ExptManのポイント

RunSeries.mの２０行目でload(‘SpkrFile4’)

このSpkrFile4.matはLというstructとRというstructを含んでいていずれも

Fs

PassBand

FiltCoef

FiltGain

FiltDelayPts

SysResp

FlatNoise

というFieldを持つ。

RunSeries.mの２０行目でload(‘flatRB52nov28LequalR’)とすると全く同じ特性のLとRがloadされる。

２２行目で、名前を変えて

SpkrData=[L R]

とし、以後はこれを使う。

２３行目でSysRespとFlatNoiseの２つのFieldを除く結果、

Fs

PassBand

FiltCoef

FiltGain

FiltDelayPts

という５つのFieldが残る。

Frozenが選ばれれば（１３６行目から１５９行目も同じであるが、ここではさらにGainControlerも使う）

４２行目からが動く。

５２行目で[Signal, Atten, Duration]=feval(myStim{:}. FsRP2)

Tone.m

ここで例えばToneをGUIで作っていた場合myStimの

１列目は‘Tone’であり、fevalによってTone.mが稼働してmyStimの２列目以下の変数を使って、Signal, Atten, Durationを作成する。例えばmyStimの３列目はSPLである。

このTone.mの段階ではキャリブレーションで得た情報は使っていない。

Noise.m

例えばNoise.mの１００行目にあるように

OrigSPL=20\*log10(1/2sqrt(2)); OrigSPL=-3.0103dB re 1Vである。

７９行目で最初に作ったnoiseのRMSを計算する。

１０２行目で

Atten(1)=20\*log10(RMS(1))-(SPL+ILD/2);ここでRMSは固定値であり、GUIで入力されmyStimの３列目と６列目に残っている情報SPLとILDを使ってAttenは決まる。

Signal自体はキャリブレーションで得た情報は使っていない。

RunSeries.mの５７行目でSpkrData(i).FiltDelayPtsを使う

myDelay=SpkrData(i).FiltDelayPts

最初から送られてきたflat.matでは

L.FiltDelayPts=128である。

このDelay分の個数のゼロをSignalの最後に追加して、５８行目でSpkrData(i).FiltCoefを使う

mySignal=filter(SpkrData(i).FiltCoef,1,[Signal(I,;) zeros(1,mydelay)]);

ここまでにできているSignalベクトルにFiltDelayPts分の０を足したものにキャリブレーションで得たフィルターをかける。

RP2内のプログラムによるフィルターとER2を含めたシステムをSignalが通過したときに、これだけのDelayが生じることが、キャリブレーションの録音で分かっているので、５９行目で頭のDelay分のデータを除去すると、元のSignalがフィルターを通して出てきた分が残る。

**GetFiltCoef.mのStage1では**

まずNGolay=12を使って、2^12=4096個の1か-1からなるGolay符号を作る。これを２セットと間に4096個のセロベクトルをはさんで計16384個のベクトルを作る。

MaxVolt（ER2の場合はMaxVolt=2.5）を使って、上記の信号を2.5か-2.5の16384個のベクトルとしDataInとする。

これをMaxSPL（ER2の場合はMaxSPL=96）とともにPlayAndRecordAttenに代入してDataOutとAttenを得て、さらにNFiltCoef=256を使って、256個のベクトルであるFiltCoefを計算する。この過程で左上のSystem Freq Responseができる。

**GetFiltCoef.mのStage2では**

PassBandで指定されたflatなノイズを出す。

Stage1で求めたFiltCoefをかける。

このFiltNoiseのRMSはFiltCoefの値とPassBandの値によっても多少変わるはずだが例えばPassBand=[100 10000]の場合はRMS=約2Vである。TDT limitの10Vを考慮して10Vp-pになっている。

OrigSPLはGetFiltCoef.mの１８５行目で作った元のNoiseのRMSを1Vを基準にdB表示したものであり、FiltCoefの値とPassBandの値によっても多少変わるはずだが例えばPassBand=[100 10000]の場合は約-44.365dBである。

GainByScaleはTDT limitの10Vを超えないようにする配慮でPassBand=[100 10000]の場合は約50dBである。

このFiltNoiseをPlayAndRecordAtten.mに入力し、

まずRecSPLとして録音されたDataOutの音圧dBSPLを計算する。

GainByFilt=RecSPL-OrigSPL-GainByScale+Atten

RecSPLはdBSPL re 1uVであるが、OrigSPLはdB re 1Vであり、GainByScaleもdB re 1Vである。

AttenはPlayAndRecordAtten.mの中で１４３行目のDelta=CurrentLevel-MaxSPLを基に計算されている。CurrentLevelはdBSPL re 1uVであり、MaxSPLもイヤホンの特性に従ってdBSPL re 1uVで決めているから、AttenもdBSPL re 1uVである。

GetFiltCoef.mの２１５行目で、Stage2で発したNoiseと録音されたDataOutのCrosscorrelationをとって、これが最大になるところを探してDelayPtsByFiltを求めている。

ここまでで右上の256ポイントのFiltCoefのグラフと、GainByFiltとDelayPtsByFiltの表示ができる。

さらに左下のグラフがかける。このグラフではStage2で元々flatな特性で作った8192個のノイズ（元々の周波数は4096個であるがRealIFFT.mで逆FFTをかけると負の周波数の分もデータができるので２倍になる）。これに同じ個数のゼロベクトルを追加して16384個のベクトルができる。さらにStage1で求めたFiltCoefをかけてFiltNoiseとしているので、イヤホンと外耳道の音響特性を加味してPassBandの範囲がflatになってPlayAndRecordAttenからDataOutとして出力されるはずである。

そしてGetFiltCoef.mの２３０行目でもういちどDelayPtsByFiltを相殺すればAmpSpecが計算できる。これを左下のFlattened Noiseとしてグラフに表示する。

PassBandの範囲内のAmpSpecのSTDも計算し、表示する。

ER2の説明書には1kHzのところで1VのACで100dB±3dBと記載がるので、STDが3dB前後で得られれば実用性があるのだろう。

PlayAndRecordAttenSR2.mでは、９６行目の繰り返す回数が３のままでは、Flattened NoiseのSTDが3dBより小さくならなかったので、繰り返しを６に変更した。また、１６０行目に行を追加して６回目の試行の結果が[DataOut, Atten, SysEnv]=PlayAndRecordAttenSR2(….).mのAttenに出てくるようにした。

またPlayAndRecordAttenではイヤホンにあわせて設定したMaxSPLの少し下の１つの音圧で提示したGolay符号でFiltCoefを決め、Stage2でもMaxSPLの少し下の１つの音圧で提示したFiltNoiseを使ってFiltGainを測定しているので、結局はイヤホンは、入力電圧に対して線形に音圧を出すことが前提になっていると考えられる。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 音圧制限のための変数 | GetFiltCoef.m で規定されるMaxSPL  この5-10dB下付近の音圧でFiltCoefを測定する。 | GetFiltCoef.m で規定されるMaxVolt  イヤホンに安全な最大電圧。  ただしRMS V | ExptMan.figにmVで入力し、ExptMan.mで最初に受け付けられるMaxVolt  ExptManParam.MaxVoltとしてRunSeries.mに供給される。  このMaxVoltは刺激生成に使われることはなく、RunSeries.mの329行目で記録された反応波形を16-bit integerに圧縮するために使われる。すなわち-32768から32367までにおさめるためである。  RA4PAは4-channels 16-bit PCMであり、入力は+/- 4 millivoltsである。4mVの-3dBすなわち2.83mVでクリッピングのインジケータが光るとある。  ExptManParam.MaxVoltは7mVになっており概ね桁は対応している。  このMaxVoltはRP2PlayEz.mやRA16RecordEz.mやAnalyzeResponse.mでは使われない。 | GetFiltCoef.mおよびPlayAndRecordAtten.mで規定される、TDTでクリッピングしないためのリミット。  ただしp-p V |
| ER2  +ExptMan.m  +RunSeries.m  +RP2PlayEz.m  +RA16RecordEz.m  +AnalyzeResonse.m  用の設定 | 96 dBSPL  FlatSpecSR2.mとFlatSpecSR2.figを使ってFlatSpecSR2.matができる。  GetFiltCoefSR2.mを使う。 | 2.5 V RMS  GetFiltCoefSR2.mを使う |  | 10V  GetFiltCoefSR2.mを使う |
| ExptMan.m+ER4用の設定 | 104 dBSPL | 3 V RMS |  | 10V |
| RB-52T  +ExptMan.m  +RunSeries\_A1.m  +RP2PlayEzA1.m  +RA16RecordEzA1.m  +AnalyzeResonse.m  用の設定 | 122 dBSPL  FlatSpecRB52T.mとFlatSpecSR2.figを使ってFlatSpecSR2.matができる。  GetFiltCoefRB52T.mを使う。  Recordingのときは  ExptMan.mの176行目のRtn=RunSeiesの箇所だけをRunSeries\_A1に書き換えて使う。 | 0.35V RMS  (1Vp-p)  GetFiltCoefRB52T.mを使う |  | 10V  GetFiltCoefRB52T.mを使う |
| RB-52T  +ExptMan.m  +RunSeries\_CAP.m  +RP2PlayEz.m  +RA16RecordEz.m  +AnalyzeResonse\_CAP.m  用の設定 |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

GainByFiltはRtn.FiltGain=GainByFiltとして

RunSeries.mの７１行目で使われる。

Atten(i)=Atten(i)+20\*log10(myscale)+SpkrData(i).FiltGain:

ここのmyscaleは６４行目で先に作ったmysignalの振幅の最大値を計算してTDT limtの10Vを超えないようにする配慮である。

RunSeries.mでは

音刺激はSignalとしてV単位の電気信号として作成され、RP2PlayEzに代入される。Tone.mでもNoise.mでもキャリブレーションで得た情報は使っていない。

GetFiltCoef.mでFiltGainを計算することによって、Ｖ単位で作ったSignal（上で塗った緑の部分）をdBSPL re 1uVに変換している。

Atten(i)=Atten(i)+20\*log10(myscale)+SpkrData(i).FiltGain:でFiltGainが正のときは、外耳道の共鳴特性により鼓膜部分では大きくなることになり、この分をAttenに追加してあらかじめ小さく音を出すことを意味する。

上の緑のAttenは５２行目の[Signal, Atten, Duration]=feval(myStim{:}. FsRP2)のAttenそのままであり、GUIからの目標のSPLを使ってV単位で表示されているものである。

以下想像ですが、

RPvdsEXも少しずつバージョンアップしており、古川先生が最初に作られたときは

１）PlayAndRecord.rcoとPlayAndRecord.rpdの組み合わせで使い始め、その後

２）PlayAndRecord.rcoとPlayAndRecord.rpxの組み合わせにしたのではないかと

思います。

最近のRPvdsExでは.rcxファイル１つにまとめられるようになっていますが、今

でも２）の組み合わせで十分動きます。

１つ気になるのは、PlayAndRecord.rpxに使われているDACDelayです。

私はRP2.1を使っていますが、もしかすると古川先生はRP2を使っていたのでしょ

うか？

A→D変換で生じるDelayはRP2.1では65samplesで、RP2では40samplesです。

D→A変換で生じるDelayはRP2.1でも30samplesで、RP2でも30samplesです。

**結局新しいPlayAndRecordSR2.rcxでは**

**Components to record signalの方のDACDelayをもうひとつ追加して２個にした。RP2.1ではA/D Sample delay=65samplesであるので、DACDelay=30samplesを２倍して60samplesを相殺しようとしている。**

**しかしGetFiltCoef.mでイヤホン含めたシステム全体のDelayPtsByFiltを相殺しているので、結局.rcxファイル内のDelayは全くいらないのかもしれない**

狩野先生

本年もよろしくお願いいたします。

結論から言えば、50 kHzのサンプリング周波数は十分だと思っています。

まず記録用のサンプリング周波数についてですが、スパイク\*波形\*を記録するう

えでの帯域としては、5 kHz位までとれていれば、まず大丈夫だと思います。そ

の場合、ナイキスト定理から考えると、10 kHz以上のサンプリング周波数でよい

はずです。

スパイクの発生時刻を記録するためにそれ以上の精度が必要な場合は、記録した

データをアップサンプリングしたうえで時刻を計算すれば、いくらでも細かくで

きます。

自分が過去にやってきた研究では、スパイク発生時刻の精度としては、10-20

micro sec.ぐらいで十分でした。論文の載せるＰＳＴＨのビンの幅は細かくても

1 msec 程度でした。

古川

(2012/01/19 18:20), Shotaro Karino wrote:

> 古川先生

>

> 今年もどうぞよろしくお願いいたします。

> １２月に送っていただいたファイルを基に、RPvdsExとmatlabによるActiveX

> controlの概要は理解できました。

> おそらく刺激波形の作成は問題ないと思うのですが、電位記録の方で基本的な疑

> 問が湧いてきました。

> 先日教えていただいたプログラムではSampling Rateが50K Hzで、RP2.1のスペッ

> クの最高値でも200K Hzとあります。そうするとスパイクタイミングは一番細か

> くても5 mocrosecondsということになるのでしょうか？

> ベルギーでは外付けのpeak pickerでスパイクを検出し、System2のET-1(event

> timer)でtime stampしており1microsecondだったように覚えています。

> もしこのような細かさをITDの実験などで要求するときにはsystem3ではRV-8　バ

> ラキューダ？を購入する必要があるのでしょうか？

> PSTHを作るプログラムのところを精確に理解していないのかもしれません。

> お時間の許すときにご教示いただけますと幸いです。

>

> 狩野章太郎