothek für Hörversuche – Programmkonzept und zu implementierende Testverfahren*. Magisterarbeit, FG Audiokommunikation.

Fechner, Gustav Theodor (1860). *Elemente der Psychophysik*. Leipzig: Breitkopf und Härtel.

Gundlach, Horst (1993). *Entstehung und Gegenstand der Psychophysik*. Berlin: Springer.

King-Smith, P. E. u. a. (Apr. 1994). „Efficient and unbiased modifications of the QUEST threshold method: theory, simulations, experimental evaluation and practical implementation.“ In: *Vision Res* 34.7, S. 885–912. ISSN: 0042-6989. URL: `http://view.ncbi.nlm.nih.gov/pubmed/8160402`.

Koch, Karl-Rudolf (2000). *Einführung in die Bayes-Statistik*. Berlin: Springer.

Maempel, Hans-Joachim (2006). *Kommunikationstechnik-Labor: Skript Perzeptive Messung und Evaluation*. Studiengang Kommunikationswissenschaft, Technische Universität Berlin.

Otto, Stefanie (2008). *Vergleichende Simulation adaptiver, psychometrischer Verfahren zur Schätzung von Wahrnehmungsschwellen*. Magisterarbeit, FG Audiokommunikation.

Taylor, M. M. (1971). „On the Efficiency of Psychophysical Measurement“. In: *Journal of the Acoustical Society of America* 49.2, S. 505–508.

Watson, A. B. und D. G. Pelli (Feb. 1983). „QUEST: a Bayesian adaptive psychometric method.“ In: *Percept Psychophys* 33.2, S. 113–120. ISSN: 0031-5117. URL: http://view.ncbi.nlm.nih.gov/pubmed/6844102.

Weibull, Waloddi (Sep. 1951). „A statistical distribution function of wide applicability.“ In: *ASME Journal of Applied Mechanics*, S. 293–297.

Wlodarski, André (2008a). *Erstellung einer Softwarebibliothek für Hörversuche – Implementierung und Strukturierung des Quellcodes*. Magisterarbeit, FG Audiokommunikation.

— (2008b). *Technische Dokumentation zur Hörversuchssoftware Whisper*. Teil der Whisper Hörversuchsumgebung, Fachbereich Audiokommunikation, Technische Universität Berlin.

Abbildungsverzeichnis

1.1 Screenshot „Stimulus Pool“ des Programms Whisper (alte Version). 12
1.2 Screenshot „edit stimulus“ des Programms Whisper. 13

2.1 Schwellwerthistogramme, angelegte Funktionen. Aus King-Smith u. a., 1994, S. 889. 25
2.2 Weibull-Funktion. Aus Watson und Pelli, 1983, S. 116. 27

3.1 Screenshot Matlab-GUI-Editor GUIDE. 33
3.2 Screenshot Matlab GUIDE Property Inspector. 35
3.3 Screenshot Matlab function-Deklaration im Editor. 37
3.4 Screenshot Whisper AP Edit Track Settings Dialog (alt und neu). 42
3.5 Screenshot Whisper AP Edit MLBayes Settings Dialog (alt). . . . 44
3.6 Screenshot Whisper AP Edit ZEST Settings Dialog (neu). 45
3.7 Screenshot Whisper Stimulus pool (neu). 49
3.8 Screenshot Whisper Defaults' Presets System. 52
3.9 Screenshot Whisper Defaults' Presets System, Tab „Stimulus“. . . 57

4.1 Screenshot Quellcode Whisper, Formatierung 63

Der Matlab-Quellcode kann hier nicht eingefügt werden – nur schon die von mir eingefügten Zeilen umfassen mehr als vierzig Seiten. Der Matlab-Quellcode liegt auf der beiliegenden CD vor – einmal komplett im „Whisper_testing_1.06“-Verzeichnis im Quellcode-Ordner, zweimal im sogenannten „diff“-Format: Dieses Programm erzeugt aus dem Vergleich zweier Dateien einen Bericht, in dem (im vorliegenden Fall) die in der älteren Datei vorkommenden Zeilen, in der neueren aber nicht mehr vorkommenden mit einem führenden < und jene Zeilen, die in der neuen Datei enthalten sind, in der alten aber nicht (also die von mir hinzugefügten) mit einem führenden >. Die Vergleichsdateien waren die der Whisper-Version testing_1.05_withABX.

Die Datei `Hinzufuegungen.txt` listet alle von mir hinzugefügten Zeilen, die Dateien im Verzeichnis „Diff“ im Quellcode-Ordner enthalten pro veränderter Datei sowohl von mir hinzugefügte als auch von mir entfernte oder ersetzte Zeilen.

Der Ordner „Neu“ im Quellcode-Verzeichnis der CD schließlich enthält die von mir komplett geschriebenen, neu zu Whisper hinzugekommenen zwei Matlab “.m“-Dateien.