rmouse - analysis of multi-electrode field potential data from rodent hippocampus

Harald Hentschke (hhntschk@gmail.com)
April 2020

1. Introduction

rmouse (rmouse.m and dependent functions and scripts) is a collection of Matlab routines designed to analyze multi-channel local field potential (LFP) data from rodent hippocampus. Hippocampal LFP data are characterized by partially interdependent rhythms in +- distinct frequency bands. Most prominent are theta oscillations (classically, 4-12 Hz) and gamma oscillations (40-100 Hz). In order to characterize these and other features of hippocampal LFP, rmouse performs a range of more or less standard analyses in the time domain (e.g. cross correlations) and in the frequency domain (e.g. spectral power).

The manual is written from a programmer's perspective for both users and programmers and is intended to provide an understanding of how the program works, where there are problems with the code and and how it could be improved. It has been developed in the laboratory of Robert A. Pearce, University of Wisconsin at Madison during a postdoctoral stay (2003-2005). Work on the code ceased after publication of the last paper for which the analyses were used, about 2009.
The code was resurrected in April 2020 for the Ten Years Reproducibility Challenge (https://github.com/ReScience/ten-years). It has been restructured and cleaned up to a minor degree, and basic functionality has been verfied. However, there is no guarantee that the more experimental avenues of analysis (see e.g. the paragraph on 'callJob') will work. Redesigning the code to run efficiently on recent hard-and software (parallelism, GPU) would have been tantamount to rewriting it from scratch and was therefore not done. The manual largely reflects the code's state around 2009.

2. Overview

It is assumed that the recorded neuronal data to be fed into rmouse stem from awake behaving animals (hence rmouse = short for 'running mouse' as the program name). Since different behaviors correspond to different neuronal signals in hippocampus, rmouse by default divides the neuronal data according to the animal's behavior and performs separate analyses on the subgroups.
The results of the analyses are saved as matlab (*.mat) files, binary data files or as figures. For each data file to be analyzed, vital information on the data and on the details of the analysis to be performed must be given in two files for which I will use the generic terms **dset** ('data set') and **anpar** ('analysis parameters'). Both are matlab scripts; in essence, they generate **global variables DS and AP**, respectively, which are matlab structures.
The analysis of data is broken up in **partial jobs** which must be explicitly specified by the user (in AP.job). There are two fundamentally different categories of partial jobs. The first category deals with the data irrespective of behavior. These are preparatory routines: for example, they filter data or detect theta peaks and write the resulting output on hard disk. Two very basic preparatory routines - one which digests information on the animal's behavior and another which detects artifacts in the neuronal data – usually run so quickly that their output is not saved on disk; instead, they must be run each time rmouse is called.
The preparatory routines provide the ground on which the second category of partial jobs builds: according to behavior, these routines pick subportions of the filtered data, theta peaks, raw data etc..They then divide these variably sized chunks into segments of a fixed size, analyze each segment, average and plot the extracted parameters and save them on disk. Once again, the computations are done separately for each behavior; all results, however, will be put in one large **results variable r**.

3. DS and AP

As mentioned above, both variables characterize the recording to be analyzed and govern the behavior of rmouse, respectively. Why not combine them into one? The idea is that there should be exactly one DS per neuronal data file describing fixed, unalterable properties of the recording, like the number of channels recorded, file name, etc.. When it comes to the details of analysis, however, we may want to probe the data in several different ways, and possibly keep all these different results (or at least parameters describing the approaches). Therefore, it is certainly useful to have more than one AP per data file, but we do not want to have a lot of redundant information in these. Hence the division.

Setting default values for analyses via a 'masterAP'

Some critical analysis parameters must be set to fixed values for a coherent set of experiments, otherwise averaging data would become ridiculously cumbersome. A good example is the number of data points per segment, AP.ppSeg. Among many other aspects, this parameter determines the frequency resolution of the power spectra. If after a first round of analysis one decided that the number of points per segment should change (e.g. increase to give a better frequency resolution), one really doesn't want to edit the several dozens to hundreds of anpar files already set up to change this parameter. An obvious solution is to set up a partial AP which contains standard settings which shall apply to all analyses. Let's call it 'masterAP'. This AP must be called immediately before running rmouse on a given data set (it may also be passed as an optional input argument to rmouse). Thus, extending the example above, we would change AP.ppSeg in the masterAP (instead of in all anpar files under consideration) and then of course still have to re-run all analyses.

Checks and changes of DS and AP

rmouse.m changes over time. Some analyses get dumped, new ones added, yet others revised. In many such cases it is inevitable to add new fields to DS and/or AP (and helpful to remove superfluous ones). From the programmer's point of view this is no problem but from the user's it is a real pain in the neck, very similar to the problem pointed out above: who would be eager to add a new parameter (=code line) to more than just a few anpar files? The first solution is identical to the solution explained above: add new fields to the masterAP. However, this approach will not work if we need to add parameters for which there is no default setting (example: DS.rawSignalInverted which had to be added after we found out that some recording hardware inverted the neuronal signals whereas other hardware didn't). In such a case, dsetanpar_fieldwork.m may be of help. It is a routine independent of rmouse.m which allows adding/removing/changing fields in a whole set of dset or anpar files.
In this context, it is important to mention that before performing any kind of analysis, rmouse performs a check of AP. It calls function rmouse_APcheck which calls **anpar_template.m**. *The AP created by anpar_template is the golden standard against which the AP of a data set to be analyzed is checked*: it must list all fields currently in use and, ideally, the most up to date explanations of each parameter's meaning. Whenever you (as a programmer) need to add fields to AP (or want to remove some), make sure you do this in anpar_template. There is also dset_template.m; however, since this structure did not change a lot no check is implemented.

4. Types of analysis

All analyses (partial jobs) that can be performed are listed in anpar_template.m. There is no 1:1 correspondence of jobs to script or function files: Some partial jobs are coded within rmouse.m, others are in scripts or functions whose file names are rmouse_*.m (* = wildcard). A complete list of all functions called by the 'main' function rmouse.m can be obtained by running Tools – Show Dependency Report (from the command window's or editor's menu).
Here is a brief description of the different partial jobs:

det_artifacts

Detects artifacts in neural signals & generates time stamp list thereof.

gen_btsl

Generates behavioral time stamp list from trigger pulses or text file (see chapter below). Many partial jobs below depend on this job as well as det_artifacts, so it is strongly advised to always run them.

chechao

Plots cross correlations of raw data in single figure window so that proper spatial order of channels can be verified.

filter&hilbert

Filters raw data in specific frequency bands and writes the resulting 'streams' (theta, gamma, etc.) on hard disk (as 2 byte-integers). Additionally, the envelope streams (gamma and theta) are generated and written to disk. The stream files are accessed by the jobs computing cross correlations. As they are saved on disk, `filter&hilbert` needs to be run only once (unless the filter settings change or the files got lost of course). Note that the stream files take up a lot of space on hard disk.

thetaPeaks

Computes amplitude & time of occurrence of peaks in theta stream.

seg_thetaPeakReg

Computes theta regularity in terms of coefficient of variance of peak amplitude and inter-peak intervals.

specgram

Computes & plots spectrogram (time course of spectral power)

seg_rawSpecPow

Computes power spectral densities & derived parameters (all pairwise, non-redundant combinations of channels).

seg_gammaEnvSpecPow

Computes spectra & derived parameters for gamma envelope (all pairwise, non-redundant combinations of channels).

seg_deltaCC, seg_thetaCC, seg_*…

These jobs compute cross correlations of delta, theta, … streams (all pairwise, non-redundant combinations of channels).

seg_thetaGammaEnvCC

Computes crosscorrelations between theta and gamma envelope (for each channel).

seg_deltaThetaEnvCC

Same as above but for delta and theta envelope (for each channel).

seg_cc_ms2rad

Converts theta-related phase lags from ms to radians, creating new fields of results variable r (NOTE: will be called automatically if any of `seg_rawSpecPow, seg_thetaCC, seg_gammaEnvCC` or `seg_thetaGammaEnvCC` is called)

sumFig

Produces summary plots. Do not run this job in conjunction with memory-demanding jobs like the cross correlation jobs if your computer reports 'out of memory' errors. The reason is that graphics tend to clutter memory. You can always run it post-analysis.

rawPlot

Generates plot of raw data excerpts.

`ovRawPlot`

Produces overview plot of raw data (the whole excerpt to be analyzed).

`csdPlot`

Produces contour plot of current source densities from theta stream excerpts (the plotting part may take awfully long).

`tcFig`

Plots time course of segment-wise computed, selected parameters

`tcPrincComp`

Computes & plots principal components from tupels of segment-wise computed, selected parameters. The idea is to see how much the behaviors differ from each other in terms of a compound variable made up of several parameters extracted by the cross correlation etc. analyses. Appealing graphics, but so far no 'serious' part of the analysis.

`tcThetaPeak`

Produces blob-plots depicting time course of theta peaks (the plots have an aesthetic appeal and may reveal variability of theta).

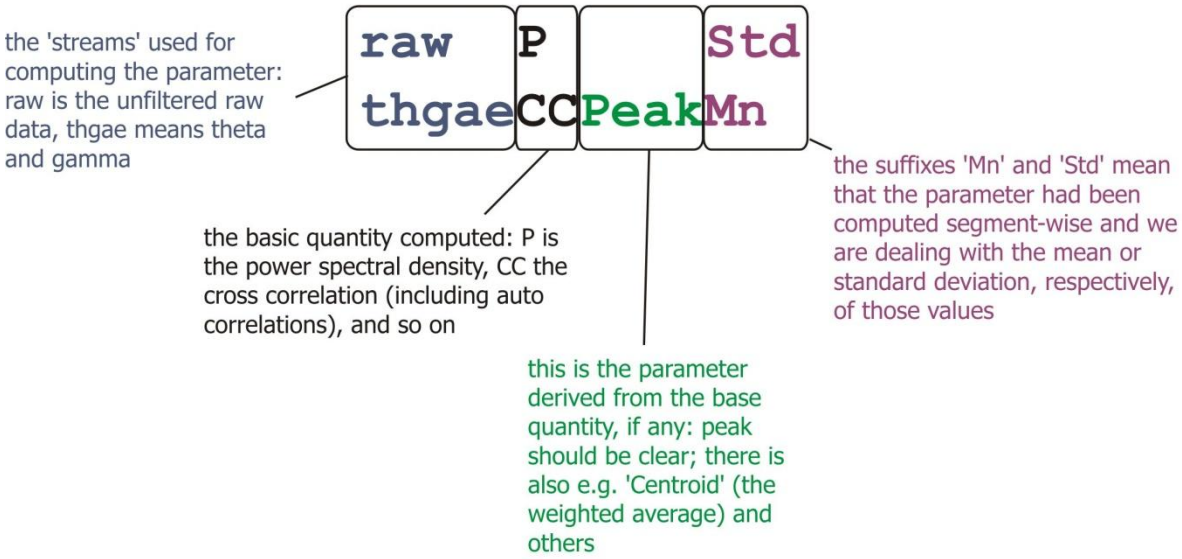
Further information can be found in the code accomplishing the partial jobs, which contains at least some comments.

Interdependency of partial jobs

Numerous results of the outputs depend, in a direct or indirect manner, on several of the analysis parameters. For example, all cross correlations depend on the filter settings in an indirect way: they are computed from streams (files on disk) previously put out by job 'filter&hilbert', which filters the raw data according to the specifications given in e.g. ANPAR.thetaCfreq. There are many more dependencies of this obvious sort - and a few not so obvious ones. Therefore, when playing around with analysis parameters, either make sure you know exactly what effect changing a particular parameter will (or will not) have or run the whole analysis from scratch.
For reasons of interdependency of results the order of partial jobs should not deviate from the one given in anpar_template. For example, job ' seg_cc_ms2rad' needs the theta peak frequency of the principal channel to compute phase lags from ms to radians; therefore, it must be preceded by jobs ' seg_rawSpecPow' and ' seg_gammaEnvSpecPow'.

Playful or very special analyses: optional input variable 'callJob'

Specific jobs (=matlab scripts or functions) may be run from within rmouse by specifying the function/script name as an (optional) input parameter to rmouse. The Matlab `eval` function is used to this end. This option is useful for running e.g. test analyses or very rarely used analyses which rely on the preparatory work done in rmouse but should not be part of the standard staple of analysis jobs. Note that the function or script will be run after conclusion of all other jobs.



5. Mixed technical details

Time stamp lists

The trigger pulses coding for the onset of certain behaviors of the animals (see further below) are essentially 'time stamps', that is, points in time at which something important happens. In rmouse.m the variables holding them are in 'tsl' or 'etsl' ((extended) time stamp list) format. This is a simple set of conventions as to how the data should be organized. The tsl or etsl are modified (merged, cut down in size, etc.) by functions which expect this format. See primer_evltsl.rtf for more information.

The logics of channel indexing

- explanation to come -

Example of user-defined channels

all channels listed in DS

DS.rawCh

LFP	IN 0
	IN 1
	IN 2
	IN 3
	IN 4
EEG	IN 5
EEG	IN 6
scoring	IN 7

all channels to be analyzed (regardless of type)

AP.rawChAnNm

IN 0
IN 1
IN 2
IN 4
IN 6

'principal' LFP channel

AP.rawChPrincNm

IN 4

Channel indices derived within rmouse from DS.rawCh and AP.rawChAnNm

DS.rawCh

IN 0
IN 1
IN 2
IN 3
IN 4
IN 5
IN 6
IN 7

all channels to be analyzed (regardless of type)

AP.chIdx

1
2
3
5
7

all LFP channels to be analyzed

AP.LFPIdx

1
2
3
5

all LFP channels (no matter whether to be analyzed or not)

AP.allLFPIdx

1
2
3
4
5

principal channel

AP.pcIdx

5

AP.rawChAnNm

IN 0
IN 1
IN 2
IN 4
IN 6

AP.LFPInd

1
2
3
4

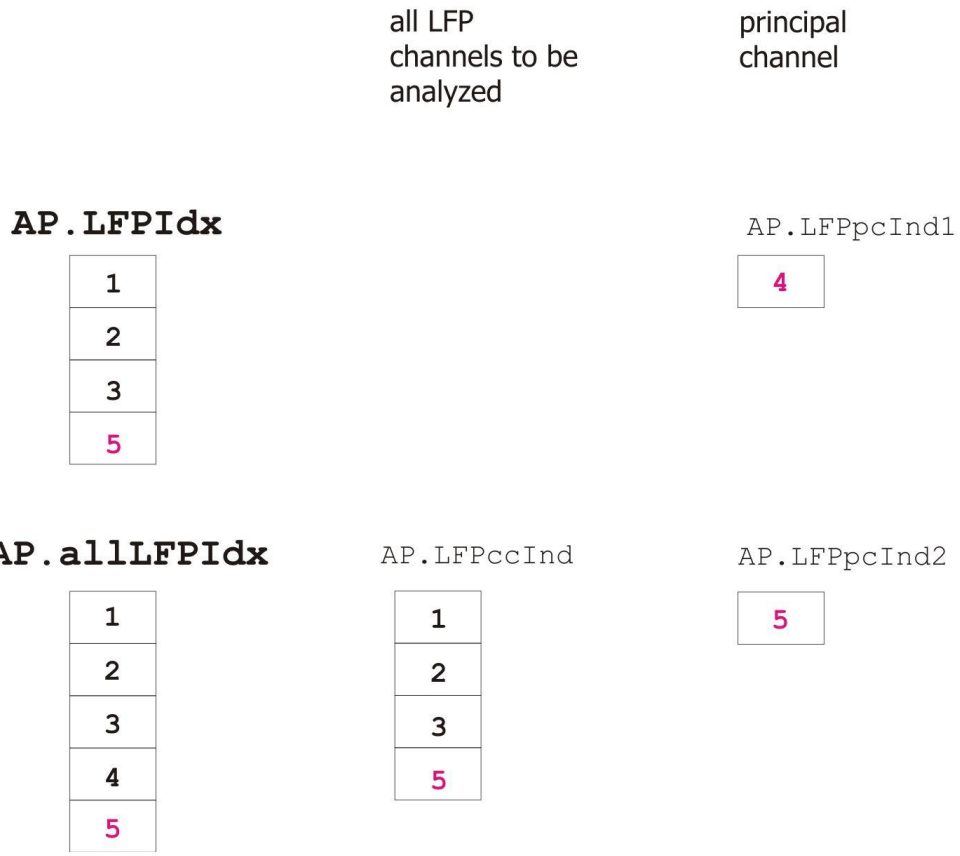
AP.allLFPInd

1
2
3
4

AP.pcInd

4

Channel indices derived within rmouse from `AP.allLFPIdx` and `AP.LFPIdx` (=derived from LFP channels, in other words, the most-used indices within rmouse)



6. Known problems and inconsistencies & ideas for future improvements (list is incomplete)

Programming issues

- reading behavioral scoring text files: tabs & other whitespace after last entry per line derail things
- global variable logstr collects all (or most) of the information dumped on screen but is not used so far for writing to logfile (matlab diary function does that)

Analysis issues

- artifacts are detected based on a simple threshold which must be determined outside matlab. Maybe develop a GUI (even a separate function?) displaying raw data & threshold + recommendation for threshold (percentiles or so)?
- Similar thought applies to the detection of the theta peak in the spectra: all the parameters which influence theta peak detection had better be tested in a program independent of rmouse which is interactive.

- alternatively, both points above plus a few other quick analyses listed below could be combined in a 'snoopy' procedure which should facilitate analysis setup:
- overview of behavior
- power spectra (all channels, but only auto-spectra and on ~10 % of data); whole range (i.e. 1-100 Hz) and theta only with freq response of filter used for peak det overlaid
- spectrogram (principal channel only)
- maybe theta raw CC (~10 % of data, only principal channel vs others)
- compute confidence intervals instead of standard deviation for spectral densities etc.? This would be of particular relevance for coherence -> see toolbox\signal\signal\private\chi2conf.m

7. Special topic for users: Behavioral scoring

1. Agile mice: recording of trigger pulses with axoscope

Behavioral changes of animals can be registered by recording short voltage pulses which code for different behaviors shown by the animals. Pulses are generated by Mark's switchbox; their duration is fixed, whereas their amplitude (currently 6 different levels) codes for the type of behavior. The occurrence of a pulse marks the onset of a behavioral epsiode of a certain type; the occurrence of the next pulse (of different amplitude) marks the end of the current and the beginning of the next behavioral episode.
If behavioral scoring is done during the neural recording (i.e. one channel of the abf file is dedicated to recording the pulses), timing is not an issue. For post-experimental scoring, however, proper synchronization (or rather, timing information) between the neural recording and the recording containing the pulses is crucial. Current strategy:

- start replay of camera tape
- as soon as timer is visible in camera view and starts counting, decide on a specific time t_0 and start recording at this time (ideally, a few seconds after timer is visible/starts counting)
- write down t_0

Other important points

- the first pulse of the recording should be triggered as soon as possible (because the behavior up to that pulse is undefined and thus the corresponding data cannot be analyzed)
- pressing a trigger button of one type twice or repeatedly is redundant, but in no way detrimental to analysis
- pressing trigger buttons of different types in quick succession may lead to loss of analyzable data. This is so because there is a minimal length an individual data segment may have to be usable for analysis (currently ~4 seconds). Nonetheless, if the animal quickly switches from one behavior to the other, press the buttons. If you made a mistake (confused buttons) and realize it, just quickly press the correct button.
- obviously, recording of the scoring pulses should be stopped some time after the 30 or 60 min experimental period is over; however, there is no need to tightly synchronize the ends of the recordings

2. Sleeping mice: writing it down in text files

The animal to be scored may show only one kind of behavior for the largest part of the 30 or 60 min recording (e.g. it is immobile due to effects of an injected drug). If this is anticipated, it is more efficient to watch the video in time lapse mode and to write down (in a text file) the times when the animal changes its behavior from immobility to something different. The information needed is the same as with the switchbox approach, that is, the start times of the various behaviours must be known, and it is assumed that the start of a behavioral episode marks the end of the preceding one. The file should have the following format:

- we need plain text files, the file extension being `'txt'`. The standard windows notepad does the job. If you use e.g. word to generate files, make sure the format is text (without line breaks)
- the start times of behavioral episodes must be given in the first column of the file, as minutes and seconds with a dot in between, e.g. like 11.25 (eleven minutes and 25 seconds). There may be more than two digits of precision for the seconds (11.259 would be 11 minutes and 25.9 seconds)
- The second column, separated from the first column by nothing but tabs or white space, must contain a number coding for the behavior. In principle, you could choose any set of numbers, like 0 for immobile, 1 for exploring, etc. Ideally, however, the numbers correspond to the pulse levels that would be used if the behavior were scored with the switchbox (this is useful in case you decide to re-score the file using the switchbox). Example: look up parameter `AP.behavType` in the corresponding AP. Let's assume one line says `'exploring', [6 10], [.5 1 .5], 'o';...` . In this case, the number coding for exploring behavior could be 7 (which is between 6 and 10). In the example below, 3 codes for immobility and 1 for grooming.
- The file may contain comment lines which must begin with %.
- example of what a text file might look like:

```
% jan 28 2004, behav scoring wt xyz, etomidate x mg/kg
0.0     7
9.12    3
9.14    7
12.02   1
12.58   3
```

- very important: note the time frame to which you refer! In most cases, you will read the time from the timer in the camera view. In this case, t_0 (mentioned above) is zero. However, should your reference be anything else, take it down.
- the first line must describe what the animal did at the outset of the scoring session (this is in analogy to the first trigger pulse mentioned above, which should be triggered as soon as possible after the scoring session started)
- the present version of rmouse will not be happy with any kind of whitespace character (space, tab, etc.) after the second number in any line. If you get funny error messages, double check that the text file is clean.
- skimming through the video in time lapse mode is the crucial time-saver, but for getting the exact times of changes in behavior you will probably need to re-watch the tape at normal speed around the crucial points

3. Vital information needed for analysis

The following is a description of parameters needed for analysis of the data in matlab. In other words, *all of this information must be put down somewhere.*

parameter/description	**name of matlab variable**
name of file containing the neural signals	`dset.abfFn`
name of file containing the behavioral scoring information (*.abf or *.txt)	`ANPAR.bScoreFn`
t_0 mentioned above: point in time at which behavioral recording was started, relative to	`ANPAR.bbor`

the beginning of the neural recording (as indicated by timer in camera view)	
the types of behavior scored and the trigger levels coding for them.	`ANPAR.behavType`

List of functions belonging to the rmouse project (last updated June 2006)

Note:

- throughout this script, the star character * is used as a wildcard (=a placeholder for anything)
- AP, ANPAR, DS, DSET, WP are all global variables. If they are listed as necessary input, they must exist (have been created) before invoking the function in question. The latter also applies to r, the major output variable of rmouse: some functions require it to reside in the workspace, which means it must be loaded from the results *.mat file
- ANPAR and DSET are structure *arrays*, that is, concatenations or even 3D matrices of individual AP and DS. Their dimensionality matters! (i) one column per animal/experimental session (ii) one row per concentration (for pharmacological experiments), (iii) one 'slice' (3rd dimension) per genotype. Bear this in mind when developing your own flavor of collect* routines.
- all functions in gray are still under construction, playgrounds or need revision, for which reason documentation may be sparse or missing
- all functions (except the collect_* routines, see there) reside in the rmouse directory or one of its subdirectories (auxil or construction)

Function/script name (*.m)	input needed	output generated	what's it good for
a few potentially useful functions/scripts which must be run from within rmouse.m by specifying their file name as optional input argument 'callJob' to rmouse.m:			
rmouse_corr_tc			correlates segment-wise computed, selected parameters with each other
rmouse_au			spiel (& under construction): generate wav files encoding theta, gamma and ripple signals as drumbeats, pink noise and chirp, respectively
rmouse_spiel			nomen est omen: this file and its namesakes are playgrounds
functions/scripts which are part of rmouse but will also work outside rmouse.m			
dset_template, anpar_template		DS, AP	as the names suggest, both contain the complete set of fields (including up-to-date comments) of DS and AP, respectively, and should be used as templates when setting up new data files for analysis
rmouse_ini		fields of WP	field .rootPath may be useful
rmouse_chan	AP, DS, WP	fields of AP and WP	generates all kinds of indices for channels
rmouse_APcheck	AP, DS	modified AP	deletes from and/or appends fields to an AP using anpar_template as the blueprint
useful accessories:			

chanCheck			this is a 'programmer's watchdog' - a function with which you can verify the correctness of the numerous channel indices produced by rmouse_chan
gen_btl			will probably never be used again, but anyway: converts behavioral timing information from one format into the other. Input format: episode number (one episode corresponding to 16384 points), output format: minute.seconds (as is the format rmouse.m requires). This was necessary for recordings with hand-scribbled notes on behavior.
abf2mat			similar story as with gen_btl, was needed only for the odd data set. This routine resamples data in an abf file and (since no routine for generating abf files is available) saves the resampled data in a *.mat file (which rmouse will automatically recognize as the data to analyze). The problematic data set(s) in question had a sampling frequency which was way too high.
abfmerge2mat			This routine merges data from two or more abf files and saves them as a *.mat file. Useful if e.g. a drug injection occurred in the middle of the second datafile (of two) and the pre-injection portion of that second file contains valuable control data
abftgap	data files	timing information	puts out time difference in seconds of the beginnings of two abf files. This information is needed e.g in plots spanning two data files
fieldsize	r (the major output of rmouse), must be in workspace		produces bar plots of the memory taken up by all fields of results variable r. In case the major output output file of rmouse appears too large, you may use this function to see which part of the results takes up most space
dsetanpar_fieldwork		modified DSET/ANPAR files	In the development of rmouse it was inevitable to add new fields to DS and/or AP on several occasions. At one point, writing a routine that does some guided batch replacement/insertion/deletion/renaming of selected fields in the dset*.m and a*.m files seemed more effective than doing it manually on the dozens if not more files that had accumulated. dsetanpar_fieldwork is the result of that contemplation. Needless to say, use it with extreme caution, test it on dummy dests/anpars.
strmview	AP, DS, *.i16 stream files , abf file	nice plots!	standalone 'stream viewer' for data streams (gamma, theta, etc.) as generated by rmouse
strmprobe	AP, DS, *.i16 stream files , abf	whatever you want	based on strmview, this function is open for all kinds of snooping data analysis based on the different streams

	file		
findBehavTransit	an 'etsl' and number codes for behavior	time points (in seconds) of behavioral transitions	the idea is to be able to pick out data segments containing transitions from one behavior to another (e.g. immobile -> exploring). This is useful for plots of raw data, e.g. for papers/posters. 'etls' stands for 'extended time stamp list'; this is the output rmouse.m generates with the job 'gen_btsl'. Behavioral codes depend on AP.behavType and will be put out on screen automatically with each run of rmouse
rview	AP, DS	plots	the idea is t have a function which plots e.g. the raw crosscorrelations of one data set. Needs serious revision
post-rmouse analysis routines:			
collect_*		ANPAR, DSET	all of these are scripts collecting and concatenating individual AP and DS. Since they are essentially listings of experiments, and for that matter come in different flavors and frequently change, they belong to the data directory. As noted above: **The dimensionality of their outputs, DSET and ANPAR, matters!** The conventions (assumed by the combine_* functions) are: (i) one column per animal/experimental session (ii) one row per concentration (for pharmacological experiments) (iii) one 'slice' (3rd dimension) per genotype
combine_*	ANPAR, DSET	anything from plots to statistical results	in general, using the information in DSET and ANPAR these functions extract relevant parameters from a set of individual results files and combine them. Some special versions and the older pilot versions combine2-4 also do plots and statistics (see below).
combine_bv	ANPAR, DSET (1D or 2D)	plots and screen output	computes the proportion of time spent in different behaviors and (i) plots its time course (all sessions lumped together) and (ii) puts out the numbers for each session. Technically, this function is a little different from the other combine routines in that it does not access the *.mat data files, but calls rmouse.m with the single job 'gen_btsl'.
combine_rr	ANPAR, DSET (column arrays with more than one row)	plots and R (extracted data)	extracts 'raw results' (rr) like power spectra from a collection of data sets and plots difference spectra. The function is specifically designed for experiments with application of a drug, i.e., for ANPAR and DSET with more than one row. It will not work with ANPAR and DSET containing just a single row.
combine_tc	ANPAR, DSET (exactly one column)	plot	plots time course (spanning several data files) of selected variables. The variables are listed within the function itself.

combine_fComod	ANPAR, DSET (column arrays with more than one row)	extracted data (R) in matfile	works similar to combine_rr. it sums crosscorrelation values in the frequency 'comodulograms' for specific combinations of frequency bands, e.g. theta-beta, theta-gamma, etc., and saves them
combine2	ANPAR, DSET (single column or single row)		the simplest version of the combine routines: generates plots and compares behaviors 'immobile' and 'exploring' for selected variables (listed within function itself). This is the only combine routine where dimensionality of DSET/ANPAR does not matter as long as they have *single* rows or columns
combine3	ANPAR, DSET (2D)	R	Extracts selected parameters from a collection of data sets. The function is specifically designed for experiments with application of a drug, i.e., for ANPAR and DSET with more than one row. However, it will also work with ANPAR and DSET containing just a single row. -> see Rdeal01
combine4	ANPAR, DSET (2D)	R (option to write to results files auto.mat and cross.mat), optionally plots	Extracts selected parameters from a collection of data sets. Compares groups of animals with unmatched data (e.g. different genotypes) and optionally does a 2-way ANOVA with repeated measures. This is done for each of the behaviors. Has not been designed for experiments with application of a drug.
combine4b	ANPAR, DSET (2D or 3D)	R (option to write to results files auto.mat and cross.mat), optionally plots	A generalization/modification of combine4 with the following features: - statistics consist of curve fitting & F-test - there is the option to do the statistics on either genotypes, behavior or drug effects
combine_r	ANPAR, DSET (2D or 3D)	R (will also write R to file)	Most general combine version stripped of the plotting and statistics functionality of combine2-4. Saves all relevant information in file.
combine_snoopy*			
set_rv			helper function for the combine2-combine4 functions, defining the results variables to examine
rdeal			produces working-style plots and statistics of selected parameters/rec sites/factors etc. as present in R. R is the collection of results as generated by **combine_r** (!). This is the newest and most general of the rdeal* functions
rdeal01			extracts selected parameters from results structure R. R is the combination of results from several experiments, won by running combine3.m
rdeal02			does a session-by-session comparison of selected results variables as computed by rmouse.m and

			combined for all sessions by combine4b.m. Comparison is done for control vs. drug; it consists of curve fitting +running an F-test (like in combine4b.m). To run the script, the contents of auto.mat or cross.mat must be loaded
specExtract			extracts spectra from combinations of data sets
r_megaplot			generates collection of plots from all results files for single variable
megaplozz			
megamegaplozz			
stgp			segTypeGlobP contains all behaviors that are of interest in summary plots
iState			
rcat			
rcatfish			
ccmovie			under construction, should at one point replace qdmovie
qdmovie			
snoopy*			

rmouse to do-list

major restructuring

- makeover of field names of r, possibly even subfields (r.cc., r.spec, etc.)
    - signal type (raw, stream (ga, th, …)) from which parameter was computed
    - frequency band under consideration (e.g. gaeTh means that in the gamma envelope stream the theta band is under consideration)
    - kind of parameter computed
    - qualifier indicating whether the field holds the mean, standard deviation etc. of the parameter in question
- detection of artifacts: display results in GUI
- detection of cc peaks: template instead of automatic detection. One template per stream in mat file
- make scripts functions:
    - global DS AP WP r logstr
    - rawCh as input arg
    - dpath=DS.dpath;
    - abfFn=DS.abfFn;
    - bsFn=AP.bScoreFn;
    - verbose=WP.verbose;
    - osi -> WP.osi
    - nLFPCh -> AP. nLFPCh
    - nAllLFPCh -> AP.nAllLFPCh
    - ccTemplate -> WP.ccTemplate
    - ccDerTemplate -> WP.ccDerTemplate
    - psTemplate -> WP.psTemplate
    - nccix -> WP.nccix

- diagNccix -> WP.diagNccix

new stuff

- include segment-wise computation of total power of raw data and gammaEnv
- (-adapt # rows in ovrawplot according to length of rec data)

small fry/detail

- D is global in rmouse – why?
- check mem demand of i16 stream writing
- theta-gammaEnv computation: get rid of the 'envelope' parameters and instead try to compute amltudes & lags from adequate positive peaks
- gen_btsl is required for any job, so why not include in precomputation
- up-to-date rmouse_ini (memory issues)