PsPM: Psychophysiological Modelling

February 4, 2015

Version 3.0

incorporating SCRalyze

by the PsPM team[1]:

Dominik R Bach, Samuel Gerster, Saurabh Khemka, Christoph Korn, Tobias Moser, Philipp C Paulus, Matthias Staib, and collaborators

Contents

[1]If you have comments on or error corrections to this documentation, please send them to the PsPM team or post them on: pspm.sourceforge.net

Part I
Background

1 What is model based analysis?

1.1 Conventional analysis

Psychophysiology is concerned with inferring psychological states from physiological (often peripheral) measurements such as skin conductance responses (SCR), heart beats, respiration, and others. Conventional approaches in psychophysiology seek to identify data features ("indices") that closely follow a psychological state of interest. Conventional data analysis algorithms extract these data features, to "index" central states. For example, to infer stimulus-evoked sympathetic arousal (SA) from skin conductance responses (SCR), one may filter the SCR data to reduce observation noise, define a response window after the stimulus, and define some criteria do detect peaks within this window. The amplitude of such peaks may then be taken as "index" of SA. The development of such indices has been a major endeavour in the field of psychophysiology. It is usually based on qualitative or semi-quantiative models of how the index relates to the central state of interest. Model-based analysis provides a formalisation of this process drawing on strict, quantitative causal models - see [1] for a review.

1.2 Model-based analysis

Model-based approaches start with explicit, mathematical models that formulate how observed data are generated by central processes. For example, a model may formulate the relation between sympathetic arousal (SA) and skin conductance responses (SCR), $SA \mapsto SCR$ in mathematical form. This kind of model, in general, is often termed a "forward model": it predicts a data time series (SCR) from a known central process (SA). Another term for this kind of model is "generative", because it describes how a central process generates the data. When we analyse experimental data, we are faced with the opposite situation: we know the observed data data but not the central process, and seek to infer the central process from the data. In order to do so, one has to turn the forward model backwards, to arrive at the relation $SA \leftarrow SCR$. In statistics, this process is often termed "model inversion". It provides the most likely estimates of the central process, given the observed data and the model. Both conventional and model-based analysis aim to infer central process from observed data. The difference is that model-based methods use a stringent mathematical language and probabilistic model inversion to do so. Note that probabilistic models acknowledge the existence of noise in the data, and they do not seek to explain that noise with variation in central states. Hence, different from what may be a common misconception in the psychophysiological literature, the criterion for the quality of a model-based method is not how well the model "fits the data", but how well it can recover the central states of interest.

1.3 Evaluation

How can we infer that one method of inferring a central state is better than another? Each method returns indices or estimates of the central state, but which ones are more precise? Often we are interested in psychological states which can not be measured directly. We propose to use experimental manipulations to create two central states that are known to differ. We can then evaluate how well a method recovers that difference. We have termed the sensitivity to distinguish these central states "predictive validity". Simply speaking, when we compare the two central states with a t-test on their estimates, the test statistic should be as high as possible. In a more rigorous fashion, we can frame this evaluation as comparison of non-nested predictive models, and use a measure of model evidence to assess the difference between methods. This approach also allows a principled statement whether methods are significantly different. All methods included in this software have empirically been shown to provide better or equal predictive validity than the best available corresponding conventional index.

2 Skin conductance (SCR) models

2.1 General

Skin conductance responses are elicited via the sympathetic nervous system. Hence, SCR models relate a psychological construct termed "sympathetic arousal" (SA) to observed SCR [2]. All SCR methods split up the relation between SA and skin conductance responses $SA \mapsto SCR$ into two relations: $SA \mapsto SN \mapsto SCR$, where SN is the sudomotor nerve activity. The peripheral model $SN \mapsto SCR$ is a biophysical model that specifies how SN activity generates SCR. The central or neural model $SA \mapsto SN$ embodies knowledge and assumptions about the kind of SN impulses that are generated by a central (psychological) state, SA.

2.2 Peripheral model

Peripheral SCR models use a standard convolution model, also termed linear time-invariant (LTI) model, introduced in [3, 4]. This model is assumed to describe all processes between initiation of a SN burst in the CNS, and the measured SCR. All these processes are not described in biophysical detail but instead lumped into a phenomenogically derived impulse response function which is convolved with the sudomotor input:

$$SN \mapsto SCR$$

$$SCR\,(t) = SN \otimes SCRF + \varepsilon = \int_0^t SCRF\,(\tau)\,SN\,(t-\tau)\,d\tau + \varepsilon\;,\; t \geq 0$$

where ε describes the observation error.

Properties

The peripheral convolution model has two specific testable properties. There is direct (invasive) and indirect evidence for each of these properties.

Time invariance: the system's response does not explicitly depend on time, or in other words, a given input always produces the same output.

- Direct evidence: The amplitude of individual SN bursts is linearly related to the amplitude of ensuing SCR, with a considerable scatter [5]; to the maximal rate of sweat expulsion; and, somewhat more weakly, to the integrated sweat production during the skin response [6]. After initial dishabituation, constant SN stimulation leads to SCR, with constant amplitude and latency [7]. Repeated SN stimulation yields slightly different SCR shapes [8], although this could be due to variation in elicited neural responses that were not measured downstream. In summary, these findings are consistent with (but do not prove or disprove) the time invariance principle.

- Indirect evidence: We have shown that for short events ($<$ 1 s duration) that are separated by at least 30 s, more than 60% of the variance in (high pass filtered) SCR can be explained under an LTI model. We have established the evidence for linear models using aversive white noise bursts, aversive electric stimulation, aversive pictures, auditory oddballs, and a visual detection task [4]. Because only 60% of the variance can be explained under this model, there is substantial residual variance - 40% of total variance. Is this residual variance due to violations of the LTI assumptions? We have shown that in the absence of any event for 60 s, signal variance amounts to more than 60% of total variance during stimulus presentation. That is to say, by introducing events, residual baseline variance is reduced by 20%. This suggests that residual variance is not caused by violations of the time invariance assumption but variation of the input (for example, spontaneous fluctuations), and observation error.

Linearity: the system's response does not depend on previous inputs.

- Direct evidence: Latency of sweat expulsion is slightly shortened when sweat expulsion rate is very high (i. e. for very strong SN bursts) but not when SN firing is frequent [6]. SCR to individual SN stimulations depend linearly on skin conductance level, and are slightly suppressed upon repetition of the stimulation after 1 s [9]. This suggests that the linearity principle does not hold under all conditions.

- Indirect evidence: The linearity principle amounts to saying that responses are not influenced by the (preceding) baseline signal. We tested this assumption by presenting two subsequent events with an inter stimulus interval (ISI) of either 2 s, 5.5 s, or 9 s, such that the baseline signal at these three time points differed markedly. Violations of the linearity principle in the peripheral system would imply that the amplitude of the subsequent response is dependent on the baseline signal, and thereby, upon the ISI. The second response was always smaller than the first. However, this was not dependent on the ISI, and hence not on the baseline signal. We interpret this effect as central repetition suppression (of the inputs into the peripheral system) and conclude that linearity is appropriate for the peripheral system. In other words, the peripheral response to one input is not modulated or affected by the response to another, even when they overlap in time [4].

In summary, it appears that both properties are good approximations to reality with the expection that non-linearities may occur if the time interval between two SN firing bursts is very short (< 1 s) or when the SN firing burst is very strong. It is in principle possible to model non-linearities in a linear model, using Volterra kernels. This approach is implemented for analysis of functional magnetic resonance images in the software SPM [10, 11]. It is not implemented for SCR models.

Skin Conductance Response Function (SCRF)

The peripheral model in its general form does not specify a particular form of the SCRF. There are three principled ways of defining a SCRF:

1. Individual response function. The most precise option is probably to measure the SCRF for each individual by providing brief inputs into the peripheral system and measuring the response. PsPM allows specifying individual response functions. The benefit of this approach has not been empirically tested.

2. Canonical response function. The most parsimonious method is to use a so-called canonical SCRF which is assumed to be constant across individuals. We have previously shown that a canonical SCRF can explain 50% of the variance in individual SCR elicited by various stimuli, while individual SCRFs explain about 75% of the variance [4]. Using a canonical SCRF, linear and non-linear models show a good predictive validity, superior to conventional methods [12], see also (Staib et al., sumitted).

3. Constrained response function. One can estimate an SCRF from the data of an actual experiment and use this for analysis. It is usually necessary to constrain the shape of the SCRF to make the model estimable. PsPM provides two options for this:

- In GLM, one can estimate parameters of the (orthogonalised) time derivative of the SCRF. In order to combine this with the canonical SCRF, the response can afterwards be reconstructed for each experimental condition, and the peak amplitude taken as estimate of SA. This approach has higher predictive validity than using the canonical SCRF alone. Less constrained options for estimating individual SCRFs are also available (in particular, using the dispersion derivative together with the time derivative) but have been shown inferior [12].

- In the non-linear model for event-related SCR, one can use the combined data from all evoked responses from one individual to estimate parameters of a third-order linear ordinary differential equation (ODE) for each participant. This approach only considers data within the inter-trial interval. In validation experiments, the ITI was not long enough to include the tail of the SCRF, and this approach has empirically been shown inferior to using a canonical response function in most cases [Staib et al., submitted].

Canonical Skin Conductance Response Function

The canonical SCRF that is currently implemented was derived from a large dataset of 1278 SCRs from 64 individuals subjected to different experimental manipulations (white noise 85 dB and 95 dB, painful electric stimulation, aversive IAPS pictures, detection of auditory oddballs, detection of a visual target in a stimulus stream). We extracted and mean-centred the 30 s following each stimulus onset and performed a principal component analysis. The first principal component served as empirical SCRF [4].

Formulation for linear models The first PCA component of the empirical SCRF was approximated was fitted by a Gaussian smoothed bi-exponential function (also named bi-exponentially modified Gaussian function), and this provided better fit than other function categories, in particular Gaussian smoothed monoexponential function, exponential-Gaussian hybrid function, or smoothed sigmoid-exponential function. The function is thus defined as

$$SCRF_{can}(t) \propto \int_0^t N(t-\tau)\left(E_1(\tau)+E_2(\tau)\right)d\tau\ ,\ t \geqslant 0 \tag{2.1}$$

$$N(t) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(t-t_0)^2/2\sigma^2}$$

$$E_i(t) = e^{-\lambda_i t}$$

$$SCR\left(t\right) = SN\left(t\right) \otimes SCRF_{can}$$

with estimated parameters: $\hat{t}_0 = 3.0745$ seconds for peak latency; $\hat{\sigma} = 0.7013$ for definition of the rise time; and $\hat{\lambda}_1 = 0.3176$ and $\hat{\lambda}_2 = 0.0708$ to define the two decay components. This function is normalised to its maximum such that an infinitely short SN impulse with unit amplitude elicits an SCR with unit amplitude. This renders parameter estimates from linear models easily interpretable in relation to conventional (peak-scoring) analysis.

Historical remarks:

(a) The appendix of [12] contains a typographic error in the definition of the canonical response function, in particular in the integration limits of eq. (2.1). However, in all versions of the software (PsPM 1.0-2.x) the canonical SCRF was constructed using the Matlab function `conv()` which defines vector convolution analogous to the integration operation defined above. This implementation has not changed since the initial formulation of the model.

(b) The initial formulation of the SCRF [3] used a Gaussian-smoothed Gamma distribution, based on a smaller dataset. In the extended data set published in [12], the polynomial part of the Gamma distribution was estimated to be one, resulting in a Gaussian smoothed exponential function.

Formulation for non-linear models An alternative formulation is provided for non-linear models which require a definition of the SCRF in the form of an ordinary differential equation (ODE). We use an inhomogenous third-order linear ODE

$$\dddot{x} + \vartheta_1 \ddot{x} + \vartheta_2 \dot{x} + \vartheta_3 x + u = 0 \tag{2.2}$$

$$u\left(t\right) = SN\left(t\right)$$

$$x = SCR\left(t\right)$$

where dot notation is used for time derivatives, and this formulation embeds convolution of the SCRF with the SN time series. Parameters were estimated from the empirical SCRF by using a Gaussian bump as canonical SN input (see section on SN below) as $\hat{\vartheta}_1 = 0.122505$, $\hat{\vartheta}_2 = 1.411425$, $\hat{\vartheta}_3 = 1.342052$. In this formulation, the delayed response (conduction latency) is not embedded in the SCRF but modelled separately as delay of the input with a parameter estimate of $\hat{\vartheta}_4 = 1.533879$ seconds. Amplitude of the response function is rescaled such that a canonical Gaussian bump SN impulse with unit amplitude elicits an SCR with unit amplitude.

Formulation for spontaneous fluctuations For modelling spontaneous fluctuations (SF), a different database was used that contained 1153 semi-automatically detected SF from 40 male participants [13] The first PCA component was taken as empirical SCRF which has slightly different shape than the empirical SCRF for evoked responses. A number of reasons may account for this difference, from participant selection over data pre-conditioning to differences in the canonical SN input. In the absence of further knowledge about the reason for this difference, we propose to use this empirical response function for SF. We estimated parameters for the non-linear model formulation $\hat{\vartheta}_1 = 0.9235$, $\hat{\vartheta}_2 = 3.9210$, $\hat{\vartheta}_3 = 2.1594$. The delay of the response is modelled explicity as delay of the SN input with parameter estimated from the model for event-related SCR, $\hat{\vartheta}_4 = 1.533879$ seconds. Amplitude of the response function is rescaled such that a canonical Gaussian bump SN impulse with unit amplitude elicits an SF with unit amplitude.

Recommendation For GLM, it is recommended to use a constrained SCRF by combining the canonical SCRF with time derivative and reconstruct the response after estimation of amplitude parameters [1]. For non-linear models, it is recommended to use the ODE formulation of the canonical SCRF [Staib et al., submitted]. The benefit of individual response functions in either scheme has not yet been tested.

2.3 Neural model

The neural model specifies how event-related or spontaneous sympathetic arousal elicits SN activity. There are three types of neural models. All models contain a latent error term ω which absorbs spontanous SN bursts and influences of SA unrelated to the experiment and is added to the experimentally induced SN time series, SN_{exp}:

$$SN\left(t\right) = SN_{exp} + \omega$$

2.3.1 General Linear Model (GLM) for evoked SA

This is the simplest neural model in which evoked SA causes an infinitely short central SN firing burst with zero latency after some event. This model is a good approximation to situations in which short external stimuli elicit SA. For each experimental event:

$$SN_{exp}(t) = \begin{cases} a & t = 0 \\ 0 & \text{otherwise} \end{cases}$$

$$a = SA$$

2.3.2 Non-linear model (DCM) for event-related SA

This model [14] assumes that event-related SA causes a Gaussian bump like SN firing burst after some delay and with a specified dispersion. For each experimental event:

$$SN_{exp}\left(t\right) = a \exp \frac{-\left(t-\mu\right)^2}{2\sigma^2} + \omega,\ 0 < \mu < \mu_{max},\ \sigma_{min} < \sigma < \sigma_{max}$$

$$a = SA > 0$$

For situations in which short external stimuli elicit SA, this model uses a canonical central burst latency of zero and dispersion of $\sigma_e = 0.3$ seconds. For situations in which central processes translate SA into SN with unknown parameters, latency and dispersion are estimated from the data, alongside the SA amplitude, using the following constraints: $\sigma_{min} = 0.1$, $\sigma_{max} = 0.5\mu_{max}$ where μ_{max} is the maximally allowed response latency.

2.3.3 Non-linear model for tonic SA (spontaneous fluctuations, SF)

Spontaneous SN bursts cause spontaneous fluctuations (SF) in skin conductance. The number of spontaneous SN bursts is thought to depend on SA. Each spontaneous SN burst is modelled as a Gaussian bump with unknown timing and amplitude and fixed dispersion. For each spontaneous SF burst:

$$SN_{SF} = a \exp \frac{-\left(t-\mu\right)^2}{2\sigma^2} + \omega,\ \sigma = \sigma_0,\ a > 0$$

The dispersion is set to $\sigma_0 = 0.3$ seconds.

2.4 Model inversion

For all models we only consider one error term: $\varepsilon' = \varepsilon + \omega \otimes SCRF$ which collapses observation error and latent error.

2.4.1 General linear model (GLM) inversion

In the GLM for SCR, only the amplitude of evoked SN is being estimated. Both latent error and observation error are lumped into one error term ε. This renders the model linear, and the estimation can be performed by simple matrix inversion. Essentially, predicted SCR functions are created per experimental condition by modelling events in an experimental condition as stick functions, and convolving them with the SCRF. The result then forms one column in a design matrix:

$$Y = X\beta + \varepsilon'$$

Here, β is the vector of amplitude parameters for each experimental condition which relates to the mean evoked SA in condition i: $\beta_i = SA_i$. Finally, each column of X is defined as

$$X_{ij} = SN_i \otimes SCRF_j$$

where SN_i is the expected time series of SN firing for an experimental condition i with unit amplitude, and j is the index of the SCRF basis set. Finally, X contains columns for the intercept. It is possible to concatenate data from different experimental sessions, and an intercept is modelled for each session. The model is inverted using a maximum-likelihood approach to estimating the Moore-Penrose pseudoinverse X^+ of X as implemented in the matlab function `pinv()`. This allows to deal with situations in which X is not of full rank.

$$\hat{\beta} = X^+ Y$$

2.4.2 Non-linear model (DCM) inversion for event-related SA

Models in which timing and/or dispersion of SN bursts need to be estimated are non-linear and are inverted using an efficient Variational Bayes algorithm [15]. The full model can be written as:

$$SCR = SCR_{exp} + SCR_{SF} + SCL + \varepsilon'$$

The SCR components are defined by eq. (2.2) with respective parameters and a respective input function $u(t)$. Specifically the experimental SCR component models evoked and event-related responses, and the SF component models spontaneous fluctuations between trials (part of the latent error). The SCL term is an explicitly modelled error term that absorbs baseline fluctuations and is defined as

$$SN_{SCL} = a \exp \frac{-(t-\mu)^2}{2\sigma^2}, \ \sigma = \sigma_0$$

where dispersion is set to $\sigma_0 = 1.0$ seconds, and μ is estimated from the data.

We propose the following limits for modelling SF and SCL between trials: SN bursts occur more than 5 s after the previous trial, and more than 2 s before the next trial. These limits ensure that the modelled SCL and SF do not absorb variance caused by the experiment. The use of different limits has not empirically been tested.

In this model, the parameter space is too large to invert it at once. Inversion is done in a trial-by-trial approach: a number of trials, $T_{1..n}$ is inverted. The ensuing parameter estimates for T_1 are extracted, and the parameter estimates for $T_{2..n}$ are used as prior values for the next inversion for trials $T_{2..n+1}$ where the estimated hidden state values at the start of trial 2 are taken as starting values of the hidden states x. This is continued until all trials have been estimated. This scheme ensures that inversion is kept tractable, and at the same time overlapping responses are taken into account. It also means that estimation errors accumulate over trials which is why part of the latent error is explicitly modelled as SF and SCL. At an inter trial interval of around 10 seconds, inverting 2 or 3 trials at the same time yields comparable predictive validity [Staib et al., submitted].

2.4.3 Non-linear model inversion for tonic SA (spontaneous fluctuations, SF)

DCM This model is inverted using a Variational Bayes algorithm [16, 15]. Because the number of SN bursts is difficult to estimate directly, the SF model inversion uses a fixed rate of SN bursts (set to $f = 0.5$ Hertz by default) the amplitude of which is estimated from the data. An amplitude threshold is defined to separate "true" SN bursts from noise. An amplitude treshold of 0.1 µS rendered the predictive highest validity in a validation paper [17]. The results from this paper have been replicated in an unpublished re-analysis using AIC as objective function in a manner similar to [12]. Re-analysing the second data set described in [17], the optimal threshold was between 0.15 - 0.20 µS. Further systematic research is needed to fine tune the optimal threshold.

MP This is a fast (<1 s per minute of data) approximation to the DCM inversion, by using a matching pursuit algorithm with the same amplitude threshold as the DCM inversion. In theory, this method is less precise, but empirically it is comparable to DCM results when analysing empirical data [Bach & Staib, submitted].

2.5 Data conditioning

2.5.1 Filtering

The currently implemented peripheral model requires high pass filtering of the data because it does not account for slow fluctuations of the skin conductance level which render the peripheral system infinite. In case of a signal with precisely known RF, the matched filter theorem provides a way of theoretically deriving a filter that maximises the signal-to-noise ratio. In our case, the true RF is not precisely known, and also varies between individuals, such that we empirically determined the filter characteristics that maximise predictive validity of SN estimates. A unidirectional 1st order Butterworth high pass filter with cut off frequency 0.05 Hz was optimal for GLM [12]. The design matrix of the GLM is filtered with the same frequeny to account for distortions of response shape. For the non-linear model (DCM) for event-related responses, an optimal filter frequeny was found at 0.0159 Hz when using a canonical SCRF and inversion of 2 trials at the same time [Staib et al., submitted]. Filter characteristics of SF models have not been investigated. In order to reduce computing load, data are resampled to 10 Hz sampling rate for all data analyses which requires a low-pass filter cut off frequeny of 5 Hz.

Historical remarks:

All models were developed using a high pass filter with cut off frequeny of 0.0159 Hz as commonly employed in convential analysis (bidirectional for phasic responses, unidirectional for spontaneous fluctuations). However, this turned out to not be the optimal filter for GLM in follow-up analyses of optimal data conditioning. Note that the canonical SCRF was developed from data filtered with the original settings, and this has not changed since.

2.5.2 Normalisation

For within-subjects analysis, we propose data normalisation after filtering to remove SCR scaling differences due to peripheral factors such as skin property. This increases predictive validity (unpublished analyses). For methods that yield trial-by-trial estimates, post-hoc normalisation of amplitude estimates before averaging within conditions yields even better predictive validity (Staib et al., in prepration). For between-subjects analysis, this is not an option as it removes between-subjects variance.

2.6 Implicit estimates of tonic sympathetic arousal

Our model for SF implies that

$$\int_I SCR\left(t\right) - SCL_0 dt \propto \sum_{i=1}^{n} a_i$$

where I is a time interval, and $a_{1..n}$ are the amplitudes of the n sudomotor bursts in this interval and SCL_0 is the skin conductance level at rest. That is, by simply taking the time integral, or area under the curve (AUC), of the baseline-corrected data, we get a measure of the number x amplitude of spontaneous SN bursts [13]. This measure is implemented in the software as well. However, we have subsequently shown that the number of spontaneous SN bursts is a better predictor of tonic SA than AUC, and hence we recommend to use the non-linear model for tonic SA.

2.7 Relation of SCR models to other approaches

Historically, three more approaches to model-based analysis of SCR have been introduced in the literature [18, 19, 20, 21] - one of them is implemented in the software package Ledalab. In this method, SN is calculated from SCR using a deterministic inverse filter. Peaks in the SN time series are then identified and used as operational index of SA. In a head-to-head comparison, predictive validity to identify states with known SA was compared for experiments involving short stimuli and evoked SA. PsPM had better predictive validity than Ledalab in 4 out of 5 tested contrasts, and equal validity in the fifth. Ledalab did not consistently perform superior to classical peak scoring analysis [2].

Part II
User Guide

3 Installation

You can find and download the PsPM package at the following URL:

http://pspm.sourceforge.net/

Decompress and copy the files to any directory of your choice. Then add the PsPM files to the Matlab search path by entering the following command in the Matlab command line:

`addpath([your Matlab path])` where [your Matlab path] is a string of the file location on your hard drive, e.g.

`addpath('C:\Program Files\MATLAB\R2013a\toolbox\PsPM\').`

Do not add subfolders of the PsPM directory to the matlab path.

Type `pspm` in the command line to start the package. Alternatively, each function can be called individually from the command line. In that case, you can use `scr_init` to load the settings, and `scr_quit` to remove PsPM subfunctions from the path and delete the settings.

The GUI requires Matlab 2007 (R14) or higher for the graphical interface. If you are using command line functions only, it should be possible to use earlier Matlab versions, but this has not been tested. Display of non-linear model inversion may not work properly under Matlab 2014 due to limitations of an external code.

4 Import

Related function: `scr_import`

Import external data files for use by PsPM. First, specify the data type. Then, other fields will come up as required for this data type. The imported data will be written to a new .mat file, prepended with 'scr_'.

Data Type

CED Spike

Text Text files can only contain numbers (i.e. no header lines with channel names) and one data column per channel. Make sure you use the decimal point (i.e. not decimal comma as used in some non-English speaking countries). At the moment, no import of event markers is possible

Matlab Each input file must contain a variable called data that is either a cell array of column vectors, or a data points $\times$ channels matrix. At the moment, no import of event markers is possible. Data structures containing timestamps cannot be imported at the moment; rather, PsPM will ask you for a sample rate.

Biopac AcqKnowledge (up to v. 3.9)

exported Biopac AcqKnowledge

Labchart exported (up to v. 7.1) Export data to matlab format (plugin for the LabChart software, available from www.adinstruments.com)

Labchart exported (v. 7.2 and higher)

VarioPort

Biograph Infiniti exported Export data to text format, both "Export Channel Data" and "Export Interval Data" are supported; a header is required

Mindmedia Biotrace exported

BrainVision

Windaq (wdq) Requires an ActiveX Plugin provided by the manufacturer and contained in the subfolder Import/wdq for your convencience. This plugin only runs under 32 bit Matlab on Windows.

Observer XT compatible

NeuroScan

BioSemi

Eyelink Imports files which have been converted to ascii format (.asc files).

Data File(s)

Specify which data file(s) to import. You can import multiple files that all have the same structure.

New Channel

Define each channel that you want to import.

Channel Type

Specify the type of data in this channel. Currently supported data types: SCR, ECG, heart beat markers, interpolated heart rate, interpolated heart period, respiration traces, interpolated respiration period, pupil size, marker channels.

Channel /Column Number Specify where in the original file to find the data - depending on your data type this will be a channel or column number. You can either specify a number (i. e. the n-th channel in the file), or (for many data types) search for this channel by its name. Note: the channel number refers to the n-th recorded channel, not to its number during acquisition: if you did not save all recorded channels, these might be different for some data types.

- Search [if available]: Search for channel by its name – this only works if the channel names are unambiguous, and not for all data types.
- Specify Channel Number [if required, otherwise a default channel number will be assigned]: Specify the n-th channel. This counts the number of channels actually recorded.

Sample Rate [if required] Sample rate in Hz (i. e. samples per second). For most data types, this is read from the original data file.

Transfer Function [for SCR data only] Enter the conversion from recorded data to Microsiemens.

- File: Enter the name of the .mat file that contains the transfer function constants. This file needs to contain the following variables: 'c' is the transfer constant: data = c * (total conductance in mcS); 'Rs' is the series resistance in Ohm (usually 0), and 'offset' any offset in the data (usually 0).
- Input: Enter the transfer constants manually.

The relation between measured conductance G to recorded voltage U is given by

$$G = \frac{1}{\frac{c}{U - offset} - R}$$

1. Transfer Constant c: Constant by which the measured conductance is multiplied to give the recorded signal (and by which the signal needs to be divided to give the original conductance).
2. Offset: Fixed offset (signal at 0 conductance).
3. Series Resistor R: Resistance of any resistors in series with the subject.

- None: No transfer function. Use this only if you are not interested in absolute values, and if the recording settings were the same for all subjects.

5 Trim

Related function: `scr_trim`

Trim away unnessecary data, for example before an experiment started, or after it ended. Trim points can be defined in seconds with respect to start of the data file, in seconds with respect to first and last marker (if markers exist), or in seconds with respect to a user-define marker. The resulting data will be written to a new file, prepended with 't'.

Data file

Choose the data file you want to trim.

Reference

Choose your reference for trimming: file start, first/last marker, or a user-defined marker. All trimming is defined in seconds after this reference – choose negative values if you want to trim before the reference.

File

Trim from xx seconds after file start to xx seconds after file start.

- From (seconds after file start): Choose a positive value.
- Trim To (seconds after file start): Choose a positive value larger than the 'from' value.

First/Last Marker

Trim from xx seconds after first marker to xx seconds after last marker.

- From: (seconds after first marker): Choose a positive (after first marker) or negative (before first marker) value.
- To: (seconds after last marker): Choose a positive (after last marker) or negative (before last marker) value.
- Marker Channel: If you have more than one marker channel, choose the reference marker channel (default: use the fist marker channel in the file)

  - Default (first marker channel=0)

  - Number

Any Marker

Trim from xx seconds after any marker of your choice to xx seconds after any marker of your choice.

- From: Choose marker number and trimming point in seconds after this marker.

  - Marker Number x: Choose an integer value.

  - Seconds after Marker x: Choose a positive (after this marker) or negative (before this marker) value.

- To: Choose marker number and trimming point in seconds after this marker.

  - Marker Number y: Choose an integer value.

  - Seconds after Marker y: Choose a positive (after this marker) or negative (before this marker) value.

- Marker Channel: If you have more than one marker channel, choose the reference marker channel (default: use the fist marker channel in the file)

  - Default (first marker channel=0)

  - Number

Overwrite Existing File

Choose "yes" if you want to overwrite existing file(s) with the same name. Default: no.

6 GLM for SCR

Related function: `scr_glm`

General linear convolution models (GLM) are powerful for analysing evoked responses that follow an event with (approximately) fixed latency. This is similar to standard analysis of fMRI data. The user specifies events for different conditions. These are used to estimate the mean response amplitude per condition. These mean amplitudes can later be compared, using the contrast manager. References: [3, 4, 12][2].

Model Filename & output directory

Specify file name and directory where the *.mat file with the resulting model will be written.

SCR Channel

Indicate the channel containing the SCR data. By default the first SCR channel is assumed to contain the data for this model. If the first SCR channel does not contain the data for this model (e. g. there are two SCR channels), indicate the the channel number (within the SCR file) that contains the data for this model.

Time Units

Indicate the time units on which the specification of the conditions will be based. Time units can be specified in ‘seconds’, number of ‘markers’, or number of data ‘samples’ . Time units refer to the beginning of the data file and not to the beginning of the original recordings e. g. if data were trimmed.

Option for marker channels:

- Marker Channel: Indicate the marker channel. By default the first marker channel is assumed to contain the relevant markers. Markers are only used if you have specified the time units as ‘markers’.

Data & Design

Add the required number of data files or sessions here. These will be concatenated for analysis.

Data File

Specify the data file containing the SCR data (and potential marker information). If you have trimmed your data, add the file containing the trimmed data.

Missing Epochs

Indicate epochs in your data in which the SCR signal is missing or corrupted (e.g., due to artifacts).

- No Missing Epochs: The whole time series will be analyzed.

- Define Missing Epochs: Define the start and end points of the missing epochs either as epoch files or manually. Missing epochs will be excluded from the design matrix. Start and end points have to be defined in seconds starting from the beginning of the session.

  - Missing Epoch File: Indicate an epoch file specifying the start and end points of missing epochs. The mat file has to contain a variable ‘epochs’, which is an m x 2 array, where m is the number of missing epochs. The first column marks the start points of the epochs that are excluded from the analysis and the second column the end points.

- Enter Missing Epochs Manually (m: nr. of epochs): Enter the start and end points of missing epochs manually. Specify an m x 2 array, where m is the number of missing epochs. The first column marks the start points of the epochs that are excluded from the analysis and the second column the end points.

Design

Specify the timing of the events within the design matrix. Timing can be specified in 'seconds', 'markers' or 'samples' with respect to the beginning of the data file. See 'Time Units' settings. Conditions can be specified manually or by using multiple condition files (i.e., an SPM-style mat file).

- Condition File: Create a file with the following variables:
  - names: a cell array of string for the names of the experimental conditions
  - onsets: a cell array of number vectors for the onsets of events for each experimental condition, expressed in seconds, marker numbers, or samples, as specified in timeunits
  - durations (optional, default 0): a cell array of vectors for the duration of each event. You need to use 'seconds' or 'samples' as time units
  - pmod: this is used to specify regressors that specify how responses in an experimental condition depend on a parameter to model the effect e.g. of habituation, reaction times, or stimulus ratings. pmod is a struct array corresponding to names and onsets and containing the fields
    - name: cell array of names for each parametric modulator for this condition
    - param: cell array of vectors for each parameter for this condition, containing as many numbers as there are onsets
    - poly (optional, default 1): specifies the polynomial degree
  - e.g. produce a simple multiple condition file by typing: `names = {'condition a', 'condition b'}; onsets = {[1 2 3], [4 5 6]}; save('testfile', 'names', 'onsets');`
- Enter conditions manually:

  - Name: Specify the name of the condition.

  - Onsets: Specify a vector of onsets. The length of the vector corresponds to the number of events included in this condition. Onsets have to be indicated in the specified time unit ('seconds', 'markers', or 'samples').

  - Durations: Typically, a duration of 0 is used to model an event onset. If all events included in this condition have the same length specify just a single number. If events have different durations, specify a vector with the same length as the vector for onsets.

- Parametric Modulator(s) [optional] (n: nr. of pmods): If you want to include a parametric modulator, specify a vector with the same length as the vector for onsets. For example, parametric modulators can model reaction times, ratings of stimuli, or habituation effects over time. For each parametric modulator a new regressor is included in the design matrix. The normalized parameters are multiplied with the respective onset regressors.

1. Name: Specify the name of the parametric modulator.
2. Polynomial Degree: Specify an exponent that is applied to the parametric modulator. A value of 1 leaves the parametric modulator unchanged and thus corresponds to a linear change over the values of the parametric modulator (first-order). Higher order modulation introduces further columns that contain the non-linear parametric modulators [e.g., second-order: (squared), third-order (cubed), etc].
3. Parameter Values: Specify a vector with the same length as the vector for onsets.

Nuisance File (optional)

You can include nuisance parameters such as motion parameters in the design matrix. Nuisance parameters are not convolved with the canonical response function. The file has to be either a .txt file containing the regressors in columns, or a .mat file containing the regressors in a matrix variable called R. There must be as many values for each column of R as there are data values in your data file. PsPM will call the regressors pertaining to the different columns R1, R2, ...

Basis Function

Basis functions for the peripheral model. Standard is to use a canonical skin conductance response function (SCRF) with time derivative for later reconstruction of the response peak.

SCRF0

SCRF without derivatives.

SCRF1

SCRF with time derivative (default). This is the recommended option based on the publication [2].

SCRF2

SCRF with time and dispersion derivative.

FIR

Uninformed finite impulse response (FIR) model: specify the number and duration of time bins to be estimated.

- Arguments
  - N: Number of Time Bins: Number of time bins.
  - D: Duration of Time Bins: Duration of time bins (in seconds).

Normalize

Specify if you want to z-normalize the SCR data for each subject. For within-subjects designs, this is highly recommended, but for between-subjects designs it needs to be set to false.

Filter Settings

Specify how you want filter the SCR data.

Default

Standard settings for the Butterworth bandpass filter. These are the optimal settings from [2]: unidirectional Butterworth band pass filter with cut off frequencies of 0.05 and 5 Hz, downsampling to 5 Hz.

Edit Settings

Create your own filter settings (not encouraged).

- Low-Pass Filter
  - Enable
    1. Cutoff Frequency: Specify the low-pass filter cutoff in Hz.
    2. Filter Order: Specify the low-pass filter order.
  - Disable
- High-Pass Filter
  - Enable
    1. Cutoff Frequency: Specify the high-pass filter cutoff in Hz.
    2. Filter Order: Specify the high-pass filter order.
  - Disable

- New Sampling Rate: Specify the sampling rate in Hz to down sample SCR data. Enter NaN to leave the sampling rate unchanged.
- Filter Direction {uni, bi}, (default: uni): A unidirectional filter is applied twice in the forward direction. A 'bidirectional' filter is applied once in the forward direction and once in the backward direction to correct the temporal shift due to filtering in forward direction.

Overwrite Existing File

Specify whether you want to overwrite existing mat files.

7 Non-Linear Model for SCR

Related function: `scr_dcm`

Non-linear models for SCR are powerful if response timing is not precisely known and has to be estimated. A typical example are anticipatory SCR in fear conditioning – they must occur at some point within a time-window of several seconds duration, but that time point may vary over trials. Dynamic causal modelling (DCM) is the framework for parameter estimation. PsPM implements an iterative trial-by-trial algorithm. Different from GLM, response parameters are estimated per trial, not per condition, and the algorithm must not be informed about the condition. Trial-by-trial response parameters can later be summarized across trials, and compared between conditions, using the contrast manager. References: [14], [Staib et al., submitted]

Model Filename & Output directory

Specify file name for the resulting model. Specify directory where the mat file with the resulting model will be written.

SCR channel

Indicate the channel containing the SCR data. By default the first SCR channel is assumed to contain the data for this model. If the first SCR channel does not contain the data for this model (e. g. there are two SCR channels), indicate the channel number (within the SCR file) that contains the data for this model.

Data & design

Add the appropriate number of data files or sessions here. Results will be concatenated.

Data File

Add the data file containing the SCR data. If you have trimmed your data, add the file containing the trimmed data.

Design

Specify the timings of individual events from your design either by creating a timing file or by entering the timings manually. The DCM framework allows you to specify two types of events:

Fixed latency (evoked): An event is assumed to elicit an immediate response. The amplitude of the sympathetic arousal will be estimated, while the latency and duration of the response are fixed. This event type is meant to model evoked responses.

Flexible latency and duration (event-related): An event is assumed to elicit sympathetic arousal within a known response window, but with unknown amplitude, latency and duration. For each event of this type, specify a time window in which the response is expected. PsPM will then estimate the amplitude, duration and latency of the response. An example for this type of event is an anticipatory response which might vary in timing between trials (e. g. in fear conditioning).

- Timing File: The timing file has to be a .mat file containing a cell array called 'events'. For a design with n trials and m events per trial (n × m events in total), the cell array has to be structured in the following way: Create m cells in the cell array (one per event type that occurs in each trial). Each cell defines either a fixed or a flexible event type. A cell that defines a fixed event type has to contain a vector with n entries, i.e. one time point per trial, in seconds, samples or markers. A cell that defines a flexible type has to contain an array with n rows and two columns. The first column specifies the onsets of the time windows for each trial, while the second column specifies the offsets of the time windows. It is assumed that all trials have the same structure, i.e. the same number of fixed and flexible event types. For individual trials with a different structure you can enter negative values as time information to omit estimation of a response. For later comparison between trials of different conditions, it is absolutely mandatory that they contain the same types of events, to avoid bias. Hence, if one condition omits an event (e. g. unreinforced trials in conditioning experiments), the omitted event needs to be modelled as well.

- Enter Timing Manually: In the DCM framework, a session is structured in n individual trials. Each trial contains m fixed and/or flexible event types. All trials need to have the same structure. Create event types to define the structure of a trial and enter the timings for all events.

  - Name (optional): Enter a name of the event. This name can later be used for display and export.

  - Onsets: For events with a fixed response, specify a vector of onsets. The length of the vector corresponds to the number of trials (n). For events with a flexible response, specify a two column array. The first column defines the onset of the time window in which the response occurs. The second column defines the offset. The number of rows of the array corresponds to the number of trials (n). All timings are specified in seconds.

Condition names (optional)

Specify the conditions that the individual trials belong to. This information is not used for the parameter estimation, but it allows you to later access the conditions in the contrast manager.

- Name: Specify the name of the condition.
- Index: Specify a vector of trial indices between 1 and n. The length of the vector corresponds to the number of events included in this condition.

Data Options

Normalization

Specify if you want to normalize the SCR data for each subject. For within-subjects designs, this is highly recommended. During analysis, data are always z-transformed, but the parameter estimats are transformed back if this is set to false.

Filter Settings

Specify if you want filter the SCR data.

- Default: Standard settings for the Butterworth bandpass filter. These are the optimal settings from [Staib et al., submitted]: bidirectional Butterworth band pass filter with cutoff frequencies of 0.0159 Hz and 5 Hz, downsampling to 10 Hz.
- Edit Settings: Create your own filter (discouraged).

  - Low-Pass Filter

  1. Enable
     - Cutoff Frequency: Specify the low-pass filter cutoff in Hz.
     - Filter Order: Specify the low-pass filter order.
  2. Disable

  - High-Pass Filter

  1. Enable
     - Cutoff Frequency: Specify the high-pass filter cutoff in Hz.
     - Filter Order: Specify the high-pass filter order.
  2. Disable

  -New Sampling Rate: Specify the sampling rate in Hz to down sample SCR data. Enter NaN to leave the sampling rate unchanged.

  -Filter Direction {uni, bi}, (default: bi): A unidirectional filter is applied twice in the forward direction. A ‘bidirectional’ filter is applied once in the forward direction and once in the backward direction to correct the temporal shift due to filtering in forward direction.

Response Function Options

Estimate the Response Function from The Data

A response function can be estimated from the data and used instead of the canonical response function (default 0 for fully flexible and 1 for fix and flex/fix paradigms). This is not normally recommended unless you have long inter trial intervals in the range of 20-30 s [Staib et al., submitted].

Only Estimate RF (Do Not Do Trial-Wise DCM)

This option can be used to estimate an individual response function to be used in analysis of another experiment.

Call External File to Provide Response Function

Call an external file to provide a response function, which was previously estimated using the option "only estimate RF".

Inversion Options

Number of Trials to Invert at The Same Time (default: 2 s)

The iterative DCM algorithm accounts for response overlap by cosidering several trials at a time. This can be set here.

SF-Free Window Before First Event [s] (default: 2 s)

The DCM algorithm automatically models spontaneous fluctuations in inter trial intervals. Here, you can define a time window before the first event in every trial in which no spontaneous fluctuation will be estimated.

SF-Free Window After Last Event [s] (default: 5 s)

The DCM algorithm automatically models spontaneous fluctuations in inter trial intervals. Here, you can define a time window after the last event in every trial in which no spontaneous fluctuation will be estimated.

Maximum Frequency of SF in ITIs [Hz] (default: 0.5 Hz)

The DCM algorithm automatically models spontaneous fluctuations in inter trial intervals. Here you can define the minimal delay between two subsequent spontaneous fluctuations.

SCL-Change-Free Window Before First Event [s] (default: 2 s)

The DCM algorithm automatically models baseline drifts in inter trial intervals. Here, you can define a window before the first event in every trial in which no change of the skin conductance level will be assumed.

SCL-Change-Free Window After Last Event [s] (default: 5 s)

The DCM algorithm automatically models baseline drifts in inter trial intervals. Here, you can define a window after the last event in every trial in which no change of the skin conductance level will be assumed.

Display Options

Display Progress Window

Show a on-line diagnostic plot for each iteration of the estimation process.

Display Intermediate Windows

Show small plots displaying the progress of each iteration in the estimation process.

8 SF models

Related function: `scr_sf`

This suite of models is designed for analysing spontaneous fluctuations (SF) in skin conductance as a marker for tonic arousal. SF are analysed over time windows that typically last 60 s and should at least be 15 s long. PsPM implements 3 different models: (1) Skin conductance level (SCL): this is the mean signal over the epoch (2) Area under the curve (AUC): this is the time-integral of the above-minimum signal, divided by epoch duration. This is designed to be independent from SCL and ideally represents the number x amplitude of SF in this time window. (3) Number of SF estimated by DCM (Dynamic Causal Modelling) or MP (Matching Pursuit): this is a non-linear estimation of the number of SF, and is the most sensitive indicator of tonic arousal. It relies on absolute data values as it implements and absolute threshold for data peaks. References: [17], [Bach & Staib, Psychophysiology, in press]

Data File

Add the data file containing the SCR data (and potential marker information). If you have trimmed your data, add the file containing the trimmed data.

Model Filename & Output directory

Specify file name for the resulting model. Specify directory where the mat file with the resulting model will be written.

Method

Choose the method for estimating tonic sympathetic arousal: AUC (equivalent to number x amplitude of spontaneous fluctuations), SCL (tonic skin conductance level), DCM or MP. The latter two estimate the number of spontaneous fluctuations, requiring absolute data units as both methods implement an absolute amplitude threshold. In theory, DCM provides highest sensitivity but is slow[17]. MP is a very fast approximation to the DCM results, and comparable in sensitivity for analysis of empirical data [Bach & Staib, Psychophysiology, in press].

Time Units

Indicate the time units on which the specification of the conditions will be based. Time units can be specified in 'seconds', number of 'markers', or number of data 'samples' . Time units refer to the beginning of the data file and not to the beginning of the original recordings e. g. if data were trimmed. Choose 'seconds', 'markers', 'samples', or 'whole' to analyse the entire data file as one epoch.

Seconds/Markers/Samples

- Epochs - define data epochs to analyse
  - Epoch File
  - Enter Epochs Manually

Markers

- Marker Channel (default: 0): Indicate the marker channel. By default the first marker channel is assumed to contain the relevant markers. Markers are only used if you have specified the time units as 'markers'.

Filter Settings

Specify if you want filter the SCR data.

Default

Standard settings for the Butterworth bandpass filter: unidirectional Butterworth bandpass filter with cut off frequencies of 0.0159 Hz and 5 Hz, down sampling to 10 Hz.

Edit Settings

Create your own filter (discouraged).

- Low-Pass Filter
  - Enable
    1. Cutoff Frequency: Specify the low-pass filter cutoff in Hz.
    2. Filter Order: Specify the low-pass filter order.
  - Disable
- High-Pass Filter
  - Enable
    1. Cutoff Frequency: Specify the high-pass filter cutoff in Hz.
    2. Filter Order: Specify the high-pass filter order.
  - Disable
- New Sampling Rate: Specify the sampling rate in Hz to down sample SCR data. Enter NaN to leave the sampling rate unchanged.
- Filter Direction): A unidirectional filter is applied twice in the forward direction. A 'bidirectional' filter is applied once in the forward direction and once in the backward direction to correct the temporal shift due to filtering in forward direction.

Channel

Indicate the channel containing the SCR data. By default the first SCR channel is assumed to contain the data for this model. If the first SCR channel does not contain the data for this model (e. g. there are two SCR channels), indicate the channel number (within the SCR file) that contains the data for this model.

Overwrite Existing File

Specify whether you want to overwrite existing mat file.

Threshold (used for DCM and MP only)

Threshold for SN detection - default 0.1 µS.

Display Progress Window (used for DCM and MP only)

Show a on-line diagnostic plot for each iteration of the estimation process (DCM) or for the result of the estimation process (MP).

Display Intermediate Windows (used for DCM only)

Show small plots displaying the progress of each iteration in the estimation process.

9 Review First-Level Model (via Batch editor)

Related function: `scr_review`

This module allows you to look at the first-level (within-subject) model to investigate model fit and potential estimation problems. This is not necessary for standard analyses. Further processing can be performed directly on the second level after first-level model estimation.

Model File

Choose model file to review.

Model Type

Specify the type of model.

GLM

Specify the plot that you wish to display:

- Design matrix
- Orthogonality
- Predicted & Observed
- Regressor names
- Reconstructed

DCM

- Inversion results for one trial: Non-linear SCR model (DCM) - Review individual trials or sequences of trials inverted at the same time.

  - Session Number (Default 1): Data session. Must be 1 if there is only one session in the file.

  - Trial Number: Trial to review.

- Predicted & Observed for all trials: DCM for event-related responses - Review summary plot of all trials.

  - Session Number

  - Figure Name [optional]: Adding a figure name saves the figure as .fig file.

- SCRF: DCM for event-related responses - review SCRF (useful if SCRF was estimated from the data).

SF

DCM for spontaneous fluctuations: Show inversion results for one episode

- Episode Number: Episode to review.

Contrasts

Display contrast names for any first level model.

10 First-Level Model Review Manager (via GUI)

Related function: `scr_review`

Display information stored in a model file through an easily accessable interface. Add a model file to the menu via *Add Models*. Available content of a highlighted model is listed on the right hand side under *Figure Selection*. For non-linear models, you can select the session number by entering its index in the field *Session number*; GLMs are inverted and displayed across all sessions. Click on any *Display* or *Show* button to visualize information. Details for each model type are listed in Chapter 9.

11 First-Level Contrasts (via Batch editor)

Related function: `scr_con1`

Define within-subject contrasts here for testing on the second level. Contrasts can be between conditions (GLM), epochs (SF) or trials (Non-linear SCR models). Contrast weights should add up to 0 (for testing differences between conditions/epochs/trials) or to 1 (for testing global/intercept effects across conditions/epochs/trials). Example: an experiment realises a 2 (Factor 1: a, A) x 2 (Factor 2: b, B) factorial design and consists of 4 conditions: aa, aB, Ab, AB. Testing the following contrasts is equivalent to a full ANOVA model: Main effect factor 1: [1 1 -1 -1], Main effect factor 2: [1 -1 1 -1], Interaction factor 1 x factor 2: [1 -1 -1 1]. To test condition effects in non-linear models, assign the same contrast weight to all trials from the same condition.

Model File(s)

Specify the model file for which to compute contrasts.

Specify Contrasts on Datatype

Contrasts are usually specified on parameter estimates. For GLM, you can also choose to specify them on conditions or reconstructed responses per condition. In this case, your contrast vector needs to take into account only the first basis function. For DCM, you can specify contrasts based on conditions as well. This will average within conditions. Use the review manager to extract condition names and their order. This argument will be ignored for other first-level models.

- Parameter: Use all parameter estimates.
- Condition: Contrasts formulated in terms of conditions in a GLM, automatically detects number of basis functions and uses only the first one (i.e. without derivatives), or based on assignment of trials to conditions in DCM.
- Reconstructed: Contrasts formulated in terms of conditions in a GLM, reconstructs estimated response from all basis functions and uses the peak of the estimated response.

Contrast(s)

Contrast Name

This name identifies the contrast in tables and displays.

Contrast Vector

This is a vector on all included conditions (GLM), trials (DCM for event-related responses), or epochs (DCM for SF). Shorter vectors will be appended with zeros. To specify a condition or trial difference, the contrast weights must add up to zero (e. g. was sympathetic arousal in condition A larger than in condition B: c = [1 -1]). To specify a summary of conditions or trials, the contrast weights should add up to 1 to retain proper scaling (e. g. was non-zero sympathetic arousal elicited in combined conditions A and B: c = [0.5 0.5]) .

Delete Existing Contrasts

Deletes existing contrasts from the model file.

12 First level contrast manager (via GUI)

Related function: `scr_con1`

The Manager guides you through the setup of pre-defined within-subject contrasts applied to the parameter estimates of your model.

Load Model

Import a model file from a subject containing parameter estimates for which to compute contrasts.

Define Contrast Name

Enter a name to identify a contrast. By pressing *New Contrast*, the contrast is added to the box *Contrasts*. You can reset existing contrasts by highlighting a contrast and pressing *Clear Contrast* or you can remove contrasts from your model file by pressing *Delete Contrast.*

Define Stats Type

Depending on the model (GLM, spontaneous fluctuation or non-linear model), different data types are available (see Chapter 11 for details). Choose the type you want to be entered into a statistical test.

Define Test

Three pre-defined types of statstics can be computed on your estimates. After choosing the test, you can add or remove conditions/trials to a group by clicking on them in the *Names* window. The color will indicate to which group they belong.

- Test of intercept: Compute the average of conditions entered in *Group 1*. This option is suitable if you are interested in the main effect of a condition.
- Test of condition differences: Compute the difference between the averages of *Group 1* and *Group 2*.
- Test of quadratic effects: Here you compute the combined effect of *Group 1* and *Group 3* against *Group 2*. Choose this option if you have three experimental conditions.

Run

Execute computation of statistics. The contrasts are stored in your model files and can be reviewed any time using the Review Manager (Chapter 10).

13 Export Statistics

Related function: `scr_exp`

Export first level statistics to a file for further analysis in statistical software, or to the screen.

Model File(s)

Specify file from which to export statistics.

Stats type

Normally, all parameter estimates are exported. For GLM, you can choose to only export the first basis function per condition, or the reconstructed response per condition. For DCM, you can specify contrasts based on conditions as well. This will average within conditions. This argument cannot be used for other models.

- Parameter: Export all parameter estimates.

- Condition: Export all conditions in a GLM, automatically detects number of basis functions and export only the first one (i.e. without derivatives), or export condition averages in DCM.

- Reconstructed: Export all conditions in a GLM, reconstructs estimated response from all basis functions and export the peak of the estimated response.

Target

Export to screen or to file?

Delimiter for Output File

Select a delimiter for the output file. Default is 'tab', you can select from a choice of several options or specify as free text.

14 Define Second-Level Model

Related function: `scr_con2`

Define one-sample and two-sample t-tests on the between-subject (second) level. A one-sample t-test is normally used to test within-subject contrasts and is equivalent to t-contrasts in an ANOVA model A two-sample t-test is required for between-subjects contrasts or interactions. This module sets up the model but does not report results.

Test Type

Specify the test type.

One Sample T-Test

- Model File(s)

Two Sample T-Test

- Model File(s) 1: Model files for group 1.
- Model File(s) 2: Model files for group 2.

Output directory

Specify directory where the mat file with the resulting model will be written.

Filename for Output Model

Specify name for the resulting model.

Define Contrastss

Define which contrasts to use from the model files, and the second level contrast names. Names can be read from the first model file, or just be numbered.

Read from First Model File

- All Contrasts
- Contrast Vector: Index of first-level contrasts to use.

Number Contrasts

- All Contrasts
- Contrast Vector: Index of first-level contrasts to use.

Overwrite Existing File

Specify whether you want to overwrite existing mat file.

15 Report Second-Level Results

Related function: `scr_rev2`
Result reporting for second level model.

Model File

Specify 2nd level model file.

Contrast Vector (only accessible from Batch Editor)

Index of contrasts to be reported (optional).

16 Display Data

Related function: `scr_display`
Display PsPM data file in a new figure.

Data File

Specify data file to display.

17 Rename File

Related function: `scr_ren`

Rename PsPM data file. This renames the file and updates the file information.

File

Choose how many files to rename.

File Name

Choose name of original file.

New File Name

Choose new file name.

18 Split Sessions

Related function: `scr_split_sessions`

Split sessions, defined by trains of markers. This function is most commonly used to split fMRI sessions when a (slice or volume) pulse from the MRI scanner has been recorded. The function will identify trains of markers and detect breaks in these marker sequences. The individual sessions will be written to new files with a suffix '_sn', and the session number. You can choose a marker channel if several were recorded.

Data File

Choose the data file, in which you want to split sessions.

Marker Channel

If you have more than one marker channel, choose the marker channel used for splitting sessions (default: use first marker channel).

- Default
- Number

Overwrite Existing File

Choose "yes" if you want to overwrite existing file(s) with the same name.

19 Artefact Removal

Related function: `scr_pp`

This module offers a few basic artefact removal functions. Currently, a median filter and a butterworth low pass filter are implemented. The median filter is useful to remove short "spikes" in the data, for example from gradient switching in MRI. The Butterworth filter can be used to get rid of high frequency noise that is not sufficiently filtered away by the filters implemented on-the-fly during first level modelling.

Data File

Choose the data file.

Channel Number

Select the channel to work on.

Filter Type

Currently, median and butterworth filters are implemented. A median filter is recommended for short spikes, generated for example in MRI scanners by gradient switching. A butterworth filter is applied in most models; check there to see whether an additional filtering is meaningful.

Median Filter

- Number of Time Points: Number of time points over which the median is taken.

Butterworth Low pass Filter

- Cutoff Frequency: Cutoff frequency has to be at least 20Hz.

Overwrite Existing File

Choose “yes” if you want to overwrite existing file(s) with the same name.

20 Downsample Data

Related function: `scr_down`

This function downsamples individual channels in an PsPM file to a required sampling rate, after applying an anti-aliasing Butterworth filter at the Nyquist frequency. The resulting data will be written to a new .mat file, prependend with 'd', and will contain all channels – also the ones that were not downsampled.

Data File

Name of the data files to be downsampled.

New Frequency

Required sampling frequency.

Channels To Downsample

Vector of channels to downsample, or all channels.

Overwrite Existing File

Choose “yes” if you want to overwrite existing file(s) with the same name.

21 Convert ECG to Heart Beat

Related function: `scr_ecg2hb`
Detect QRS complexes in ECG data and write timestamps of detected R spikes into a new heart beat channel. This function uses an algorithm by Pan & Tompkins adapted from [22]. Hidden options for minimum and maximum heart rate become visible when the job is saved as a script and should only be used if the algorithm fails.

Data File

Specify data file. The detected heart beat data will be written to a new channel in this file.

Channel

Number of heart beat channel (default: first heart beat channel).

22 Convert Heart Beat to Heart Period

Related function: `scr_hb2hp`
Interpolate heart beat time stamps into continuous heart period data and write into new channel. This function uses heart period rather than heart rate because heart period varies linearly with ANS input into the heart.

Data File

Specify data file. The interpolated heart period data will be written to a new channel in this file.

Sample Rate

Sample rate for heart period channel. Default: 10 Hz.

Channel

Number of heart beat channel (default: first heart beat channel).

23 Convert Respiration to Respiration Period

Related function: `scr_resp2rp`

Data File

Specify data file. Specify data file. The interpolated respiration period data will be written to a new channel in this file.

Sample Rate

Sample rate for respiration period channel. Default: 10 Hz.

Channel

Number of respiration channel (default: first heart beat channel).

24 Troubleshooting and known problems

24.1 Path

If the PsPM folder is added to the path with all subfolders, the batch editor will not work. Only put the PsPM main folder (containing the function pspm.m) to the path.

24.2 Diagnostic graphics for non-linear model inversion

During non-linear model inversion, a diagnostic window is displayed for each trial. Between two trials, there is a short period of time during which several tabs appear in that window. If you click on one of the hidden tabs, model inversion will fail due to a display error.

Diagnostic plots do not work in Matlab versions 2014 and upwards. This is due to a limitation of an external software. We are working on a fix.

24.3 Export statistics into text format with tab delimiter

The tab delimiter in the exported files is not correctly interpreted by text editors like MS Word. Nonetheless, the data can be imported into softwares such as Excel or SPSS.

Part III
Tutorial

Contributed by Christoph Korn & Matthias Staib.

25 Appraisal data

Here, we analyze SCR data using a general linear model (GLM). The example data set comprises SCR data from 15 participants and can be downloaded from pspm.sourceforge.net (see: sample data). Results from this data set have been previously published [12, 2]. Each participant saw 45 neutral and 45 aversive pictures within one session that included two short breaks. SCR data were recorded using a 0.5 V coupler, optical (wave to pulse) transducer, and CED Spike, with a minimum sampling rate of 100 Hz.

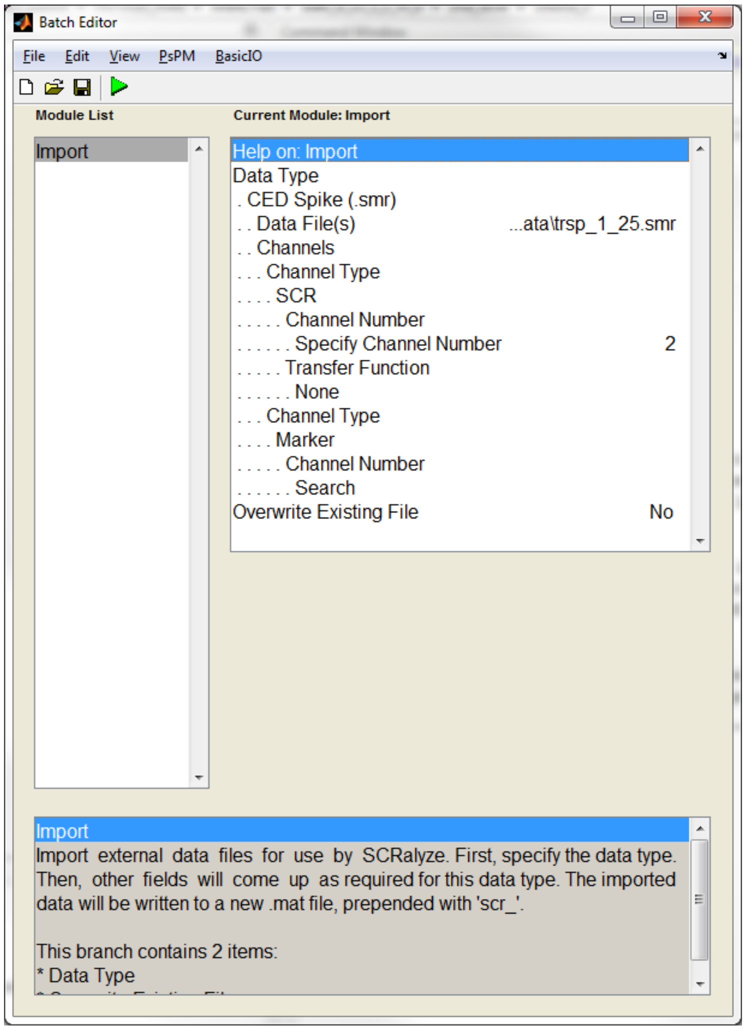
25.1 Import

We import the SCR data and the corresponding marker data. We first import data from one subject.

- In the batch editor go to *PsPM* → *Data Preparation* → *Import*.
- Specify the Data Type. Several data formats are available. The current data set is in the smr format.
- Select the SCR Data File of the first subject `trsp_1_25.smr`.
- Specify the *Channels*. We need one SCR channel and one marker channel.
  - The *Channel Number* of the SCR channel is 2.
  - For the *Transfer Function* choose *None*.
  - For the *Channel Number* of the marker channel put in *Search* to test the automatic search (otherwise you can specify the number of the marker channel manually. Here, the marker channel is 0).

See Figure 25.1 for a picture of the final batch editor. We recommend saving the batch and script so that you can easily re-run your analyses at a later time point (see Chapter 27). Run the batch. A mat file with the name `scr_trsp_1_25.mat` will be created in the same folder as the original data file.

You can have a look at the imported data by loading this file into MATLAB. The cell array data contains two structures. The first one contains the SCR data and the second one the marker data (this data set contains 96 markers).



```
Help on: Import
Data Type
. CED Spike (.smr)
. . Data File(s)                         ...ata\trsp_1_25.smr
. . Channels
. . . Channel Type
. . . . SCR
. . . . . Channel Number
. . . . . . Specify Channel Number                      2
. . . . . Transfer Function
. . . . . . None
. . . Channel Type
. . . . Marker
. . . . . Channel Number
. . . . . . Search
Overwrite Existing File                                No
```

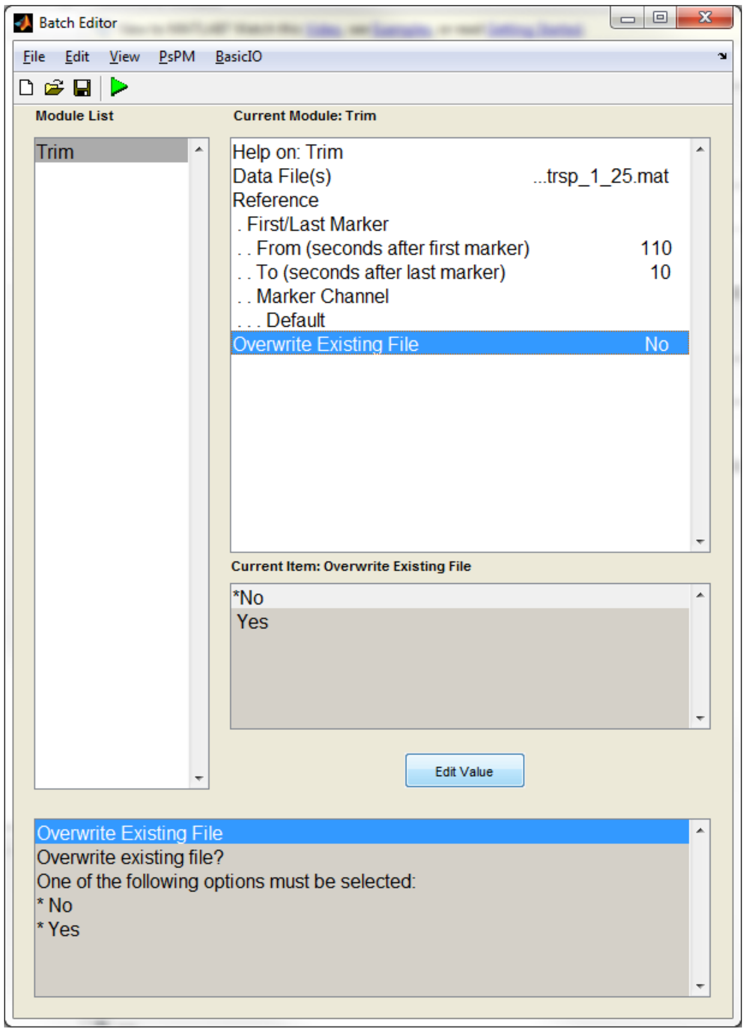
Import
Import external data files for use by SCRalyze. First, specify the data type. Then, other fields will come up as required for this data type. The imported data will be written to a new .mat file, prepended with 'scr_'.

This branch contains 2 items:
* Data Type

Figure 25.1: Batch editor for importing raw SCR data.

25.2 Trim

Most of the time, the SCR recording is switched on some time before the actual experiment starts and is switched off some time after it ends. These parts of the time series may contain large artefacts from subject motion, setting up the equipment, etc. Trimming excludes these parts from the analyses.



Current Module: Trim

```
Help on: Trim
Data File(s)                              ...trsp_1_25.mat
Reference
. First/Last Marker
. . From (seconds after first marker)              110
. . To (seconds after last marker)                  10
. . Marker Channel
. . . Default
Overwrite Existing File                             No
```

Current Item: Overwrite Existing File

```
*No
 Yes
```

```
Overwrite Existing File
Overwrite existing file?
One of the following options must be selected:
* No
* Yes
```

Figure 25.2: Batch editor for trimming SCR data.

- In the batch editor go to *PsPM* → *Data Preparation* →*Trim*.
  - Select the currently imported file as Data File. Alternatively, you can use the *Dependency* button.
- You can choose different References to specify how data should be trimmed. Here, select *First/Last Marker* and
  - *From (seconds after first marker)*: +110 (In the current data set the first seconds are not task-relevant and are thus trimmed. Note that a positive number will delete at least the first marker of the original

data. Thus, make sure that a possible specification of the First-Level takes this into account.)

  - *To (seconds after first marker)*: 10 (The specified time window should be large enough to completely include the last trial.)

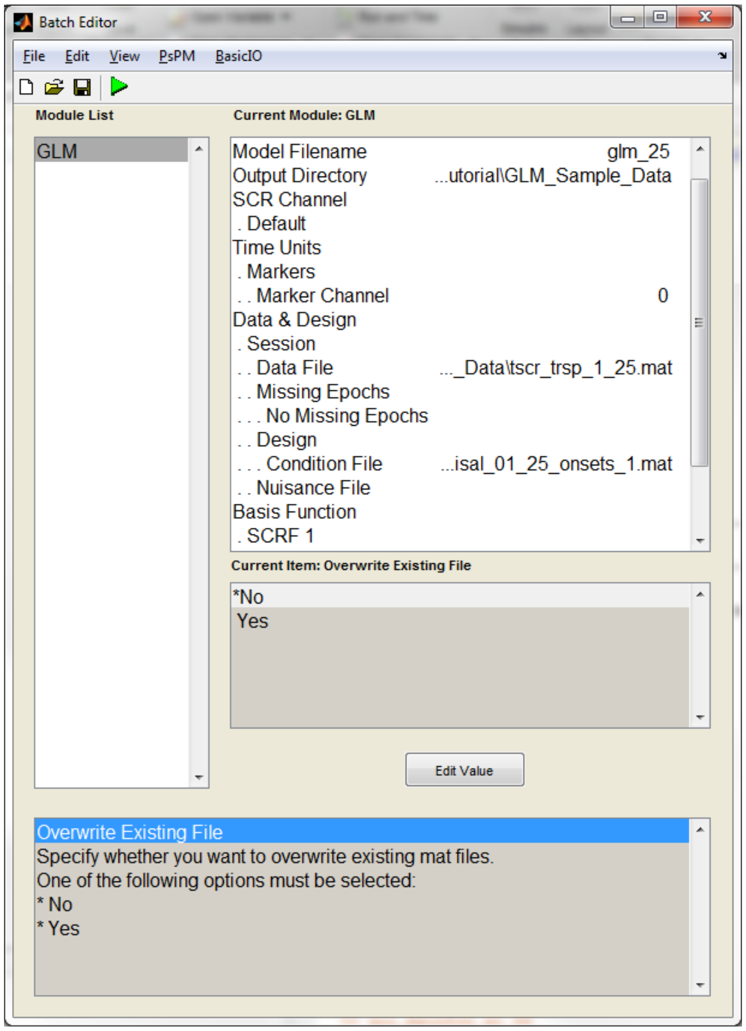
You can also specify time points before the markers by using negative numbers (see above). See Figure 25.2 for a picture of the final batch editor. Again, it is recommended to save the batch and script for future use. Run the batch. A mat file with the name `tscr_trsp_1_25.mat` will be created (i.e., a "t" will be pre-pended).

25.3 First-Level

Now, we specify the GLM on the first (i.e., the subject-specific) level, estimating average responses per experimental condition (this equivalent to averaging responses over trials in classical "peak scoring" analysis, or to performing a first level analysis in SPM).

- In the batch editor go to *PsPM* → *First Level* → *SCR* →*GLM*.
- *Model Filename*: This is the name of the resulting mat file containing the GLM. Here, we call it `glm_25`.
- *Output Directory*: The resulting mat file containing the GLM will be saved in this folder.
- *Time Units* on which the specification of the conditions will be based. Here, we want to analyze with respect to Markers.
- *Session(s)*: Similar to SPM, SCRalyze allows the specification of multiple sessions per subject. Here, we only have one session.
  - Add the trimmed data (`tscr_trsp_1_25.mat`) as Data File. Alternatively, you can use the *Dependency* button.
  - *Missing Epochs* allows the specification of time periods which you do not want to include in the analysis for example because they contain corrupted data. In the example data set, there are no missing epochs.
  - *Data & Design*: This option allows specifying the different conditions of the experiment. You need to provide the information when the different conditions started. This time information has to be in the unit specified above (see *Time Units*). In the current data set, condition information is provided in Markers and is stored in the mat file `trSP_appraisal_01_25.mat`, which you should select as *Condition File*. When you open this file in MATLAB, you will see that the file contains a 1x3 cell "names" and a 1x3 cell "onsets" specifying three conditions. The numbers in "onsets" correspond to the nth marker when the respective trial started.

  – *Nuisance File* allows the inclusion of nuisance confounds such as motion or respiration parameters. Here, we leave it empty.



Model Filename glm_25
Output Directory ...utorial\GLM_Sample_Data
SCR Channel
. Default
Time Units
. Markers
. . Marker Channel 0
Data & Design
. Session
. . Data File ..._Data\tscr_trsp_1_25.mat
. . Missing Epochs
. . . No Missing Epochs
. . Design
. . . Condition File ...isal_01_25_onsets_1.mat
. . Nuisance File
Basis Function
. SCRF 1

Current Item: Overwrite Existing File

*No
Yes

Overwrite Existing File
Specify whether you want to overwrite existing mat files.
One of the following options must be selected:
* No
* Yes

Figure 25.3: Batch editor for First-Level analysis.

- *Basis Function*: We chose the default *SCRF 1*, which is the version with time derivative. The default basis set follows the recommendations by [12].
- *Normalize*: The design of the current data set is "within-subjects." Therefore, we want to z-normalize the data.

- *Filter Settings*: We choose the default filter settings, which follow the recommendations by [12]:
  - first-order low-pass Butterworth filter with a cutoff frequency of 5 Hz
  - first-order high-pass Butterworth filter with a cutoff frequency of 0.05 Hz
  - Data are resampled to 10 Hz.
  - The filter direction is unidirectional.

See Figure 25.3 for a picture of the final batch editor. Again, we recommend saving the batch and script for future use. Run the batch. A mat file with the specified name will be created in the specified folder.

25.4 Review First-Level Model

You can have a look at plots related to the first-level model in the batch editor.

- In the batch editor go to *PsPM* → *First Level* → *Review First-level Model.*
- Specify the Model file that you want to review. Select the mat file containing the GLM (e.g., `glm_25.mat`).
- The *Model Type* is of course GLM in the current example. You can set the following options to get different types of information:
  1. *Design matrix*: On the x-axis, you see the number of regressors in the model. Here we have specified 7 regressors in total. On the y-axis, you see the time course of the experiment. Note that there are 7 regresssors because the specification for the *Basis Function* was *SCRF with time derivative*. That is, 3 (conditions times) x 2 (basis functions) + 1 (constant) = 7 regressors. You can see that neutral and aversive are intermixed and fall into three blocks. See Figure 25.4.
  2. *Orthogonality*: On the x- and y-axes you see the specified regressors. The plot depicts the collinearity between regressors. See Figure 25.5.
  3. *Predicted & Observed*: You see the model fit including the observed response over the whole time series, which you can visually compare to the predicted time series by the specified model. Note that the predicted data will never follow the observed data very closely, because we estimated the average response per condition which will deviate from the response in each individual trial. See Figure 25.6.
  4. *Regressor names*: The names of the regressors as specified in the *Conditions File* under *Data & Design* are written to the MATLAB command window. The labels bf 1 and bf 2 refer to the two used basis functions (*SCRF with time derivative*).

5. *Reconstructed*: Here you see the estimated responses per condition for the first basis function. These are estimates of the true responses which contain noise. The noise term can be negative but if all estimated responses across an entire group are negative it is a good idea to look for confounds or correlations in the design. See Figure 25.7.

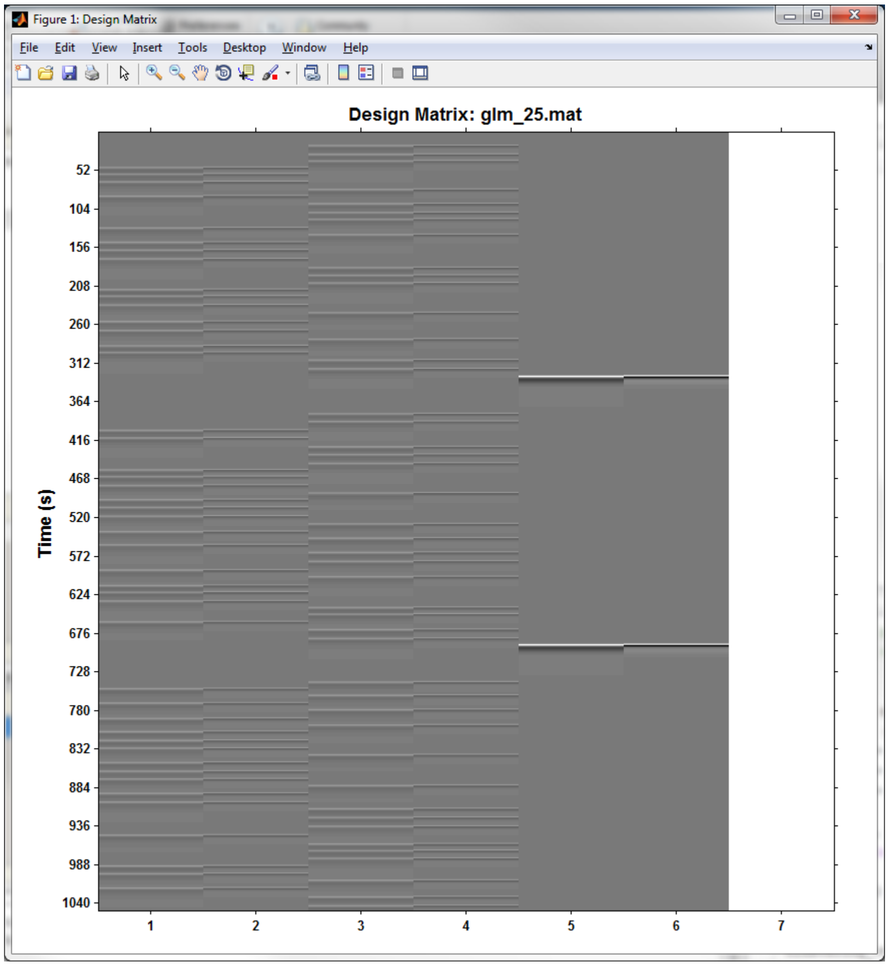


Figure 25.4: Design matrix for sample participant. On the x-axis, you see the number of regressors in the model. There are 7 regresssors (i.e., 3 (conditions times) x 2 (basis functions) + 1 (constant)).

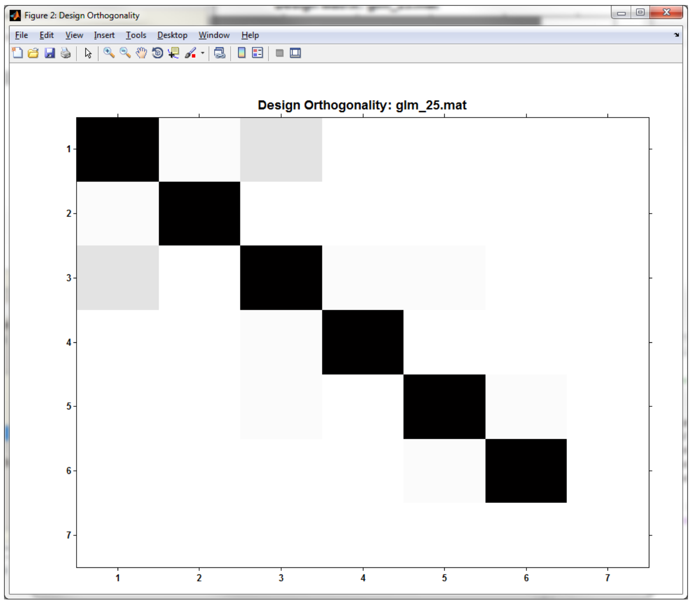


Figure 2: Design Orthogonality
File Edit View Insert Tools Desktop Window Help
Design Orthogonality: glm_25.mat
1
2
3
4
5
6
7

Figure 25.5: Design orthogonality plot for sample participant. On the x- and y-axes you see the 7specified regressors.

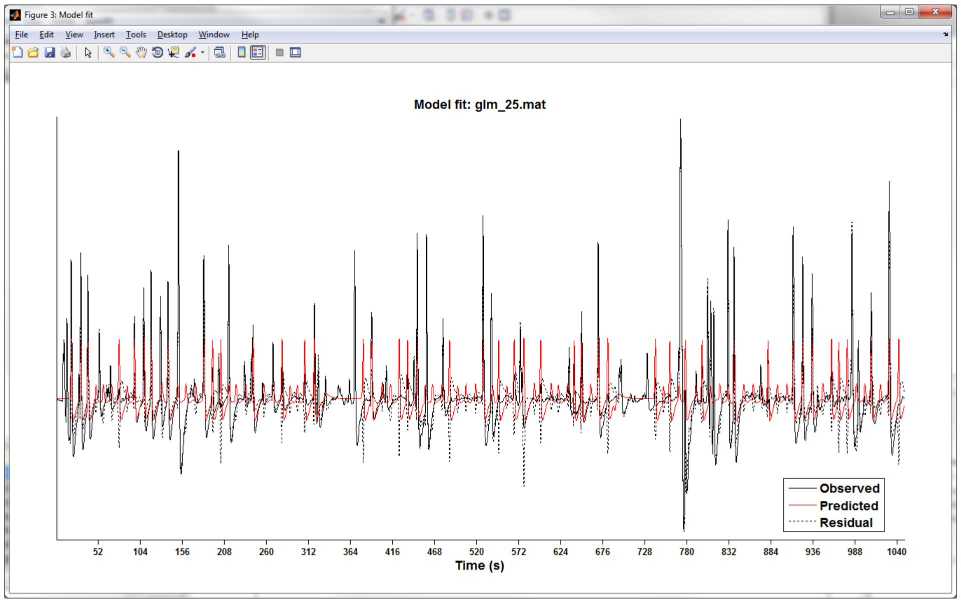


Figure 25.6: Model fit for sample participant. You see the observed response and the predicted response according to the model over the whole time series.

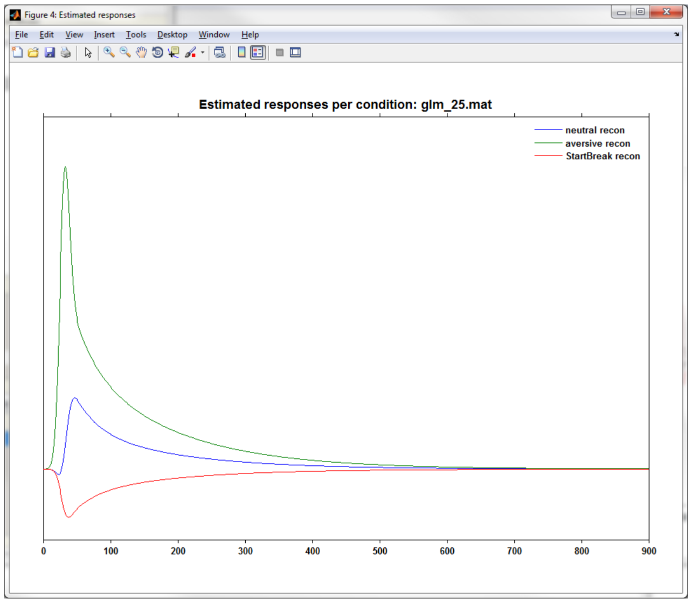


Figure 25.7: Estimated responses per condition for sample participant. You see the estimated responses per condition for the first basis function.

25.5 First-Level Contrast Manager

We want to compare the difference between aversive and neutral pictures across participants (i.e., on the second level). This can be tested in a paired t-test. A paired t-test is essentially a one-sample t-test on difference values between two conditions. The contrast manager allows you to compute such difference values, for each participant (i.e., on the first level).

- In the batch editor go to *PsPM* → *First-Level* → *First-Level Contrasts*.
- *Model File(s)*: Select the mat file containing the GLM (here `glm_25.mat`).
- *Stats type*: You can choose between 3 different options to set up the contrasts. Here, we choose the option Condition, which specifies contrasts formulated in terms of conditions and automatically uses only the first basis function (i.e., without derivatives).
- *Contrast*
  - *Contrast Name*: Chose a name so that you can easily refer to the contrast (e.g., aversive>neutral)
  - *Contrast Vector*: Specify a vector of the contrast you want to test for. Here, we are interested in the difference between aversive and neutral pictures, i.e., the first and the second condition. Therefore,

specify the vector [-1, 1, 0]. You could also leave out the last zero because trailing zeros are automatically appended. (But you could not leave out the first zero in a contrast such as [0 -1 1].)

25.6 Export Statistics

You can use this functionality to export all necessary parameters on the screen or into an extra file, which allows you to analyze the data in the statistics program of your choice. For details see *PsPM → First Level → Export Statistics.*

25.7 Adapting scripts for multiple participants and second-level analyses

For didactic purposes, we have so far only considered one participant. But in most experiments, you want to analyze data from multiple subjects (i.e., a second-level analysis). For a description on how to do this see Chapters 27 and 28. You can find a script that runs the above described analyses for multiple participants at pspm.sourceforge.net (import_trim_glm_contr_job_loop).

26 Delay fear conditioning data

The example data set comprises SCR data from 20 participants, with 40 trials each, which can be downloaded from pspm.sourceforge.net together with the result files of the model inversion. Participants underwent a differential delay fear conditioning paradigm where coloured circles were presented for 4 seconds each. Either the blue or orange coloured circles (balanced over participants) were probabilistically paired with an electric stimulation (unconditioned stimulus, US) that coterminated with the stimulus (conditioned stimulus, CS+) while the CS- were always presented alone.

Presenting the CS+ causes phasic sympathetic arousal in anticipation of an electric shock. This anticipatory reaction typically occurs with an unknown and variable latency after stimulus onset. For such cases, a GLM should not be used because it assumes that sympathetic arousal follows the stimulus with a fixed latency. In contrast, the non-linear model allows estimating (i) a variable onset within a user-specified time window and (ii) a variable duration of a sympathetic response. Additionally, the non-linear model includes estimation of spontaneous fluctuations that occur between trials.

In this tutorial, the non-linear model is estimated. The estimated parameters can then be used to test the within-subject hypothesis that the sympathetic arousal between CS+ and CS- are different on a group level.

26.1 Setup the Non-Linear Model

In this step you prepare the model estimation. For this, you need to provide timing information about the experimental events which can either be entered manually or as a Matlab file. For large numbers of events as in this experiment, a timing file can easily be created using a few lines of Matlab code. Here we use timing information saved in the trimmed PsPM file. In this file, a trigger was saved at every CS onset. For timing information from other sources, the code has to be adapted accordingly. Go to the Matlab command window and enter the following lines, but replace `[insert path!]` by the correct path where the filtered SCR data from pspm.sourceforge.net is saved on your hard drive, e.g. `C:\PsPM_Tutorial\nonlinear\`.

```
data_path = [insert path!]; % set the correct PsPM file path here!

p = 1; % the first participant
SOA = 3.5; % delay between CS and US onset in seconds

% load events of CS onsets from trimmed scr data
pname = fullfile(data_path,sprintf('tscr_s%i.mat',p))
[sts, infos, data, filestruct] = scr_load_data(pname,'events');

CS_onset = data{1, 1}.data; % recorded triggers of CS onset in seconds
US_onset = data{1, 1}.data + SOA; % US starts 3.5 seconds after CS

% variable latency for CS response between CS onset and US onset/omission
events{1} = [CS_onset US_onset]; % define an interval of 3.5 seconds for each
trial
% constant onset for US response
events{2} = US_onset;

save(fullfile(data_path,sprintf('event_timings_s%i.mat',p)),'events')
```

The resulting file `event_timings_s1.mat` contains the variable events with two cells.

- The first cell *events{1}* is a 40 x 2 matrix with the onset and offset of a time window for each trial in which we expect the response to the CS (both CS+ and CS- are treated the same here). We expect the response to the CS to occur with a variable onset in time. Note that the window for the CS response ends when the US starts (or is omitted), i.e. after 3.5 seconds.

- The second cell *events{2}* is a 40 x 1 vector that marks the onset of the electrical stimulation. For this event we expect an evoked response with constant latency after US occurrence. Importantly, we provide timings *both for the occurrence and the omission of the US.* Modelling an unconditioned response in all trials is necessary to get unbiased estimates for the CS responses. In general, the trial structure should always be the same for all trials, across experimental conditions.

Now the non-linear model estimation can be set up (Figure 26.1).

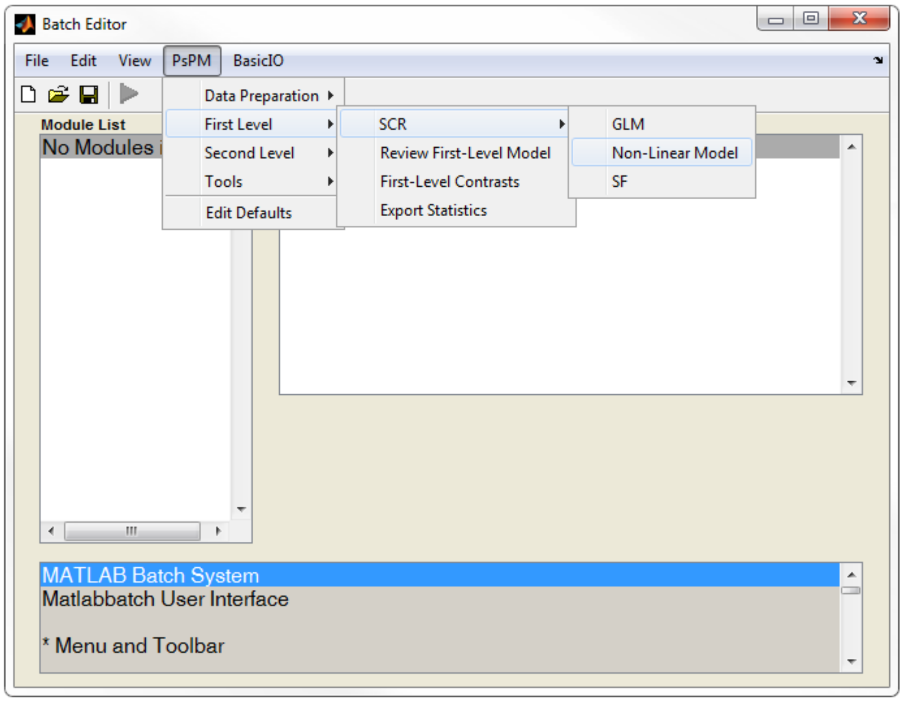


Figure 26.1: Prepare a batch to set up a non linear model for SCR analysis.

- In the batch editor go to *PsPM* → *First Level* → *SCR* → *Non-Linear Model*
- *Model Filename*: dcm_s1
- *Output Directory*: Specify a folder where the model le will be saved
- *Channel*: Default
- *Data & Design:*→*New: Session*
  - *Data File*: choose tscr_s1.mat from your hard drive
  - *Design*→*Timing File*→Select the previously created `event_timings_s1.mat` which was saved to your scr folder
  - *Condition names: New Condition* here we can dene conditions that each trial belongs to. Create three new conditions. Note that we split CS+ trials in reinforced (CS+|US+) and non-reinforced (CS+|US-) trials because CS+|US+ trials are excluded from some analyses.
    - *Condition*

        · *Name: CS+/US-*
        · *Index: 6 7 11 14 16 18 26 29 38 40*
      * *Condition*
        · *Name: CS+/US+*
        · *Index: 13 17 19 20 21 24 28 31 32 37*
      * *Condition*
        · *Name: CS-*
        · *Index: 1 2 3 4 5 8 9 10 12 15 22 23 25 27 30 33 34 35 36 39*

- *Display Options*
  - *Display Progress Window*: If you plan to run the estimation in the background, you can turn off the progress window with this option
- *For the remaining fields you can keep the default settings*

Save the batch and script as `batch_model_s1.m` and press run. A window will be displayed that shows the progress of model estimation for each trial. The estimation for an entire session takes several minutes depending on your system. The file `dcm_s1.mat` will be created. If you want to, you can review the first-level model by clicking *Review model* in the PsPM user interface. In the new window (Figure 26.2) add the non-linear model `dcm_s1.mat` and choose *Display All trials for one session.* A graphics window (Fig X) will display the original data (blue) together with the model estimation (green).

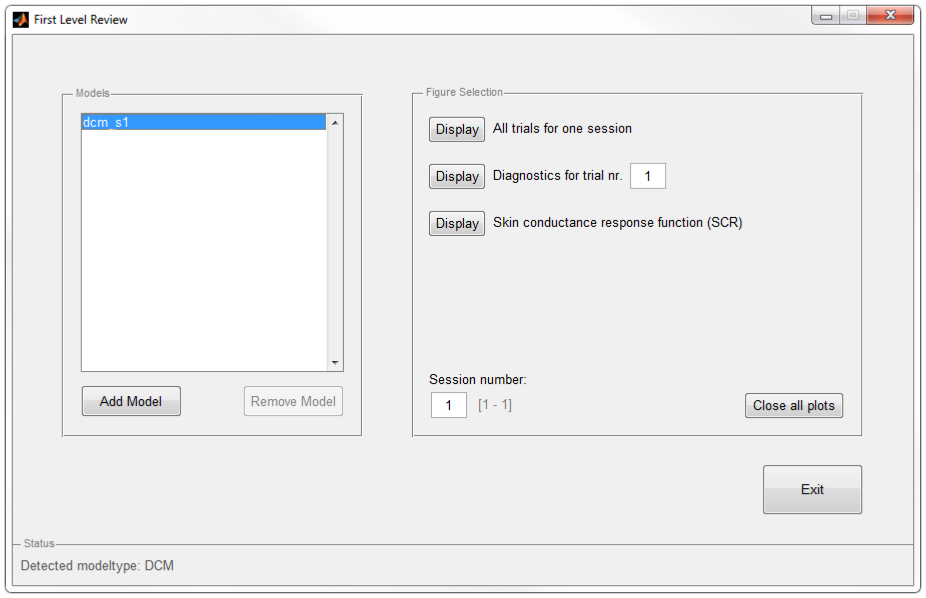


Figure 26.2: Use *Review Model* to inspect parameter estimation for one or multiple trials.

26.2 First-level Contrasts

In this step you specify the hypothesis that a CS+ elicits a stronger sympathetic arousal than a CS- for an individual participant by providing a contrast weight vector.

- In the batch editor go to *PsPM→First Level→First-level Contrasts*
- *Model File(s):* Select `dcm_s1.mat`
- *Specify Contrasts on Datatype: param*
- *Contrasts: New Contrast*
  - *Contrast*
    - *Name: Fear Learning*
    - *Contrast Vector: -1 -1 -1 -1 -1 2 2 -1 -1 -1 2 -1 0 2 -1 2 0 2 0 0 0 -1 -1 0 -1 2 -1 0 2 -1 0 0 -1 -1 -1 -1 0 2 -1 2* Note: A weight of +2 is assigned to the CS+|US- because we have twice as many CS- and the contrast vector has to sum up to zero.
    - *Name: Electric Stimulation* We include this contrast optionally to check if the electrical stimulation caused an increase in sympathetic arousal which is a premise for successful fear conditioning.
    - *Contrast Vector: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 3 -1 -1 -1 3 -1 3 3 3 -1 -1 3 -1 -1 -1 3 -1 -1 3 3 -1 -1 -1 -1 3 -1 -1 -1*
- *Delete: existing contrasts: No (default)*

Save the batch and script as `batch_contrast_s1.m` and press run. The model file of participant s1 now contains the contrast that can be forwarded to a statistical test on group level. Preparing analyses for more than one scr file is explained in Chapter 27.

26.3 Exporting statistics

PsPM provides basic tools for statistical analysis on the estimates of the non-linear model. Alternatively you can export the results as a text file to use them in different software applications.

- In the batch editor go to SCR→First Level→Export Data
- *Model File(s):* Select *dcm_ s1.mat*
- *Specify Export on Datatype: param*
- *Target*
  - *Filename* Enter `dcm_s01`
- *Specify Delimiter for Output File*

  - *Semicolon*

The text file will be saved to Matlab's current working directory.

27 Adapting scripts for multiple participants

In most experiments, you want to analyze data from multiple subjects, which implies that you have to repeat model estimation several times. There are two simple strategies to make this easier.

- Use dependencies: You can specify all steps for one subject in one go. At some points, you need to specify a file from a previous step (e.g., when trimming you need the file containing the imported data). In these cases, you can use the button "Dependency" to specify the corresponding file.
- Loop over participants in the script: You can easily adapt the scripts for many subjects by including a loop and adapting the file names. This is independent of whether or not you use dependencies. To show you how this works, see the examples below.

27.1 Importing and trimming data

Here, we create an example script for importing and trimming data. We chose these steps since they are required in all types of analysis. The current example is based on the Appraisal dataset described in Chapter 25.

- In the batch editor specify all parameters in PsPM → Data Preparation → Import.
- You do not need to specify the SCR Data File. This will later be put in the script that loops over subjects.
- In the batch editor specify all parameters in PsPM → Data Preparation → Trim.
- Use the dependency button to select the imported Data File.
- Save batch and script.
- Open the saved job script (e.g., `batch_import_trim_job.m`) and build a loop around the lines of code that were saved. In this loop, create a subject-specific filename (and make sure to include the whole path to the files in the variable data_path).

```
data_path = [insert path!];

for p = 25:39; % participants
  % load participant-specific raw data
  matlabbatch{1}.pspm{1}.prep{1}.import.datatype.spike.datafile = ...
```

```
    {fullfile(data_path,sprintf('trsp_1_%i.smr',p))};
  spike.importtype{1}.scr.chan_nr.chan_nr_spec = 2;
  spike.importtype{1}.scr.transfer.none = true;
  spike.importtype{2}.marker.chan_nr.chan_nr_spec = 0;
  matlabbatch{1}.pspm{1}.prep{1}.import.datatype.spike = spike;
  matlabbatch{1}.pspm{1}.prep{1}.import.overwrite = false;
  matlabbatch{2}.pspm{1}.prep{1}.trim.datafile(1) = ...
    cfg_dep('Import:  Output File', substruct('.', ...
    'val', '{}',{1}, '.','val', ...
    '{}',{1}, '.','val', '{}',{1}), substruct('()',{':'}));
  matlabbatch{2}.pspm{1}.prep{1}.trim.ref.ref_mrk.from = 0;
  matlabbatch{2}.pspm{1}.prep{1}.trim.ref.ref_mrk.to = 15;
  matlabbatch{2}.pspm{1}.prep{1}.trim.ref.ref_mrk.mrk_chan.chan_def = 0;
  matlabbatch{2}.pspm{1}.prep{1}.trim.overwrite = false;

  % run batch
  scr_jobman('initcfg');
  scr_jobman('run', matlabbatch);

end
```

Run the script. The folder `data_path` now contains trimmed scr files for all subjects.

27.2 Inverting non-linear models

Here we adapt the model inversion from Chapter 26 to multiple subjects. We use similar loops as above to create timing files and PsPM batches.

First, open the MATLAB script that created the timing file `event_timings_s1.mat` and construct a loop around it for subjects 2 to 20. Execute the script to obtain files containing timing information for subjects 2 to 20. Alternatively you can use the timing files for subjects 2 to 20 downloaded from pspm.sourceforge.net.

Next, open the saved job script (e.g., `batch_model_s1_job.m`) created in Chapter 26 and build a loop around the lines of code that were saved. Make sure to include the whole path to the files in the variable `data_path`. You can use the following code as guideline.

```
% set the path where you saved the scr data files
data_path = [insert path!];

% set the path where you saved the model file of participant 1
model_path = [insert path!]; % initiate PsPM scr_init % loop through participants
2 to 20

% initialize PsPM
scr_init
scr_jobman('initcfg')
```

```
for p = 2:20

  clear matlabbatch dcm
  dcm.modelfile = ['dcm_s' num2str(p)];
  dcm.outdir = {'D:\SCRalyze_tutorial\dcm_hra'};
  dcm.chan.chan_def = 0;
  dcm.session.datafile = ...
    {fullfile(data_path,['tscr_s' num2str(p) '.mat'])};
  dcm.session.timing.timingfile = ...
    {fullfile(data_path,['event_timings_s' num2str(p) '.mat'])};
  dcm.session.condition(1).name = 'CS+|US-';
  dcm.session.condition(1).index = [6 7 11 14 16 18 26 29 38 40];
  dcm.session.condition(2).name = 'CS+|US+';
  dcm.session.condition(2).index = [13 17 19 20 21 24 28 31 32 37];
  dcm.session.condition(3).name = 'CS-';
  dcm.session.condition(3).index = ...
    [1 2 3 4 5 8 9 10 12 15 22 23 25 27 30 33 34 35 36 39];
  dcm.data_options.norm = 0;
  dcm.data_options.filter.def = 0;
  dcm.resp_options.crfupdate = 0;
  dcm.resp_options.indrf = 0;
  dcm.resp_options.getrf = 0;
  dcm.resp_options.rf = 0;
  dcm.inv_options.depth = 2;
  dcm.inv_options.sfpre = 2;
  dcm.inv_options.sfpost = 5;
  dcm.inv_options.sffreq = 0.5;
  dcm.inv_options.sclpre = 2;
  dcm.inv_options.sclpost = 5;
  dcm.inv_options.ascr_sigma_offset = 0.1;
  dcm.disp_options.dispwin = 0;
  dcm.disp_options.dispsmallwin = 0;

  matlabbatch{1}.pspm{1}.first_level{1}.scr{1}.dcm = dcm;

  % run batch
  scr_jobman('run', matlabbatch);

end
```

Similarly to the previous step, the batch of the first-level Contrast of participant 1 is adapted to participants 2 to 20. Run the following code in MATLAB, but replace `[insert path!]` with the path on your hard drive where you saved `dcm_s1.mat`:

```
% set the path where you saved the estimated non-linear model
model_path = [insert path!];

% initialize PsPM
```

```
scr_init
scr_jobman('initcfg')

% loop through participants 2 to 20
for p = 2:20

  clear matlabbatch
  matlabbatch{1}.pspm{1}.first_level{1}.contrast.modelfile = ...
    {fullfile(model_path,['dcm_s' num2str(p) '.mat'])};
  matlabbatch{1}.pspm{1}.first_level{1}.contrast.datatype = 'param';
  matlabbatch{1}.pspm{1}.first_level{1}.contrast.con(1).conname = 'Fear Learning';
  matlabbatch{1}.pspm{1}.first_level{1}.contrast.con(1).convec = ...
    [-1 -1 -1 -1 -1 2 2 -1 -1 -1 2 -1 0 2 -1 2 0 2 ...
    0 0 0 -1 -1 0 -1 2 -1 0 2 -1 0 0 -1 -1 -1 -1 0 2 -1 2];
  matlabbatch{1}.pspm{1}.first_level{1}.contrast.con(2).conname = ...
    'Electric Stimulation';
  matlabbatch{1}.pspm{1}.first_level{1}.contrast.con(2).convec = ...
    [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 3 -1 -1 -1 ...
    3 -1 3 3 3 -1 -1 3 -1 -1 -1 3 -1 -1 3 3 -1 -1 -1 -1 3 -1 -1 -1];
  matlabbatch{1}.pspm{1}.first_level{1}.contrast.deletecon = true;

  % run batch manually
  scr_jobman('run', matlabbatch);

end
```

Run the script. The statistics for each contrast are now added to the model file.

28 Second-level Contrast Manager

In most cases, we want to compare the difference between conditions across participants on a group level (i.e., the second level in a hierarchical model). Each participant represents one data point that goes into a one sample t-test.

Below we demonstrate how to set-up Second-level contrasts for GLM and non-linear models.

28.1 GLM

For the appraisal data of Chapter 25.

28.1.1 Define Second-Level Model

- In the batch editor go to *PsPM→Second Level→Define Second-Level Model.*
  - *Test Type: One Sample T-Test*
  - *Model File(s)*: Select the model files from all participants
  - *Output Directory:* Choose a location to save the group result

  - *Filename for Output Model*: Chose a name so that you can easily refer to the contrast
  - *Define Contrast: Read from First Model File*: Choose *All Contrasts.* This will select the contrast(s) that we created in the first level (i.e., in our case aversive>neutral)

28.1.2 Report Second-Level Results

- In the batch editor go to *PsPM →Second Level→Report Second-Level Results.*
  - *Model File*: Select the second level model file that you created in the previous step.
  - *Contrast Vector*: Leave empty because we want all contrasts to be reported.

Run the batch. A figure appears that shows you a bar graph of the mean of the parameter estimates and the statistics will be printed to the MATLAB command window.

28.2 Non-linear models

Here, we test the difference in anticipatory sympathetic arousal elicited by a CS+ vs CS- on a group level (see Chapter 26). As a supplementary hypothesis we want to check if the US+ indeed elicited a stronger response than the US-. From the 1st level contrast, eight statistics were generated for each model file. Use the *Review Model* utility from the PsPM main GUI to get a list of all contrasts. In the *First Level Review Manager*, click on *Add Model*, choose any model file created previously and press *Done.* Now press the Show button next to *Contrast names in command window* (Figure 28.1) to get a list of all contrasts:

- *Contrast 1: Fear Learning - Flexible response # 1: amplitude*
- *Contrast 2: Electric Stimulation - Flexible response # 1: amplitude*
- *Contrast 3: Fear Learning - Flexible response # 1: peak latency*
- *Contrast 4: Electric Stimulation - Flexible response # 1: peak latency*
- *Contrast 5: Fear Learning - Flexible response # 1: dispersion*
- *Contrast 6: Electric Stimulation - Flexible response # 1: dispersion*
- *Contrast 7: Fear Learning - Fixed response # 1 response amplitude*
- *Contrast 8: Electric Stimulation - Fixed response # 1 response amplitude*

Our hypothesis is concerned only with the response amplitude of the flexible event (anticipation of the CS) and the response amplitude of the fixed event (electrical stimulation), which are the first and eighth contrasts. The indices of these two contrasts are used for the setup of the Second-Level model below.

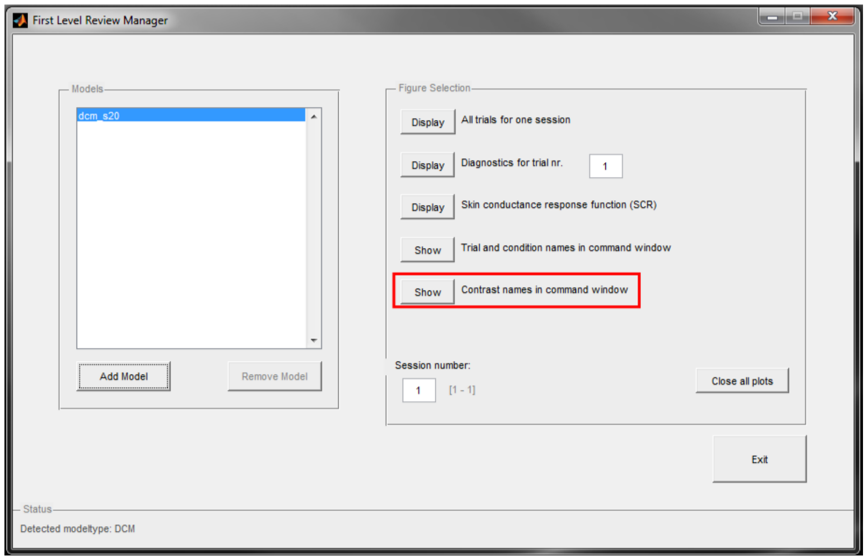

First Level Review Manager
Models
dcm_s20
Add Model
Remove Model
Figure Selection
Display
All trials for one session
Display
Diagnostics for trial nr.
1
Display
Skin conductance response function (SCR)
Show
Trial and condition names in command window
Show
Contrast names in command window
Session number:
1
[1 - 1]
Close all plots
Exit
Status
Detected modeltype: DCM

Figure 28.1: The First Level Review Manager displays all contrasts that are stored in a model file.

- In the batch editor go to PsPM→Second Level→Define Second-Level Model
- Test Type: One Sample T-Test
- Model File(s): Select the model files from all participants
- Output Directory: Choose a location to save the group result
- Filename for Output Model: Fear_Learning
- Define Contrast: Read from First Model File
  - Contrast Vector: [1, 8]
- Overwrite Existing File: No (default)

Run the batch. You can display the results of the second level contrasts by pressing *Report 2nd level result* in the PsPM user interface and choosing `Fear_Learning.mat`. A Figure displays the mean differences and standard error for the hypothesis that the CS+ elicits a stronger sympathetic arousal than CS- and that an electric stimulation results in an increase in sympathetic arousal (Figure 28.2).

Statistical parameters are printed in the MATLAB command window (Figure 28.3).

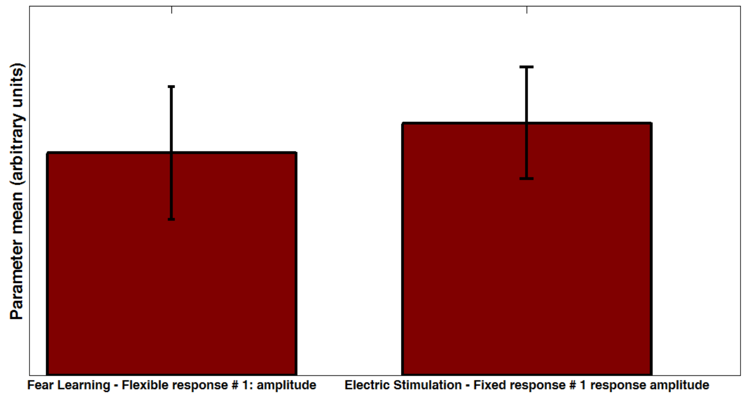
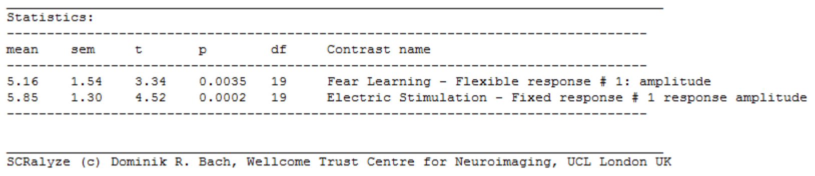


Figure 28.2: Display of results from a group analysis (2nd level). Left bar: difference in amplitude of sympathetic arousal (aSA) to CS+|US- minus CS-. Right bar: difference in aSA to US+ minus US-



```
Statistics:
--------------------------------------------------------------------------------
mean    sem     t       p        df     Contrast name
--------------------------------------------------------------------------------
5.16    1.54    3.34    0.0035   19     Fear Learning - Flexible response # 1: amplitude
5.85    1.30    4.52    0.0002   19     Electric Stimulation - Fixed response # 1 response amplitude
--------------------------------------------------------------------------------

SCRalyze (c) Dominik R. Bach, Wellcome Trust Centre for Neuroimaging, UCL London UK
```

Figure 28.3: Statistics from one-sampled t-tests CS|US- minus CS- and US+ minus US- for 20 participants.

Part IV
Developer's Guide

29 General

29.1 Data files: General structure

In PsPM the data is saved in mat-files. Each file contains two variables:

- `infos` A struct variable with general infos
- `data` A cell array with a cell for each channel.

The cells contain a struct with channel specific infos and data. The structs have the mandatory fields:

- `infos.duration` (in seconds)
- `data{n}.header`
  - `data{n}.header.chantype` (as defined in the settings)
  - `data{n}.header.sr` (sample rate in 1/second, or timestamp units in seconds)
  - `data{n}.header.units` (data units, or 'events')
  - `data{n}.data` (actual data)

Additionally, a typical file contains the optional infos:

- `infos.sourcefile`
- `infos.importfile`
- `infos.importdate`
- `infos.sourcetype`
- `infos.recdate`
- `infos.rectime`

Some data manipulation functions (in particular, scr_trim) update infos to record some file history.

29.2 How to add a new import data type

29.2.1 Add function

Function name: scr_get_xxx (where xxx is the data type name).

Format:

```
[sts, import, sourceinfo] = scr_get_xxx(datafile, import)
```

The function needs to take an import job and add, for each job, fields

- .data - the actual data for this channel (column vector)
- .sr - the sample rate for this channel (only if .autosr enabled in scr_init)

optional fields

- .marker - for marker channels (timestamps or continuous, see scr_get_marker)
- .markerinfo – optional, see scr_get_marker

- .minfreq - minimum frequency for pulse channels
- .units - if data units are defined by the recording software
- sts: -1 if import is unsuccessful

sourceinfo: contains information on the source file, with field

- .chan - a cell of string descriptions of the imported source channels, e. g. names, or numbers any optional fields that will be added to infos.source (e. g. recording date & time, and others)

Notes for multiple blocks: file formats that support multiple block storage within one file can return cell arrays import{1:blkno} and sourceinfo{1:blkno}; PsPM will save individual files for each block, with a filename 'scr_fn_blk0x.mat'.

29.2.2 Add information to settings

The file scr_init contains a block that defines possible import data types. Add a new field here

```
% Description of data type
% ------------------------------------------
defaults.import.datatypes(1) = ...
struct('short', 'xxx', ...  % short name for internal purposes
   'long', 'Datatype description', ...  % long name for GUI
   'ext', '*', ...  % data file extension
   'funct', @scr_get_xxx, ...  % import function
   'chantypes', {{defaults.chantypes.type}}, ...  % allowed channel
types
   'chandescription', 'channel', ...  % description of channels for
GUI
   'multioption', 1, ...  % import of multiple channels for GUI
   'searchoption', 1, ...  % allow channel name search for GUI
   'automarker', 0, ...  % marker not stored in separate channel
   'autosr', 1); % sample rate automatically assigned
```

Good to know:

- the “long” definition is used in the GUI – make sure it's readable
- if no event channels can be imported, change .chantypes
- if channels have searchable names in the import file, set .searchoption = 1
- if no channel number needs to be assigned for the marker channel, set .automarker = 1

- if sample rate is contained in import file and determined during import, set .autosr = 1
- if you need external functions – put them into a folder in the ‘import’ subdirectory and add/remove this path within the scr_get_xxx function

29.3 How to add a new channel type

29.3.1 Add function

Function name: scr_get_xxx (where xxx is the channel type)

Format:

```
[sts, data] = scr_get_channeltype(import)
```

data: data cell of structure readable by scr_load_data

Good to know: for event channels, use the function scr_get_events to convert various event formats into time stamps (see scr_get_marker or scr_get_hb as an example)

29.3.2 Add information to settings

Add information on the new channel type and import function to

```
defaults.chantypes(k).type = 'xxx';           % channel type name
defaults.chantypes(k).import = @scr_get_xxx;  % conversion function
defaults.chantypes(k).data = 'xxx';           % 'wave' or 'events'
```

29.4 How to add a new GLM type

29.4.1 Add information to settings (Example SCR)

```
defaults.glm(1) = ...
   struct('modality', 'scr', ...  % modality name
     'cbf', struct('fhandle', @scr_bf_scrf, 'args', 1), ...
                                   % default basis function/set
     'filter', struct('lpfreq', 5, 'lporder', 1, ...
'hpfreq', 0.05, 'hporder', 1, 'down', 10, 'direction', 'uni'));
                                   % default filter settings
```

29.4.2 Add default basis function

Function name: scr_bf_xxx

Function arguments: vector of arguments, first element is time resolution, further arguments as defined in defaults.glm(n).cbf.args

29.5 Warning IDs in PsPM

29.5.1 General

29.5.2 Function specific

scr_load_data

- invalid_data_structure
- nonexistent_channeltype

scr_trim

- marker_out_of_range

scr_find_channel

- not_allowed_channeltype
- multiple_matching_channels
- no_matching_channels

scr_get_scr

- no_conversion_constant

scr_import

- invalid_channeltype

scr_pp

- invalid_freq

scr_prepdata

- no_low_pass_filtering
- downsampling_failed
- nonint_sr

scr_get_timing

- invalid_vector_size
- event_names_dont_match
- no_numeric_vector
- no_integers

30 List of data formats

30.1 Supported Channel types

Data format	SCR	ECG	Heart Rate	Heart Beat	Heart Period	Respiration	Respiration Rate	Pupil Size	Marker	Custom
CED Spike	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Matlab	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Text	✓	✓	✓		✓	✓	✓	✓		✓
Biopach AcqKnowledge (≤ v3.9.0)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Biopac AcqKnowledge (exported)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Labchart exported (≤ v7.1)	✓	✓	✓		✓	✓	✓	✓	✓	✓
Labchart exported (≥ v7.2)	✓	✓	✓		✓	✓	✓	✓	✓	✓
VarioPort	✓	✓	✓		✓	✓	✓	✓	✓	✓
Biograph Infiniti (exported)	✓			✓		✓				
Mindmedia Biotrace (exported)	✓	✓	✓		✓	✓	✓	✓	✓	✓
Brain Vision	✓	✓	✓		✓	✓	✓	✓	✓	✓
Windaq (wdq)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Observer XT compatible	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
NeuroScan	✓	✓	✓		✓	✓	✓	✓	✓	✓
BioSemi	✓	✓	✓		✓	✓	✓	✓	✓	✓
Eyelink								✓	✓	✓

30.2 Futher settings

Data format	Datatype	File extension	Manufacturer	Import multiple channels	Search channel names	Automarkers	Ask for sampling rate
CED Spike	spike	.smr	CED	✓	✓		
Matlab	mat	.mat		✓			✓
Text	txt	.txt		✓	✓		
Biopach AcqKnowledge (≤ v3.9.0)	acq	.acq	Biopac	✓	✓		
Biopac AcqKnowledge (exported)	acqmat	.mat	Biopac	✓	✓		
Labchart exported (≤ v7.1)	labchartmat_ext	.mat	ADInstruments	✓	✓	✓	✓
Labchart exported (≥ v7.2)	labchartmat_in	.mat	ADInstruments	✓		✓	
VarioPort	vario	.vpd	Becker MediTec	✓	✓	✓	
Biograph Infiniti (exported)	biograph	.txt	Thought Technology				
Mindmedia Biotrace (exported)	biotrace	.txt	MindMedia			✓	
Brain Vision	brainvision	.eeg	BrainProducts	✓	✓	✓	
Windaq (wdq)	windaq	.wdq	Dataq	✓			
Observer XT compatible	observer	.any	Noldus	✓	✓		
NeuroScan	cnt	.cnt		✓	✓	✓	
BioSemi	biosemi	.bdf		✓	✓	✓	
Eyelink	txt	.asc		✓		✓	

30.3 How to

30.3.1 CED Spike

Import the original acquisition files

30.3.2 Text

Text files can only contain numbers (i.e. no header lines with channel names) and one data column per channel. Make sure you use the decimal point (i.e. not decimal comma as used in some non-English speaking countries). At the moment, no import of event markers is possible

30.3.3 Matlab

Each input file must contain a variable called data that is either a cell array of column vectors, or a data points $\times$ channels matrix. At the moment, no import of event markers is possible. Data structures containing timestamps cannot be imported at the moment; rather, PsPM will ask you for a sample rate.

30.3.4 Biopac AcqKnowledge

Import the original acquisition files

30.3.5 exported Biopac AcqKnowledge

30.3.6 Labchart exported (up to v. 7.1)

Export data to matlab format (plugin for the LabChart software, available from www.adinstruments.com)

30.3.7 Labchart exported (v. 7.2 and higher)

30.3.8 VarioPort

Import the original acquisition files

30.3.9 Biograph Infiniti exported

Export data to text format, both "Export Channel Data" and "Export Interval Data" are supported; a header is required

30.3.10 Mindmedia Biotrace exported

30.3.11 BrainVision

Import the original acquisition files

30.3.12 Windaq (wdq)

30.3.13 Observer XT compatible

30.3.14 NeuroScan

30.3.15 BioSemi

30.3.16 Eyelink

31 GUI

Contributed by Gabriel Gräni.

31.1 Matlabbatch: Getting started

1. Add the trunk folder to the matlab path
2. Type scr_init into the command window (after the execution of the command the folders scr_cfg and matlabbatch should be added to the matlab path)
3. Start matlabbatch by the typing cfg_ui into the command window
4. If the item SCR exists in the menu bar of matlabbatch you can skip steps 5 to 7 and continue at step 8
5. Select –> File –> Add Application
6. Navigate to the folder scr_cfg on the left hand side of the window and select the file scr_cfg.m on the right hand side –> Press the button Done
7. A new item, called SCR, will appear in the upper menu bar.
8. By selecting SCR the desired action can be selected (at the moment, there is only Data Preparation –> {Import, Trim} available)

31.1.1 Example Function: Trim

This example demonstrates how matlabbatch can be used to execute a function. For all other functions matlabbatch behaves in the same manner.

- Select a file by pressing the Select Files Button (under Datafile)
- Select Reference and choose an item in the lower part of the window
- Fill in the desired values in the fields which are marked with "<-X"
- After you have chosen a file and filled in all values correctly, you will see a green arrow on the upper left part of the window
- By pressing on the green arrow the selected file will be trimmed according to the filled in values

31.2 Matlabbatch: How to

31.2.1 Preliminaries

- Add folder of matlabbatch to the matlab path
- Add first application and then load the batch in order to execute a function

31.2.2 Some notes for creating a new application

- Leafs (items) are specified first
- Assigning child items to .val or .values fields of their parent items
- Root node of a tree is specified last
- Some examples of items:
  - cfg_item:
    ```
    item1= cfg_item; % Defines generic configuration item
    item1.name = 'Def 1'; % The display name
    item1.tag    = 'def1'; % The name appearing in the harvested
    job
    % structure.  This name must be unique
    % among all items in the val field of the
    % superior node
    item1.val = {true}; % Value of item (optional)
    item1.help = {'Help...'}; % Help text
    ```
  - cfg_entry:
    ```
    entry1 = cfg_entry; % Defines entry configuration item
    entry1.name = 'Input';
    entry1.tag = 'input';
    entry1.strtye = 'r'; % Type of values which can be entered

    entry1.num = [1 1]; % Expected dimension of the input
    entry1.help = {'Help...'};
    ```
  - cfg_choice:
    ```
    choice = cfg_item; % Defines choice configuration item
    choice.name = 'Choice';
    choice.tag = 'choice';
    choice.values = {item1, entry1}; % Defines which items will
    be
    % selectable in the choice menu.
    choice.help = {'Help...'};
    ```
  - cfg_exbranch:
    ```
    fct = cfg_exbranch; % Defines the branch that has information

    % about how to run this module
    ```

```
fct.name = 'Trim';
fct.tag = 'trim';
fct.val = {choice}; % The items that belong to this branch.

% All items must be filled before this
% branch can run or produce virtual
% outputs
fct.prog = @cfg_run_fct; % A function handle that will be
called
% with the harvested job to run the
% computation
trim.vout = @cfg_vout_fct; % A function handle that will be

% called with the harvested job to
% determine virtual outputs
trim.help = {Help...'};
```

  - There exists a number of other item classes. Here is a list of the most important classes: cfg_item, cfg_entry, cfg_choice, cfg_menu, cfg_exbranch, cfg_files, cfg_branch, cfg_repeat

    For more information call the help function in matlab (e.g. help help cfg_item)

- Note:
  The inputs to each module have to be described in a tree-like structure. During data entry, there is no way to change the tree structure based on input data. Add application to the configuration tree by default

31.2.3 Add application to the configuration tree by default

In the following it is shown how an application can be added to the menu bar of matlabbatch by default (without adding it every time matlabbatch is started)

- Start matlabbatch and add the appliaction cfg_confgui in the folder matlabbatch/cfg_confgui
- Put Generate code into the Module list by selecting ConfGUI –> Generate code in the menu bar
- Fill out all the input fields on the right side:
  - Output filename: This file will contain the whole menu structure, validity contraints and links to run time code of the appliaction.
  - Output directory: All files which are created by the ConfGUI will be stored into this directory (chose a directory which is added to the matlab path)

  - Root node of config: Name of the root node of the appliaction's configuration tree
  - Options:
    1. Create Defaults File: Yes
    2. Create mlbatch_appcfg File: Yes
    3. Create Code for addpath(): No

- Finally press the green arrow on the upper left side of the batch editor
- As no error occurred 3 new files ({Output filename}.m, {Output filename}_def.m, cfg_mlbatch_appcfg.m) should be created and added into the folder {Output directory}.
- Each time matlabbatch is started, it will search for any cfg_mlbatch_appcfg.m file (this file contains the names of the configuration files) and will add the corresponding application to the batch editor.

31.2.4 Add modules to module list

Example: Module Import and Trim will be added to the module list

```
arg1 = 'scr.prep.import_data';
arg2 = 'scr.prep.trim';
mod_cfg_id1 = cfg_util('tag2mod_cfg_id',arg1);
mod_cfg_id2 = cfg_util('tag2mod_cfg_id',arg2);
cjob = cfg_util('initjob');
mod_job_id1 = cfg_util('addtojob', cjob, mod_cfg_id1);
mod_job_id2 = cfg_util('addtojob', cjob, mod_cfg_id2);
cfg_util('harvest', cjob, mod_job_id1);
cfg_util('harvest', cjob, mod_job_id2);
cfg_ui('local_showjob', findobj(0,'tag','cfg_ui'), cjob);
```

31.2.5 Changes

- In the function cfg_onscreen at line 36 figure(fg); is commented out in order to prevent the appearance of the GUI for a short time if the function cfg_ui('Visible', 'off') is called.

31.3 Matlabbatch: changing help texts and fieldnames

31.3.1 File structure of matlabbatch GUI

There exist two files per function: 1 configuration file and 1 run file. The configuration file defines the structure of the corresponding function in the matlabbatch GUI whereas the run file firstly gathers all entered values and secondly calls the corresponding SCR function. Both types of files are located in the subfolder scr_cfg. The name of a configuration or a run file consists of two parts. The prefix of a configuration filename is called scr_cfg_ whereas the filename of a run file begins with scr_cfg_run. The second part of the filename is named after the function name (eg. for the function scr_import.m -> scr_cfg_import.m, scr_cfg_run_import.m).

31.3.2 Edit help texts and fieldname

In order to change any help text or fieldname in a matlabbatch GUI function the corresponding configuration file has to be opened. For each item in a matlabbatch GUI function a struct variable which contains several struct fields is defined in the configuration file.

- Help text The field .help defines the help text of the item which can be edited in order to change the help text. As soon as matlabbatch has been closed and opened again, the changes in the help text will be visible in matlabbatch GUI.
- Fieldname The fieldname of an matlabbatch GUI item is defined by the struct field .tag . In case a fieldname of an item should be changed be careful to verify if no other item, which has the same root node, hold the same fieldname. Otherwise matlabbatch will not work properly. After the fieldname of an item has been changed the run file (scr_cfg_run_functionname.m) of the corresponding function has to be adapted as well in order to ensure that the function call in the run file is done properly.

32 Test Environment

Contributed by Linus Rüttimann.

32.1 Unittest: General implementation

In PsPM the Matlab Unit Testing Framework is used for testing of functions. For each tested function there is a Matlab class with the name 'functionname_test', which contains the unittests for that specific function. Additionally there is a documentation page for each of the test classes, where information about the unittests can be found.

To run the unittests of a test class, an object of the class has to be created:

testCase = functionname_test

where 'testCase' is an arbitrary object name and 'funtionname_test' is the name of a test class. Then all the unittest that are contained in the test class can be run with:

testCase.run

A specific unittest can be run with:

testCase.run(‘unittest_name’)

Remember that a new test class object must be generated each time the test class has been changed.

32.2 Testcases: scr_add_channel

32.2.1 Information

Testclass: scr_find_channel_test

Function: sts = scr_add_channel(fn, newdata, msg)

32.2.2 Testcases

Invalid input arguments

Function name: invalid_input(this)

Description: Checks for warnings, if the input arguments are invalid.

Setup:

```
newdata.header.sr = 100;
```

Tests:

Input	Expected warning
scr_add_channel(’fn’) [invalid fn, no newdata]	ID:invalid_input
scr_add_channel(’fn’, ’foo’) [invalid fn and newdata]	ID:invalid_input
scr_add_channel(newdata, newdata) [no fn]	ID:invalid_input

Valid input arguments

Function name: valid_input(this)

Description: Generates a test channel and passes it to scr_add_channel.

Tests:

1. Test if function runs throught without any warnings
2. Test if ’sts’ is equal 1
3. Test if ’newdata’ really has been added to the dataset

(a) Test if 'scr_load_data' is successfull
(b) Test if 'msg' has been added to infos.history
(c) Test if 'data' has 2 channels

32.3 Testcases: scr_find_channel

32.3.1 Information

Testclass: scr_find_channel_test

Function: chan = scr_find_channel(headercell, chantype)

32.3.2 Testcases

Invalid input arguments

Function name: invalid_inputargs(this)

Description: Checks for warnings, if the input arguments are invalid.

Setup:

```
headercell = {'heart', 'scr', 'pupil'};
```

Tests:

Input	Expected warning
scr_find_channel('str','scr') [no headercell]	ID:invalid_input
scr_find_channel(headercell, 'str')	ID:not_allowed_channeltype
scr_find_channel(headercell, 4) [no string chantype]	ID:invalid_input

Valid Input Arguments

Function name: valid_inputargs(this)

Description: Checks for correct return value if the input arguments are valid

Setup:

```
headercell = {'heart', 'scr', 'pupil', 'mark', 'gsr', 'eda'};
```

Tests:

Input	Exp. Output	Expected warning
scr_find_channel(headercell, 'pupil')	3	
scr_find_channel(headercell, 'resp')	0	ID:no_matching_channels
scr_find_channel(headercell, 'scr')	-1	ID:multiple_matching_channels
scr_find_channel(headercell, {'mark', 'str', 'bla'})	4	
scr_find_channel(headercell, {'call', 'str', 'me'})	0	no matching channel, but no warning
scr_find_channel(headercell, {'scr', 'gsr', 'eda'})	-1	multiple matching channels, but no warning

32.4 Testcases: scr_get_ecg

32.4.1 Information

Testclass: scr_get_ecg_test
Function: [sts, data] = scr_get_ecg(import)

32.4.2 Testcases

Test

Function name: test(this)
Description: Test if all fields are returned correctly

Tests:

1. Test if 'sts' is equal 1.
2. Test if data.data is equal import.data
3. Test if data.header.chantype is 'ecg'
4. Test if data.header.units is equal import.units
5. Test if data.header.sr is equal import.sr

32.5 Testcases: scr_get_events

32.5.1 Information

Testclass: scr_get_events_test
Function: [sts, import] = scr_get_events(import)

32.5.2 Testcases

Check warnings

Function name: check_warnings(this)
Description: Checks for warnings, if the field '.markers' is missing or contains

invalid content.

Tests:

Input	Expected warning
Missing marker field	ID:nonexistent_field
import.marker = ‘foo’	ID:invalid_field_content

Timestamps

Function name: timestamps(this)
Description: Checks for correct output if the input is timestamp data

Tests:

1. Test if ‘sts’ is equal 1.
2. Test if the length of the output data is equal to the length of the input data

Continuous

Function name: continuous(this)
Description: Checks for correct output if the input is continuous data

Tests:

1. Perform three tests with different settings

   Tests:

   (a) Test if ‘sts’ is equal 1.
   (b) Test if the length of the field ’markerinfo’ is equal to the length of the output data.
   (c) Test if the length of the output data is equal to the expected number of pulses in the input data.

   Settings:

   (a) flank = ‘both’ (default)
   (b) flank = ’both’ & data offset 50
   (c) flank = ‘ascending’
   (d) flank = ‘descending’
   (e) inverted input signal
   (f) signal with angular flanks
   (g) check with

2. Additional test for setting (b): Test if data offset has been removed in the output data.
3. Additional test for setting (c) and (d): Test if positions returned by output data correspond to flank changes in the input data.
4. Test if markerinfo is not set if it has been set before.

32.6 Testcases: scr_get_eyelink

32.6.1 Information

Testclass: scr_get_eyelink_test
Function: [sts, data] = scr_get_eyelink(import)

32.6.2 Methods

verify_basic_data_structure

Function name: verify_basic_data_structure(this, data, sourceinfo, channel_types)
Description: Tests if the returned data structre is valid and match a given expected pattern.

Tests:

1. Test if all channels are numeric
2. Test if recorded time and date have a valid format
3. Test if blink channels have correct unit
4. Test if pupil channels have either 'diameter' or 'area' as unit
5. Test if channels labeled with 'position' have unit 'pixel'
6. Test if channels labeled with 'blink' have unit 'blink'

32.6.3 Testcases

test_two_eyes

Function name: test_two_eyes(this)
Description: Test if the returned data structure fits into the pattern of a two eyes data set.

Tests:

1. Create an import data set and the expected channel data set an pass it to 'verify_basic_data_structure()'

test_one_eye

Function name: test_one_eye(this)
Description: Test if the returned data structure fits into the pattern of a one eye data set.

Tests:

1. Create an import data set and the expected channel data set an pass it to 'verify_basic_data_structure()'

32.7 Testcases: scr_get_hb

32.7.1 Information

Testclass: scr_get_hb_test
Function: [sts, data] = scr_get_hb(import)

32.7.2 Testcases

Test

Function name: test(this)
Description: Test if all fields are returned correctly

Tests:

1. Test if 'sts' is equal 1.
2. Test if data.data is equal import.data
3. Test if data.header.chantype is 'hb'
4. Test if data.header.units is 'events'
5. Test if data.header.sr is 1

32.8 Testcases: scr_get_hr

32.8.1 Information

Testclass: scr_get_hr_test
Function: [sts, data] = scr_get_hr(import)

32.8.2 Testcases

Test

Function name: test(this)
Description: Test if all fields are returned correctly

Tests:

1. Test if 'sts' is equal 1.
2. Test if data.data is equal import.data
3. Test if data.header.chantype is 'hr'
4. Test if data.header.units is equal import.units
5. Test if data.header.sr is equal import.sr

32.9 Testcases: scr_get_marker

32.9.1 Information

Testclass: scr_get_marker_test
Function: [sts, data] = scr_get_marker(import)

32.9.2 Testcases

Test

Function name: test(this)
Description: Test if all fields are returned correctly

Tests:

1. Test if 'sts' is equal 1.
2. Test if data.data is equal import.data
3. Test if data.header.chantype is 'marker'
4. Test if data.header.units is 'events'
5. Test if data.header.sr is 1

32.10 Testcases: scr_get_pupil

32.10.1 Information

Testclass: scr_get_pupil_test
Function: [sts, data] = scr_get_pupil(import)

32.10.2 Testcases

Test

Function name: test(this)
Description: Test if all fields are returned correctly

Tests:

1. Test if ‘sts’ is equal 1.
2. Test if data.data is equal import.data
3. Test if data.header.chantype is ‘pupil’
4. Test if data.header.units is equal import.units
5. Test if data.header.sr is equal import.sr

32.11 Testcases: scr_get_resp

32.11.1 Information

Testclass: scr_get_resp_test
Function: [sts, data] = scr_get_resp(import)

32.11.2 Testcases

Test

Function name: test(this)
Description: Test if all fields are returned correctly

Tests:

1. Test if ‘sts’ is equal 1.
2. Test if data.data is equal import.data
3. Test if data.header.chantype is ‘resp’
4. Test if data.header.units is equal import.units
5. Test if data.header.sr is equal import.sr

32.12 Testcases: scr_get_scr

32.12.1 Information

Testclass: scr_get_scr_test
Function: [sts, data] = scr_get_scr(import)

32.12.2 Testcases

There are three test functions. One for the case that no transfer parameters are defined, one for the case that the transfer parameters are defined in a struct and one for the case that they are defined in a .mat file. They are all performing the following tests, plus eventually some individual tests

Tests:

1. Test if ‘sts’ is equal 1.
2. Test if the field data.data exists
3. Test if the field data.data is not empty
4. Test if the field data.header.units exists
5. Test if the field data.header.sr exists
6. Test if the field data.header.chantype exists
7. Test if data.header.sr is equal import.sr
8. Test if data.header.chantype is ‘scr’

No transfer parameters

Function name: no_transferparams(testCase)
Description: Test if all fields are returned correctly, if no transfer parameters are defined.

Additional Tests:

No additional tests

Struct transfer parameters

Function name: stuct_transferparams(testCase)
Description: Test if all fields are returned correctly, if the transfer parameters are defined in a struct.

Additional Tests:

1. Check for warning if the conversion constant (import.transfer.c) is not defined
2. Checks that there are no warnings if import.transfer.Rs or import.transfer.offset is not defined.

File transfer parameters

Function name: file_transferparams(testCase)
Description: Test if all fields are returned correctly, if the transfer parameters are defined in a .mat file.

Additional Tests:

1. Check for warning if the transfer parameter file doesn’t exist.

32.13 Testcases: scr_get_timing

32.13.1 Information

Testclass: scr_get_timing_test
Function: [sts, multi] = scr_get_timing('onsets', intiming, timeunits)
[sts, epochs] = scr_get_timing('epochs', epochs)
[sts, epochs] = scr_get_timing('events', events)

32.13.2 Testcases

Invalid input arguments

Function name: invalid_inputargs(this)
Description: Checks for warnings, if the input arguments are invalid.

Tests:

Input	Expected warning
scr_get_timing('epochs') [missing input var]	ID:invalid_input
scr_get_timing('onsets', 'str') [no timeunits var]	ID:invalid_input
scr_get_timing('foo') [unknown format]	ID:invalid_input
scr_get_timing('onsets', intiming, 'samples') [two sessions with nonmatching number of conditions]	ID:number_of_elements_dont_match
scr_get_timing('onsets', intiming, 'samples') [two sessions with nonmatching condition names]	ID:event_names_dont_match
scr_get_timing('onsets', intiming, 'samples') [intiming.onsets{1} is no numeric vector]	ID:no_numeric_vector
scr_get_timing('epochs', fn_mat, 'samples') [epochs is not an integer array]	ID:no_integers

Case Epochs

Function name: case_epochs(this)
Description: Checks the function in 'epochs' mode.
Function: [sts, epochs] = scr_get_timing('epochs', epochs)

Test 1 (matfile input)

Input: mat file with variable: epochs = [1 4; 2 5; 3 6]

Check if sts==1 and if the return value is equal the input array.

Test 2 (spm input)

Input: mat file with variable: onsets{1} = [1 2 3]';onsets{2} = [4 5 6]';

Check if sts==1 and if the return value is equal [onsets{1}, onsets{2}].

Test 3 (textfile input)

Input: textfile with variable: epochs = [1 4; 2 5; 3 6]

Check if sts==1 and if the return value is equal the input array.

Test 4 (matrix input)

Input: matix: epochs = [1 4; 2 5; 3 6]

Check if sts==1 and if the return value is equal the input array.

Case onsets

Function name: case_onsets(this)
Description: Checks the function in 'onsets' mode.
Function: [sts, multi] = scr_get_timing('onsets', intiming, timeunits)

Test 1

Input: mat file with the variables:
names = {'name1', 'name2'};
onsets = {[1 2], [3 4]};
pmod.name = {'name3', 'name4'};
pmod.param = {[2 3], [4 5]};
pmod.poly = {2, 2};
save(fn_mat, 'names', 'onsets', 'pmod');

Function call:
[sts, outtiming] = scr_get_timing('onsets', fn_mat, 'samples');

Tests:
Check if sts==1, if onsets and names are unchanged and if
outtiming.pmod.param == {[2 3], [4 9], [4 5], [16 25]}

Test 2

Input:
mat file with the variables: names = {'name1', 'name2'};
onsets = {[1 2 3], [3 4 5]}; durations = {[3 4 5]', [5 6 7]'};
pmod.name = {'name3', 'name4'};
pmod.param = {[2 3 4], [4 5 6]};

pmod.poly = {2, 1};

Function call:
[sts, outtiming] = scr_get_timing('onsets', fn_mat, 'samples');

Tests:
Check if sts==1, if onsets,names and durations are unchanged and if outtiming.pmod.param == {[2 3 4], [4 9 16], [4 5 6]}

Case events

Function name: case_events(this)
Description: Checks the function in 'events' mode.
Function: [sts, epochs] = scr_get_timing('events', events)

Check the function if input is a one element cell array and a multiple element cell array. Check for warnings (ID:invalid_vector_size) if elements have more than two columns and if not all elements have the same number of rows.

32.14 Testcases: scr_get_<datatype>

32.14.1 Information

The datatype import functions are all tested in a similar way. The individual testclasses must inherit the class 'scr_get_superclass', from which they inherit the main test function 'valid_datafile'. They also have to implement the property 'fhandle', which is a function handle to the specific import function.

The tests are performed with the sampledata files that are listed in the SampleDataMasterList.docx file (as at 18.11.2013).

Superclass: scr_get_superclass

Testclasses: scr_get_acq_test
scr_get_acqmat_test
scr_get_biograph_test
scr_get_biosemi_test
scr_get_biotrace_test
scr_get_brainvis_test
scr_get_labchartmat_ext_test
scr_get_labchartmat_in_test
scr_get_mat_test
scr_get_obs_test
scr_get_spike_test
scr_get_superclass
scr_get_txt_test

scr_get_vario_test
scr_get_eyelink_test

Function: [sts, import, sourceinfo] = scr_get_<datatype>(datafile, import)

32.14.2 Setup

define testcases

In this method the testcases are defined and the testdata is generated (if needed). Each testcase is a cell in the cellarray 'testcases'. Each testcase has the following fields:

- .pth: the path to the samplefile
- .import: the input variable

For datatypes which support blocks there has to be an additional field:

- .numofblocks

32.14.3 Testcases

Valid datafile

Function name: valid_datafile(this)

Description: The main test function, for tests with valid inputdata. It tests all testcases equally.

Tests:

1. Test if 'sts' is equal 1.
2. If the datatype supports blocks, test if the number of blocks is correct.
3. Test if number of elements of the returned 'import' variable is correct.
4. Test if each importjob has a field 'data', that is a numeric vector.
5. Test if each importjob has a field 'sr', that is a number.
6. Test if each importjob has a field 'type'.
7. Test if all event importjobs have a field 'marker'.
8. Test if all importjobs have duration below 1h.
9. Test if all importjobs have a samplerate between 1 and 10000 for continuous channels or between 10^-6 and 1 for timestamp channels.

invalid datafile

Function name: invalid_datafile(this)
Description: The main test function, for tests with invalid inputdata.

Tests:

If the datatype supports multiple channels: Check for warning when trying to import a channel, that is not contained in the file ('ID:channel_not_contained_in_file').

32.15 Testcases: scr_glm

32.15.1 Information

Testclass: scr_glm_test
Function: glm = scr_glm(model, options)

There are seven testcase functions. One invalid input arguments test and test 1 to 6. Tests 1 to 5 are of the same kind. There are one or multiple testcases per test function, have a look at the testcase description for more information. In these tests only kronecker delta functions are used as basis functions, furthermore all conditions, pmods and nuisance regressors are pairwise orthogonal. The data is also not down sampled and not filtered. With these limitations it's easy to calculate the datavectors and the expected stats. For each testcase it is then tested:

- If numel(glm.names) has the expected value.
- If numel(glm.stats) has the expected value.
- If glm.stats has the expected value (with a tolerance of 1%).

In test 6 the default basis functions are used, and not all conditions and pmods are orthogonal. The data is down sampled and low and high pass filtered. In exchange the stats are not tested for correct values, just for the correct number of elements.

32.15.2 Testcases

Invalid input arguments

Function name: invalid_input (this)
Description: Checks for warnings, if the input arguments are invalid.

Tests:

Input	Expected warning
scr_glm(model) [no timeunits field]	ID:invalid_input
scr_glm(model) [no timeunits var]	ID:invalid_input
scr_glm(model) with model.timeunits = ‘foo’ [no valid timeunits field]	ID:invalid_input
scr_glm(model) with model.timing = zeros(10,2) [no valid timing field]	ID:invalid_input
scr_glm(model) with model.modality = ’foo’ [no valid modality field]	ID:invalid_input
scr_glm(model) with model.channel = ’scr’ [no valid channel field]	ID:invalid_input
scr_glm(model) with model.norm = ’no’ [no valid norm field]	ID:invalid_input
scr_glm(model) with model.filt.down = ’none’ [filt.down is not numeric]	ID:invalid_input
scr_glm(model) with model.bf.fhandle = ’foohandle’ [non existing bf]	ID:invalid_fhandle
scr_glm(model) with numel(model.datafile) != numel(model.timing)	ID:number_of_elements_dont_match
scr_glm(model) with model.missing is struct [non valid missing field]	ID:invalid_input
scr_glm(model) with numel(model.datafile) != numel(model.missing)	ID:number_of_elements_dont_match
scr_glm(model) with model.nuisance is struct [non valid nuisance field]	ID:invalid_input
scr_glm(model) with numel(model.datafile) != numel(model.nuisance)	ID:number_of_elements_dont_match
scr_glm(model) with no R variable in the nuisance file	ID:invalid_input
scr_glm(model) with R variable in the nuisance file that has not the same length as the datafile	ID:number_of_elements_dont_match

Test 1

Function name: test1(this)
Description: Basic test with one basis function, one session, no nuisance regressors, no missings and one condition. Timeunits are seconds.

Testcases:

1. no pmods

2. one pmod
3. two pmods

Test 2

Function name: test2(this)
Description: Test with one basis function, one session, no nuisance regressors, no missings and two conditions. Timeunits are seconds.

Testcases:

1. no pmods
2. first condition: no pmods; second condition: one pmod
3. first condition: one pmod; second condition: two pmods

Test 3

Function name: test3(this)
Description: Test with one basis function, one session, two nuisance regressors (1Hz cosinus, 1Hz sinus), no missings, one condition and no pmods. Timeunits are seconds.

Testcases:

Only one testcase.

Test 4

Function name: test4(this)
Description: Test with one basis function, two sessions, no nuisance regressors, no missings and one condition.

Testcases:

1. timeunits are seconds
2. timeunits are samples
3. timeunits are markers

Test 5

Function name: test5(this)
Description: Test with two basis functions, one session, no nuisance regressors and one condition. Timeunits are seconds.

Testcases:

1. no missings
2. with missings

Test 6

Function name: test6(this)
Description: Test with default basis function and non-orthogonal conditions and pmods

Testcase:

Default basis functions, no nuisance regressors, no missings, two sessions and two conditions. Timeunits are seconds.

- first condition: two pmods (with pmod(1).poly{1} = 2 and pmod(1).poly{2} = 3)
- second condition: no pmods

32.16 Testcases: scr_import

32.16.1 Information

Testclass: scr_import_test
Function: outfile = scr_import(datafile, datatype, import, options)

32.16.2 Testcases

Invalid input arguments

Function name: invalid_inputargs(ths)
Description: Checks for warnings, if the input arguments are invalid.

Tests:

Test No.	Input	Expected warning
1	scr_import(datafile, datatype) [no import variable]	ID:invalid_input
2	scr_import(datafile, datatype, ‘foo’) [no cell/struct import var.]	ID:invalid_input
3	scr_import(datafile, ‘foo’, import) [invalid channeltype]	ID:invalid_channeltype
4	scr_import(5, datatype, import) [no char filename]	ID:invalid_input

Invalid import variable structure

Function name: invalid_import_struct(this)
Description: Checks for warnings, if the structure of the import variable is invalid.

Tests:

Test No.	Input	Expected warning
1	Multiple channel, though not supported	ID:ivalid_import_struct
2	Not allowed channeltype	ID:ivalid_import_struct
3	No sr given, though autosr is not supported	ID:ivalid_import_struct
4	Nonexistent file	ID:nonexistent_file

One datafile

Function name: one_datafile(this)
Description: Checks the function, if datafile is a string (import of one datafile) and all inputs are correct. The outfile is checked with the scr_load_data function. The tests are performed with a spike samplefile and a labchartmat_in samplefile (to check the handling of blocks).

Multiple datafiles

Function name: multiple_datafiles(this)
Description: Checks the function, if datafile is a cell array of strings (import of multiple datafiles) and all inputs are correct. The outfiles are tested with the scr_load_data function.

32.17 Testcases: scr_load_data

32.17.1 Information

Testclass: scr_load_data_test
Function: [sts, infos, data, filestruct] = scr_load_data(fn, chan)

32.17.2 Setup

If not otherwise declared, the input variable fn is referring to a datafile which was generated with scr_testdata_gen and consists out of the following channels:

```
data{1}.chantype = 'scr';
data{2}.chantype = 'marker';
data{3}.chantype = 'hr';
data{4}.chantype = 'hb';
data{5}.chantype = 'marker';
data{6}.chantype = 'resp';
```

```
data{7}.chantype = 'scr';
```

The duration of the channels is 10s.

32.17.3 Testcases

Invalid input arguments

Function name: invalid_inputargs(testCase)
Description: Checks for warnings, if the input arguments are invalid.

Tests:

Input	Expected warning
scr_load_data [no filename]	ID:invalid_input
scr_load_data(1) [no char filename]	ID:invalid_input
scr_load_data(fn, -1) [neg. channel no]	ID:invalid_input
scr_load_data(fn, 'foobar') [no allowed ch type]	ID:invalid_input
scr_load_data(fn, foo) [missing field in foo struct]	ID:invalid_input
scr_load_data(fn, {1}) [invalid channel option]	ID:invalid_input
scr_load_data(stuct) [struct has no infos field]	ID:invalid_input

Invalid datafile

Function name: invalid_datafile(testCase)
Description: Checks for warnings, if the datafile is invalid.

Tests:

Test No.	Input	Expected warning
1	non-existent datafile	ID:nonexistent_file
2	missing 'infos' variable	ID:invalid_data_structure
3	missing 'data' variable	ID:invalid_data_structure
4	missing 'data' field in 'data{2}'	ID:invalid_data_structure
5	missing 'header' field 'data{3}'	ID:invalid_data_structure
6	missing 'sr' field in 'data{7}.header'	ID:invalid_data_structure
7	data{4} is a nx2 vector (instead of a nx1 vector)	ID:invalid_data_structure
8	the length of data{1}.data is incompatible with the duration	ID:invalid_data_structure
9	An entry of data{2}.data is larger than 'duration'	ID:invalid_data_structure
10	data{5} has an non-existent chantype ('scanner')	ID:invalid_data_structure
11	duplicates (9) with struct chan input	ID:invalid_data_structure

Return all channels

Function name: valid_datafile_0(testCase)
Description: Checks the function, if all channels shall be returned (chan = 0).

Return all channels (struct input)

Function name: valid_datafile_1(testCase)
Description: Checks the function, if all channels shall be returned (chan = 0) and the input is a struct.

Return one channel

Function name: valid_datafile_2(testCase)
Description: Checks the function, if only one channel shall be returned (chan = 2).

Return one channel

Function name: valid_datafile_3(testCase)
Description: Checks the function, if multiple channels shall be returned (chan = [3 5]).

Return scr channels

Function name: valid_datafile_4(testCase)
Description: Checks the function, if only the scr channels shall be returned.

Return event channels

Function name: valid_datafile_5(testCase)
Description: Checks the function, if only the event channels shall be returned.

Save data

Function name: valid_datafile_6(testCase)
Description: Checks the function, if data is to be saved (chan struct).

32.18 Testcases: scr_pp

32.18.1 Information

Testclass: scr_pp_test
Function: newfile = scr_pp('median', datafile, n, channelnumber) or newfile = scr_pp('butter', datafile, freq, channelnumber)

32.18.2 Testcases

Invalid input

Function name: invalid_input(this)
Description: Checks for warnings, if the input arguments are invalid.

Tests:

Input	Expected warning
scr_pp('butter', 'file') [no freq]	ID:invalid_input
scr_pp('foo', 'file', 100) [no valid first argument]	ID:invalid_input
scr_pp('butter', 'file', 19) [freq below 20]	ID:invalid_input

Median test

Function name: median_test(this)
Description: Checks medianfilter functionality

Setup:

Testfile with 3 channels (scr, hb, scr).

Tests:

1. Filter one channel [Input: newfile = scr_pp('median', testfile, 50, 3)]
   i. Check if sts == 1, when data is loaded with scr_load_data.
   ii. Check if newfile has the same number of channels as testfile
2. Filter multiple channel [Input: newfile = scr_pp('median', testfile, 50)]
   i. Check if sts == 1, when data is loaded with scr_load_data.
   ii. Check if newfile has the same number of channels as testfile

Butterworth filter test

Function name: butter_test(this)
Description: Checks butterworth filter functionality

Setup:

Testfile with 3 channels (scr, hb, scr).

Tests:

1. Filter one channel [Input: newfile = scr_pp('butter', testfile, 40, 3)]
   i. Check if sts == 1, when data is loaded with scr_load_data.
   ii. Check if newfile has the same number of channels as testfile

2. Filter multiple channel [Input: newfile = scr_pp('butter', testfile, 40)]
   i. Check if sts == 1, when data is loaded with scr_load_data.
   ii. Check if newfile has the same number of channels as testfile

32.19 Testcases: scr_prepdata

32.19.1 Information

Testclass: scr_prepdata_test
Function: [sts, outdata, newsr] = scr_prepdata(data, filt)

32.19.2 Testcases

Invalid input

Function name: invalid_input(this)
Description: Checks for warnings, if the input arguments are invalid.

Tests:

Input	Expected warning
scr_prepdata([1 2 3]) [no filt variable]	ID:invalid_input
scr_prepdata(data, filt) [filt has no hporder field]	ID:invalid_input
scr_prepdata('foo', filt) [no numeric data]	ID:invalid_input
scr_prepdata(data, filt) [with lpfreq = 'foo' (not valid)]	ID:invalid_input

Hipassfilter test

Function name: hipassfilter_test(this)
Description: Checks hipassfilter functionality (without downsampling)

Setup:
data = rand(1000,1);

filt.sr = 100;
filt.lpfreq = 'none';
filt.lporder = 1;
filt.hpfreq = 20;
filt.hporder = 1;
filt.down = 'none';

Tests:

1. Unidirectional tests [filt.direction = 'uni']
   i. Check if sts == 1
   ii. Check if newsr == filt.sr
   iii. Check if outdata is empty

iv. Check if length(outdata) == length(data)

2. Unidirectional tests [filt.direction = 'bi']
   i. Check if sts == 1
   ii. Check if newsr == filt.sr
   iii. Check if outdata is empty
   iv. Check if length(outdata) == length(data)

Lowpassfilter test

Function name: lowpassfilter_test(this)
Description: Checks hipassfilter functionality (without downsampling)

Setup:
data = rand(1000,1);
filt.sr = 100;
filt.lpfreq = 40;
filt.lporder = 1;
filt.hpfreq = 'none';
filt.hporder = 1;
filt.down = 'none';

Tests:

Same tests as in hipassfilter_test. Additionally there is a check for a warning if filt.lpfreq is higher (or equal) than the nyquist frequency:

Input	Expected warning
scr_prepdata(data, filt) [filt.sr = 100; filt.lpfreq = 60]	ID:no_low_pass_filtering

Bandpassfilter test

Function name: bandpassfilter_test(this)
Description: Checks bandpassfilter functionality (without downsampling)

Setup:
data = rand(1000,1);
filt.sr = 200;
filt.lpfreq = 99;
filt.lporder = 1;
filt.hpfreq = 20;
filt.hporder = 1;
filt.down = 'none';

Tests: Same tests as in hipassfilter_test.

Integer samplerate ratio downsampling test

Function name: int_sr_ratio_downsample_test(this)
Description: Checks downsampling functionality, if the ratio between filt.sr and filt.down is an integer.

Setup:

ratio = 2; %ratio between filt.sr and filt.down

filt.down = 100;
filt.sr = ratio
filt.down; filt.lpfreq = 40;
filt.lporder = 1;
filt.hpfreq = 'none';
filt.hporder = 1;
filt.direction = 'uni';

data = rand(filt.sr * 10,1);

Tests:

1. Check if sts == 1
2. Check if newsr == filt.down
3. Check if outdata is empty
4. Check if ratio*length(outdata) == length(data)

32.20 Testcases: scr_pulse_convert

32.20.1 Information

Testclass: scr_pulse_convert_test
Function: wavedata = scr_pulse_convert(pulsedata, resamplingrate, samplingrate)

32.20.2 Testcases

Invalid input

Function name: invalid_input(testCase)
Description: Pass invalid input arguments and test if the error message is correct.

Tests:

Input	Expected warning
scr_pulse_convert()	ID:invalid_input
scr_pulse_convert(10^-3 * (1:10000)')	ID:invalid_input
scr_pulse_convert(10^-3 * (1:10000)', 10000)	ID:invalid_input

Valid input

Function name: valid_input(testCase)
Description: Pass generated, valid data and test if function issues no warning.

Tests:

1. Test function without downsampling the data
2. Test function with downsampling the data

32.21 Testcases: scr_ren

32.21.1 Information

Testclass: scr_ren_test
Function: out_newfilename = scr_ren(filename, newfilename)

32.21.2 Testcases

Invalid input

Function name: invalid_input (this)
Description: Checks for warnings, if the input arguments are invalid.

Tests:

Input	Expected warning
scr_ren('fn') [no newfilename]	ID:invalid_input
scr_ren({'fn1', 'fn2'}, {'rfn1', 'rfn2', 'rfn3'}) [non same size cell arrays]	ID:invalid_input

Char Valid Input

Function name: char_valid_input (this)
Description: Checks the function if the input variables are of type char. It uses scr_load_data to check the files.

Tests:

1. Check if out_newefilename = newfilename
2. Check if sts==1 (of scr_load_data output)
3. Check if the field 'infos.rendata' exists
4. Check if the field 'infos.newname' exists

5. Check if the original file has been deleted

Cell Valid Input

Function name: cell_valid_input (this)
Description: Checks the function if the input variables are of type cell. It uses scr_load_data to check the files.

Tests:

The inputs are two-element cell arrays. For both elements the same tests as in the char_valid_input function are performed individually.

32.22 Testcases: scr_split_sessions

32.22.1 Information

Testclass: scr_split_sessions_test
Properties: expected_number_of_files = 3;
Function: newdatafile = scr_split_sessions(datafile, markerchannel, options)

32.22.2 Setup

For the tests a testdatafile with three channels is used (duration is 100s). The markerchannel data is:

data = [1 4 9 12 30 31 34 41 43 59 65 72 74 80 89 96]'

Hence if MAXSN=10 & BRK2NORM=3 (default values) the datafiles should be split into 3 files. If different values are being used, update the property 'expected_number_of_files' of the testclass object accordingly.

32.22.3 Testcases

Invalid input

Function name: invalid_input (this)
Description: Checks for warnings, if the input arguments are invalid.

Tests:

Input	Expected warning
scr_split_sessions() [no filename]	ID:invalid_input
scr_split_sessions (2) [no string filename]	ID:invalid_input
scr_split_sessions ('fn', 'foo') [no numeric marker channel no.]	ID:invalid_input

One datafile

Function name: one_datafile(this)
Description: Checks the function if the variable 'datafile' is of type char (one datafile). The markerchannel number is not assigned explicitly.

Tests:

1. Check if the file has been split into 'expected_number_of_files' files For each output file the following tests are performed:
2. Check if sts == 1, when data is loaded with scr_load_data.
3. Check if number of channels is correct.
4. Check it the field infos.slitdate exists
5. Check if the field infos.splitsn exists
6. Check if the field infos.splitfile exists.

Multiple datafiles

Function name: multiple_datafiles(this)
Description: Checks the function if the variable 'datafile' is of type cell (two datafiles). The markerchannel number is assigned explicitly.

Tests:

For both datafiles the same tests as in the one_datafile function are performed individually. Additionally it is tested if the number of input files does match the number of output files.

32.23 Testcases: scr_trim

32.23.1 Information

Testclass: scr_trim_test
Function: newdatafile=scr_trim(datafile, from, to, reference, options)

32.23.2 Setup

If not otherwise declared, the input variable fn is referring to a datafile which was generated with scr_testdata_gen and consists of the following channels:

data{1}.chantype = 'scr';
data{2}.chantype = 'marker';
data{3}.chantype = 'hr';
data{4}.chantype = 'hb';

data{5}.chantype = 'marker';
data{6}.chantype = 'resp';
data{7}.chantype = 'scr';

The duration of the data recording is 10s.

32.23.3 Testcases

Invalid input arguments

Function name: invalid_inputargs(testCase)
Description: Checks for warnings, if the input arguments are invalid.

Tests:

Input	Expected warning
scr_trim(testCase.fn, [1 2], 5, 'marker') [invalid from parameter]	ID:invalid_input
scr_trim(testCase.fn, 0, 'bla', 'marker') [invalid to parameter]	ID:invalid_input
scr_trim(testCase.fn, 0, '[]', 'marker') [invalid to parameter]	ID:invalid_input
scr_trim(fn, 0, 5) [no reference]	ID:invalid_input
scr_trim(fn, 0, 5, 6) [no char or 2-element numeric reference]	ID:invalid_input
scr_trim(fn, 0, 5, ‘bla’) [invalid char reference]	ID:invalid_input
scr_trim(fn, 0, 5, [-1 5]) [invalid numeric start reference]	ID:invalid_input
scr_trim(fn, 0, 5, [5 4]) [invalid numeric start/end reference]	ID:invalid_input

32.23.4 Reference = ‘marker’ tests

Function name: marker_tests(testCase)
Description: A wrapper function for tests with reference = ‘marker’. It executes the methods markertest_k, where the testcases are defined.

markertest_1

Description: from and to are set so that the trimming points are out of the range [0,duration]. Hence the data should not be trimmed.

Expected warning: ID: marker_out_of_range

Input: scr_trim(fn, -20, 20, 'marker')

markertest_2

Description: from and to are set so that the trimming points are exactly (0, duration). Hence the data should not be trimmed.

Input: from = -1 * marker(1) to = duration - marker(end) scr_trim(fn, from, to, 'marker')

markertest_3

Description: from and to are set so that the trimming points in the range [0,duration].

Input: scr_trim(fn, 1, -2, 'marker')

32.23.5 Reference = 'file' tests

Function name: file_tests(testCase)

Description: A wrapper function for tests with reference = 'file'. It executes the methods filetest_k, where the testcases are defined.

filetest_1

Description: from and to are set so that the trimming points are out of the range [0,duration]. Hence the data should not be trimmed.

Expected warning: ID: marker_out_of_range

Input: scr_trim(fn, -12.5, 50, 'marker')

filetest_2

Description: from and to are set so that the trimming points are exactly (0, duration). Hence the data should not be trimmed.

Input: scr_trim(fn, 0 , duration, 'marker')

filetest_3

Description: from and to are set so that the trimming points in the range [0,duration].

Input: scr_trim(fn,2.1, duration – 2.5, 'marker')

Numeric reference tests

Function name: num_tests(testCase)
Description: A wrapper function for tests with reference = [a b] (a, b are two integers with a<b). It executes the methods markertest_k, where the testcases are defined.

numtest_1

Description: from and to are set so that the trimming points are out of the range [0,duration]. Hence the data should not be trimmed.

Expected warning: ID: marker_out_of_range

Input: scr_trim(fn, -20, 20, [2 14])

numtest_2

Description: from and to are set so that the trimming points are exactly (0, duration). Hence the data should not be trimmed.

Input: from = -1 * marker(3) to = duration - marker(8) scr_trim(fn, from, to, [3 8])

numtest_3

Description: from and to are set so that the trimming points in the range [0,duration].

Input: scr_trim(fn, -1.5, 2, [2 7])

numtest_4

Description: Second reference point is out of the marker range; from is set to ‘none’. Hence the data should not be trimmed.

Expected warning: ID: marker_out_of_range

Input: scr_trim(fn, ‘none’, 0, [1 (numel(marker) + 1)])

Multiple file reference tests

Function name: multiple_files(testCase)

Description: The input variable datafile is either a cell array of two filenames or a cell array of two stucts. In both cases it is tested whether the return value is also a cell array of two filenames and whether both files are trimmed correctly.

Options tests

Marker channel number option

Function name: marker_chan_num_option_test(testCase)

Description: Tests if the option marker_chan_num is working correctly. There are two tests: Test 1: Checks for a warning if the selected channel is no marker channel. Test 2: Checks if the selected channel is actually used.

33 External functions and tools

33.1 VB (Variational Bayes) inversion algorithm by Jean Daunizeau

Updated October 2014

Changes made for use in PsPM:

VBA_ReDisplay.m, fixed try-catch syntax in various places iby adding a comma after "try" to avoid warning in matlab > 2007

VBA_inv.m, line 42: added warning off/on to suppress the warning "Matrix is singular, close to singular or badly scaled. Results may be inaccurate. RCOND = NaN."

34 List of functions

Name	Main author	Test exists	Test Doc
f_SCR	Dominik Bach & Jean Daunizeau	-	-
f_SF	Dominik Bach	-	-
g_SCR	Dominik Bach	-	-
pspm	Dominik Bach	x	x
scr	Dominik Bach	x	x
scr_add_channel	Dominik Bach	x	-
scr_align_channels	Dominik Bach	-	-
scr_axpos	Dominik Bach	-	-
scr_bf_brf	Saurabh Khemka & Dominik Bach	-	-
scr_bf_FIR	Dominik Bach	-	-
scr_bf_Fourier	Dominik Bach	-	-
scr_bf_hprf	Dominik Bach	-	-
scr_bf_scrf_f	Dominik Bach	-	-
scr_bf_scrf	Dominik Bach	-	-
scr_butter	Dominik Bach	-	-
scr_con1	Dominik Bach	-	-

scr_con2	Dominik Bach	-	-
scr_contrast	Dominik Bach	-	-
scr_dcm_inv	Dominik Bach	-	-
scr_dcm	Dominik Bach	-	-
scr_denoise_spike	Dominik Bach	-	-
scr_display	Philipp C Paulus	-	-
scr_down	Dominik Bach	-	-
scr_downsample	Dominik Bach	-	-
scr_ecg2hb	Philipp C Paulus	-	-
scr_exp	Dominik Bach	-	-
scr_filtfilt	Dominik Bach	-	-
scr_find_channel	Dominik Bach	x	x
scr_get_acq	Dominik Bach	x	x
scr_get_acqmat	Dominik Bach	x	x
scr_get_biograph	Dominik Bach	x	x
scr_get_biosemi	Dominik Bach	x	x
scr_get_biotrace	Dominik Bach	x	x
scr_get_brainvis	Dominik Bach	x	x
scr_get_cell	Dominik Bach	-	-
scr_get_cnt	Dominik Bach	-	-
scr_get_custom	Tobias Moser	-	-
scr_get_ecg	Dominik Bach	x	x
scr_get_events	Dominik Bach	x	x
scr_get_eyelink	Christoph Korn, Tobias Moser	x	x
scr_get_hb	Dominik Bach	x	x
scr_get_hp	Dominik Bach	-	-
scr_get_hr	Dominik Bach	x	x
scr_get_labchartmat_ext	Dominik Bach	x	x
scr_get_labchartmat_in	Dominik Bach	x	x
scr_get_marker	Dominik Bach	x	x
scr_get_markerinfo	Dominik Bach	-	-
scr_get_mat	Dominik Bach	x	x
scr_get_obs	Linus Rüttimann	x	x
scr_get_pupil	Dominik Bach	x	x
scr_get_resp	Dominik Bach	x	x
scr_get_rf	Dominik Bach	-	-
scr_get_rp	Dominik Bach	-	-
scr_get_rr	Dominik Bach	-	-
scr_get_scr	Dominik Bach	x	x
scr_get_spike	Dominik Bach	x	x
scr_get_timing	Dominik Bach	x	x
scr_get_txt	Dominik Bach	x	x

scr_get_vario	Dominik Bach	x	x
scr_get_wdq	Dominik Bach	-	-
scr_get_wdq_n	Tobias Moser	x	-
scr_glm_recon	Dominik Bach	-	-
scr_glm	Dominik Bach	x	x
scr_hb2hp	Dominik Bach	-	-
scr_hb2hr	Dominik Bach	-	-
scr_import	Dominik Bach	x	x
scr_init	Dominik Bach	-	-
scr_jobman	Gabriel Gräni	-	-
scr_job_create	Dominik Bach	-	-
scr_load_data	Dominik Bach	x	x
scr_load1	Dominik Bach	-	-
scr_merge	Dominik Bach	-	-
scr_pp	Dominik Bach	x	x
scr_prepdata	Dominik Bach	x	x
scr_pulse_convert	Dominik Bach	x	-
scr_quit	Dominik Bach	-	-
scr_ren	Dominik Bach	x	x
scr_resp2rp	Dominik Bach	-	-
scr_resp2rr	Dominik Bach	-	-
scr_rev_con	Dominik Bach	-	-
scr_rev_dcm	Dominik Bach	-	-
scr_rev_glm	Dominik Bach	-	-
scr_rev2	Dominik Bach	-	-
scr_review	Gabriel Graeni	-	-
scr_sf_auc	Dominik Bach	-	-
scr_sf_dcm	Dominik Bach	-	-
scr_sf_mp	Dominik Bach	-	-
scr_sf_scl	Dominik Bach	-	-
scr_sf_theta	Dominik Bach	-	-
scr_sf	Dominik Bach	-	-
scr_show_arms	Dominik Bach	-	-
scr_spike_convert	Dominik Bach	-	-
scr_split_sessions	Linus Rüttimann	x	x
scr_transfer_function	Dominik Bach	-	-
scr_trim	Dominik Bach	x	x
scr_ledalab	Dominik Bach	-	-
scr_peakscore	Dominik Bach	-	-
scr_sf_get_theta	Dominik Bach	-	-
scr_transfer_fit	Dominik Bach	-	-

Part V
Release notes

35 PsPM Version 3.0

Import

New data types were implemented

- Noldus Observer compatible
- Eyelink

Untested data types

CNT data import has not been not tested – please contact the developers with sample data files.

Filtering for SCR models

Previous versions of PsPM have used a bi-directional high pass filter of 0.0159 for all SCR analyses. We have recently shown a better predictive validity for GLM with a unidirectional filter of 0.05 Hz. This also implies that different filters are used for different models. These are now set as defaults, and the way the default settings are implemented has changed. It is now possible to alter the filter settings in the model definition, although we discourage this.

New SF method

A matching pursuit algorithm is now implemented to approximate the number of spontaneous fluctuations (SF) in skin conductance [Bach & Staib, Psychophysiology, in press].

General linear modelling (GLM)

Parametric modulators

Parametric modulators (pmods) are z-normalised before being entered into the design matrix. This is to account for possibly very large or very small numbers – a badly scaled design matrix can cause induced instability in the inversion. The parameter estimates of the pmods were not transformed back in previous versions, i. e. the parameter estimates of the pmods were independent of the scaling of the pmods. This is appropriate as long as they are the same for all datasets, or if analysis is done strictly on a within-subject level. Some researchers have reported designs in which inference was to be drawn on parameter estimates of pmods on a between-group level, and where the pmods systematically differed between these groups. To account for this possibility, the parameter estimates are now transformed back, such that they refer to the pmods in their original scaling/units.

Parametric confounds

Previous versions of PsPM contained an option to include a parametric modulator across all event types, to account for confounds across all conditions. For example, in an experiment with 5 conditions, one could have included 5 regressors, plus one reaction time confound across all events, without including an associated regressor that contains the event onsets for all these events. This option was removed.

Design matrix filtering

Previous versions of PsPM filtered the design matrix after orthogonalisation of basis sets. This can introduce unwanted dependencies between regressors. PsPM 3.0 filters the regressors first, then orthogonalises the basis sets.

File format

Some minor changes have been made to the data format. In particular, marker channels from previous versions can not be read with the current version - such data files have to be re-imported. Model files have changed drastically, and model files from previous versions can not be read with the current version of the software.

VB inversion

The VBA toolbox (`http://code.google.com/p/mbb-vb-toolbox/`) was updated in October 2014 [16]. This update incorporates bugfixes in this toolbox and slightly changed the model estimates in our test models. In terms of predictive validity, we noted that there was no consistent benefit of the old or new version of this code. A full description of this benchmarking will follow in the next release of the software.

Part VI
Acknowledgements

SCRalyze versions 1-2 were developed at the FIL, Wellcome Trust Centre for Neuroimaging, University College London, under the auspices of Karl J. Friston and Raymond J. Dolan. Guillaume Flandin contributed to GLM, and Jean Daunizeau to DCM. Eric Featherstone, Alfonso Reid, Oliver Josephs, and Chloe Hutton helped with the import functions. This development was funded by the Swiss National Science Foundation with a personal grant to Dominik R. Bach, and by the Wellcome Trust with a programme grant to Raymond J. Dolan.

PsPM 3.0 was developed at the University of Zurich, funded by the University of Zurich and by the Swiss National Science Foundation with a project grant.

PsPM uses a number of SPM functions. For import of spike data, PsPM makes use of the SON library written by Malcolm Lidierth at King's College London, and of functions provided by the SPM physiology toolbox, written by Chloe Hutton and Eric Featherstone at the FIL. Biopac AcqKnowledge import uses a routine created by Sebastien Authier and Vincent Finnerty at the University of Montreal. The DCM inversion uses a variational Bayesian inversion scheme written by Jean Daunizeau.

The Matlabbatch GUI was written by Gabriel Gräni at University of Zurich. Help texts and tutorial were developed by Christoph Korn and Matthias Staib. The data display was written by Philipp C Paulus, University of Dresden. The test environemnt was contributed by Linus Rüttimann, and extended by Tobias Moser, both University of Zurich.

Part VII
References

References

[1] Bach, D. R & Friston, K. J. (2013) Model-based analysis of skin conductance responses: Towards causal models in psychophysiology. *Psychophysiology* **50**, 15–22.

[2] Bach, D. R. (2014) A head-to-head comparison of scralyze and ledalab, two model-based methods for skin conductance analysis. *Biol Psychol.*

[3] Bach, D. R, Flandin, G, Friston, K. J, & Dolan, R. J. (2009) Time-series analysis for rapid event-related skin conductance responses. *J Neurosci Methods* **184**, 224–34.

[4] Bach, D. R, Flandin, G, Friston, K. J, & Dolan, R. J. (2010) Modelling event-related skin conductance responses. *Int J Psychophysiol* **75**, 349–56.

[5] Bini, G, Hagbarth, K. E, Hynninen, P, & Wallin, B. G. (1980) Thermoregulatory and rhythm-generating mechanisms governing the sudomotor and vasoconstrictor outflow in human cutaneous nerves. *J Physiol* **306**, 537–52.

[6] Sugenoya, J, Iwase, S, Mano, T, & Ogawa, T. (1990) Identification of sudomotor activity in cutaneous sympathetic nerves using sweat expulsion as the effector response. *Eur J Appl Physiol Occup Physiol* **61**, 302–8.

[7] Kunimoto, M, Kirnö, K, Elam, M, & Wallin, B. G. (1991) Neuroeffector characteristics of sweat glands in the human hand activated by regular neural stimuli. *J Physiol* **442**, 391–411.

[8] Kunimoto, M, Kirnö, K, Elam, M, Karlsson, T, & Wallin, B. G. (1992) Neuro-effector characteristics of sweat glands in the human hand activated by irregular stimuli. *Acta Physiol Scand* **146**, 261–9.

[9] Kunimoto, M, Kirnö, K, Elam, M, Karlsson, T, & Wallin, B. G. (1992) Nonlinearity of skin resistance response to intraneural electrical stimulation of sudomotor nerves. *Acta Physiol Scand* **146**, 385–92.

[10] Friston, K. J, Josephs, O, Rees, G, & Turner, R. (1998) Nonlinear event-related responses in fmri. *Magn Reson Med* **39**, 41–52.

[11] Friston, K. J, Mechelli, A, Turner, R, & Price, C. J. (2000) Nonlinear responses in fmri: the balloon model, volterra kernels, and other hemodynamics. *Neuroimage* **12**, 466–77.

[12] Bach, D. R, Friston, K. J, & Dolan, R. J. (2013) An improved algorithm for model-based analysis of evoked skin conductance responses. *Biol Psychol* **94**, 490–7.

[13] Bach, D. R, Friston, K. J, & Dolan, R. J. (2010) Analytic measures for quantification of arousal from spontaneous skin conductance fluctuations. *Int J Psychophysiol* **76**, 52–5.

[14] Bach, D. R, Daunizeau, J, Friston, K. J, & Dolan, R. J. (2010) Dynamic causal modelling of anticipatory skin conductance responses. *Biol Psychol* **85**, 163–70.

[15] Daunizeau, J, Friston, K. J, & Kiebel, S. J. (2009) Variational bayesian identification and prediction of stochastic nonlinear dynamic causal models. *Physica D* **238**, 2089–2118.

[16] Daunizeau, J, Adam, V, & Rigoux, L. (2014) Vba: a probabilistic treatment of nonlinear models for neurobiological and behavioural data. *PLoS Comput Biol* **10**, e1003441.

[17] Bach, D. R, Daunizeau, J, Kuelzow, N, Friston, K. J, & Dolan, R. J. (2011) Dynamic causal modeling of spontaneous fluctuations in skin conductance. *Psychophysiology* **48**, 252–257.

[18] Alexander, D. M, Trengove, C, Johnston, P, Cooper, T, August, J. P, & Gordon, E. (2005) Separating individual skin conductance responses in a short interstimulus-interval paradigm. *J Neurosci Methods* **146**, 116–23.

[19] Benedek, M & Kaernbach, C. (2010) Decomposition of skin conductance data by means of nonnegative deconvolution. *Psychophysiology* **47**, 647–58.

[20] Benedek, M & Kaernbach, C. (2010) A continuous measure of phasic electrodermal activity. *J Neurosci Methods* **190**, 80–91.

[21] Lim, C. L, Rennie, C, Barry, R. J, Bahramali, H, Lazzaro, I, Manor, B, & Gordon, E. (1997) Decomposing skin conductance into tonic and phasic components. *Int J Psychophysiol* **25**, 97–109.

[22] Pan, J & Tompkins, W. J. (1985) A real-time qrs detection algorithm. *IEEE Trans Biomed Eng* **32**, 230–6.