

1. ÜBER DIESES DOKUMENT

Dieses Dokument ist *work in progress* und dient vorerst nur der Sammlung diverser (aus dem Quellcode von bwview extrahierter) Formeln.

2. ANALYSIS.C

Die Datei *analysis.c* enthält die Funktionen zur Analyse der EEG-Daten.

2.1. Vorbemerkung: Analysis types. Das Tool bietet drei verschiedene Analyse-Typen:

- 0 : Blackman-Fenster (default)
- 1 : IIR Biquad Filter mit $Q = 0,5$
- 2 : IIR Biquad Filter mit $Q = 0,72$

Im folgenden wird vorerst nur die Analyse mit Analyse-Typ 0 (Blackman-Fenster) betrachtet, da wir in der Anwendung bisher auch nur diese genutzt haben.

2.2. struct BWSetup. Ein Struct mit den grundlegenden Ausführungsparametern:

2.2.1. Definition. Der Struct ist folgendermaßen definiert:

```
91 struct BWSetup {
92     int typ;           // Analyse-Typ (0-2, s.o.)
93     int off;           // Offset im input file (in samples, gezaehlt ab 0)
94     int chan;          // anzuzeigender Kanal (gezaehlt ab 0)
95     int tbase;         // Time-base (Samples pro Datenpunkt (horizontal))
96     int sx;            // Zahl der zu berechnenden Spalten (size-X)
97     int sy;            // Zahl der zu berechnenden Zeilen (size-Y)
98     double freq0, freq1; // oberste und unterste Frequenz (Achtung: s.u.)
99     double wwrat;      // Verhaeltnis von Fensterbreite zur Wellenlaenge
                        // der Mittenfrequenz
100 };
```

2.2.2. *Zeilen- und Spaltenzahl.* Die Werte für sx und sy entsprechen der Anzahl aktuell darstellbarer Pixel (verstellbar durch Skalierung des GUI-Fensters).

2.2.3. *Oberste und unterste Frequenz.* Auf die Zeilen (Anzahl: sy) werden äquidistante Frequenzbänder (*noch zu übersetzen: in log-freq-space*) zwischen $freq0$ und $freq1$ verteilt. Das bedeutet, dass das oberste Band knapp unter $freq0$ und das unterste Band knapp über $freq1$ liegt. Um also z.B. sechs Bänder zwischen 128 Hz und 256 Hz zu erhalten, setzt man $BWSetup.sy = 6$, $BWSetup.freq0 = 256$, $BWSetup.freq1 = 128$.

2.3. **struct BWAnal.** Ein Struct mit detaillierten Parametern für die Analyse sowie den entsprechenden Daten-Arrays:

2.3.1. *Definition.* Der Struct ist folgendermaßen definiert:

```

109 struct BWAnal {
110     BWFile *file;
111     BWBlock **blk;    // Liste der geladenen Bloecke
112     int n_blk;        // Anzahl Bloecke in der Liste
113     int bsiz;         // Blockgroesse
114     int bnum;         // Nummer des vordersten Blocks in der Liste
115     int half;         // Benoetigt dieser Filter nur die linke Haelfte der
                        // Daten ? 0 Nein, 1 Ja
116
117     fftw_plan *plan;  // Big list of FFTW plans (see note below for
                        // ordering)
118     int m_plan;       // Maximum plans (i.e. allocated size of plan[])
119
120     int inp_siz;       // Size of data in inp[], or 0 if not valid
121     fftw_real *inp;    // FFT'd input data (half-complex)
122     fftw_real *wav;    // FFT'd wavelet (real)
123     fftw_real *tmp;    // General workspace (complex), also used by IIR
124     fftw_real *out;    // Output (complex)
125
126     // "Oeffentlich" lesbare, nicht veraenderliche Informationen:

```

```

127     int n_chan;          // Anzahl Kanäle in Input-Datei
128     double rate;         // Abtastrate
129
130     // "Oeffentlich" lesbare, sich veraendernde Informationen:
131     BWSetup c;           // Aktuelles Setup
132     float *sig;          // Signal mid-point values: sig[x], or NAN for sync
                           // errors
133     float *sig0;         // Signal minimum values: sig0[x], or NAN for sync
                           // errors
134     float *sig1;         // Signal maximum values: sig1[x], or NAN for sync
                           // errors
135     float *mag;          // Magnitude information: mag[x+y*sx]
136     float *est;          // Estimated nearby peak frequencies: est[x+y*sx],
                           // or NAN if can't calc
137     float *freq;         // Centre-frequency of each line (Hz): freq[y]
138     float *wwid;         // Logical width of window in samples: wwid[y]
139     int *awwid;          // Actual width of window, taking account of IIR
                           // tail: awwid[y]
140     int *fftp;           // FFT plan to use (index into ->plan[], fftp[y
                           // ]%3==0)
141     double *iir;         // IIR filter coefficients: iir[y*3], iir[y*3+1],
                           // iir[y*3+2]
142     int yy;              // Number of lines currently correctly calculated in
                           // arrays
143     int sig_wind;        // Are the ->sig arrays windowed ? 0 no, 1 yes
144
145     // "Oeffentlich" schreibbare Informationen:
146     BWSetup req;         // Angefordertes Setup
147 };

```

2.4. **Makro PLAN_SIZE(n).** Ein Makro zur Berechnung, wie groß das Array für einen *fftw-plan* sein muss.

$n \bmod 3$ bestimmt den Typen des Plans:

- 0 : real \mapsto komplex
- 1 : komplex \mapsto real
- 2 : komplex \mapsto komplex

Das Makro berechnet:

$$PLAN_SIZE(n) = \begin{cases} 3 \cdot 2^{\lfloor \frac{n}{6} \rfloor}, & \text{wenn } n \bmod 3 \text{ ungerade.} \\ 2 \cdot 2^{\lfloor \frac{n}{6} \rfloor}, & \text{wenn } n \bmod 3 \text{ gerade.} \end{cases}$$

2.5. **static void copy_samples(BWAnal *aa, fftw_real *arr, int off, int chan, int len, int errors).** Eine Funktion, die den mit *off* und *len* ausgewählten Bereich der EEG-Daten eines Kanals (*chan*) einem *BWBlock* (*struct* definiert in *file.c*, *Details sind noch to do*) in ein Array kopiert. Geht der gewählte Bereich über die Grenzen der Datei hinaus, werden entsprechend Nullen in das Array eingetragen.

Anmerkung: Code ist nicht ins Detail überprüft, die Ausführungen beruhen nur auf den Kommentaren und einem groben Blick auf den Code.

2.6. **static void recreate_arrays(BWAnal *aa).** Führt erst einmal *free* auf alle Arrays des BWAnal-Structs aus, um sie dann anhand der im Struct eingetragenen Zeilen- und Spaltengrößen (*aa->c.sx* und *aa->c.sy*) neu zu allokalieren.

2.7. **BWAnal * bwanal_new(char *fmt, char *fnam).** Erzeugt einen neuen BWAnal-Struct für die Datei *fnam*, die mit den in *fmt* übergebenen Formatspezifikationen geladen wird. Dabei wird (neben einigen Default-Parametern, die später wieder überschrieben werden) folgender Parameter gesetzt:

Block size *bsiz* = 1024

2.8. **void bwanal_start(BWAnal *aa).** Die Funktion *bwanal_start(BWAnal *aa)* startet die Berechnungen. Sie wird aufgerufen von der *main*-Funktion (siehe *bwview.c*).

Vorbemerkung: Von den drei

sx := Anzahl Spalten

sy := Anzahl Zeilen

$tbase :=$ Samples pro Datenpunkt/Pixel

$wurat :=$ Ratio of window width to the centre-frequency wavelength

$log0 = \ln(f_{MAX})$

$log1 = \ln(f_{MIN})$

Für jede Zeile $a < sy$:

Mittenfrequenz: $f_a = e^{log0 + \frac{a+0.5}{sy}(log1-log0)}$

(Logische) Fenstergröße in Samples: $wwid_a = \frac{f_{SAMPLE}}{f_a} wurat$

$siz = sx \cdot tbase + \lfloor wwid_a \rfloor + 2 + 10$

$b = \lfloor \log_2(siz) \rfloor \cdot 6$

Solange $PLAN_SIZE(b) > siz$: $\{ b = b - 1 \}$

Index auf zu benutzenden fft Plan: $ftp_a = b$

Dazugehörige Fenstergröße: $awwid_a = PLAN_SIZE(b)$

Maximale Plangröße: $maxsiz = PLAN_SIZE(b)$;

} // Ende für jede Zeile

to be continued...