Pattern component modelling toolbox

Jörn Diedrichsen, Spencer A. Arbuckle, and Atusushi Yokoi

Contents

How to use this manual

This manual provides an introduction to how to use the Pattern component modelling (PCM) toolbox. The theory behind this approach is laid out in an accompanying paper (Diedrichsen et al., 2017) - but the main ideas are described here in the introduction. We then provide an overview over the toolbox functions, and explain the different steps of model specification, model estimation, visualisation, and model comparison following the real examples presented in the paper. The toolbox comes with a few example "recipes" that provide examples of usage on empricial data. Finally, the last section contains some of the mathematical and algorithmic details necessary to understand the implementation.

Introduction

The study of brain representations aims to illuminate the relationship between complex patterns of activity occurring in the brain and "things in the world" - be it objects, actions, or abstract concepts. By understanding internal syntax of brain representations, and especially how the structure of representations changes across different brain regions, we ultimately hope to gain insight into the way the brain processes information.

Central to the definition of representation is the concept of decoding (deCharms and Zador, 2000). A feature (i.e. a variable that describes some aspect of the "things in the world") that can be decoded from the ongoing neural activity in a region is said to be represented there. For example, a feature could be the direction of a movement, the orientation and location of a visual stimulus, or the semantic meaning of a word. Of course, if we allow the decoder to be arbitrarily complex, we would use the term representation in the most general sense. For example, using a computer vision algorithm, one may be able to identify objects based on activity in primary visual cortex. However, we may not conclude necessarily that object identity is represented in V1 - at least not explicitly. Therefore, it makes sense to restrict our definition of an explicit representation to features that can be linearly decoded by a single neuron from some population activity (DiCarlo and Cox, 2007; DiCarlo et al., 2012; Diedrichsen and Kriegeskorte, 2017; Kriegeskorte, 2011).

While decoding approaches are very popular in the study of multi-voxel activity patterns (Haxby et al., 2001; Norman et al., 2006; Pereira et al., 2009), they are not the most useful tool when we aim to make inferences about the nature of brain representations. The fact that we can decode feature X well from region A does not imply that the representation in A is well characterized by feature X - there may be many other features that better determine the activity patterns in this region.

Encoding models, on the other hand, characterize how well we can explain the activities in a specific region using a sets of features. The activity profile of each voxel (here shown as columns in the activity data matrix), is modeled as the linear combination of a set of features (Fig. 1a). We will use the term voxels interchangeably with the more general term measurement channel, which could, depending on the measurement modality, refer to a single neuron, an electrode, or sensor. Each voxel has its own set of parameters ($\mathbf{W}$) that determine the weight of each feature. This can visualized by plotting the activity profile of each voxel into the space spanned by the experimental conditions (Fig. 1b). Each dot refers to the activity profile of a channel (here a voxel), indicating how strongly the voxel is activated by each condition. Estimating the weights is equivalent to a projection of each of the activity profiles onto the feature vectors. The quality of the model can then be evaluated by determining how well unseen activity data can be predicted. When estimating the weights, encoding models often use some form of regularization, which essentially imposes a prior on the feature weights. This prior is an important component of the model. It determines a predicted distribution of the activity profiles (Diedrichsen and Kriegeskorte, 2017). An encoding model that matches the real distribution of activity profiles best will show the best prediction performance.

The interpretational problem for encoding models is that for each feature set that predicts the data well, there is an infinite number of other (rotated) features sets that describe the same distribution of activity profiles and, hence, predict the data equally well. The argument may be made that to understand brain representations, we should not think about specific features that are encoded, but rather about the distribution

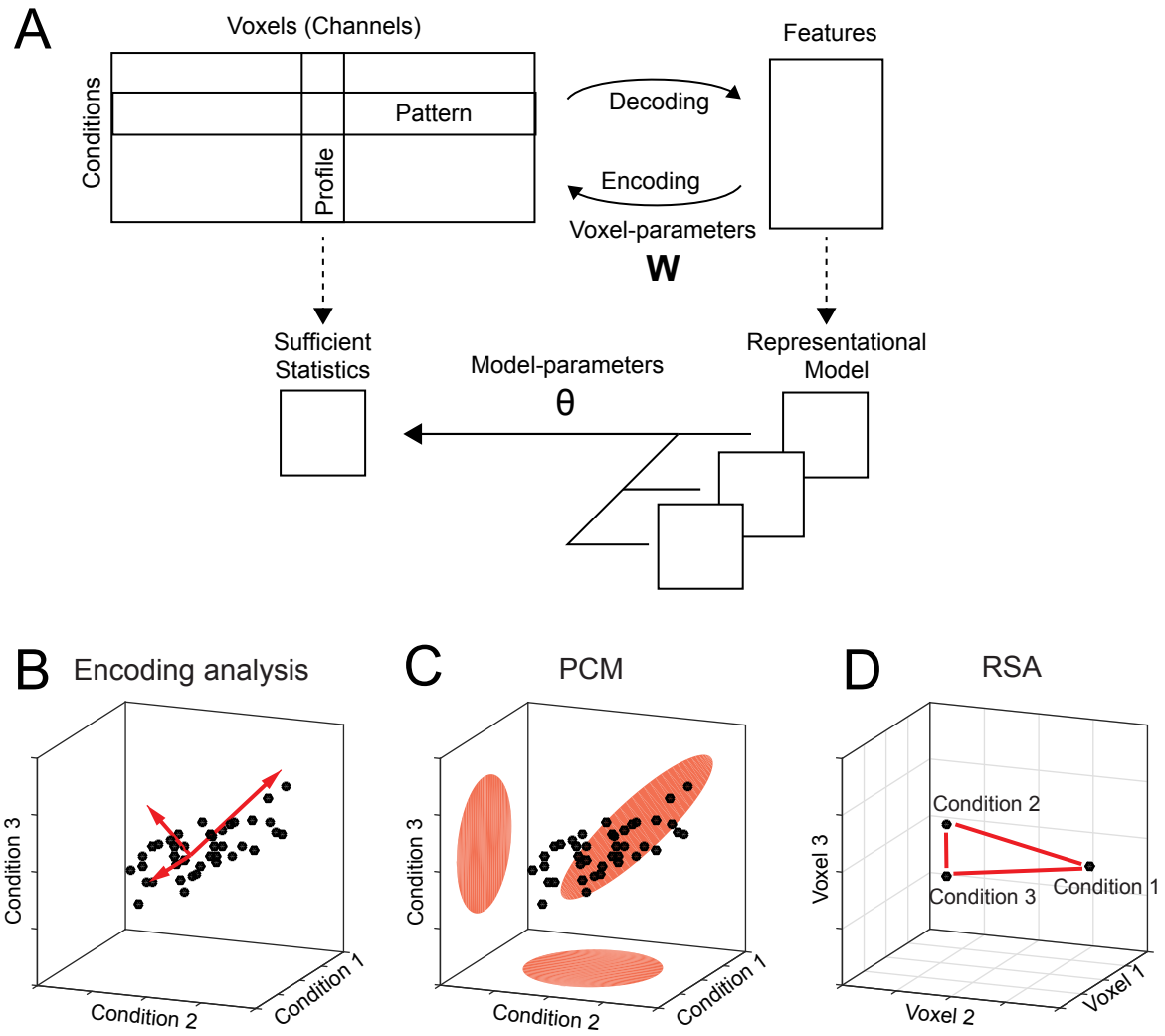

Figure 1: *Decoding, encoding and representational models.* (**A**) *The matrix of activity data consists of rows of activity patterns for each condition or of columns of activity profiles for each voxel (or more generally, measurement channel). The data can be used to decode specific features that describe the experimental conditions (decoding). Alternatively, a set of features can be used to predict the activity data (encoding). Representational models work at the level of a sufficient statistics (the second moment) of the activity profiles. Models are formulated in this space and possibly combined and changed using higher-order model parameters (θ).* (**B**) *The activity profiles of different voxels are plotted as points in the space of the experimental conditions. Features in encoding models are vectors that describe the overall distribution of the activity profiles.* (**C**) *The distribution can also be directly described using a multivariate normal distribution (PCM).* (**D**) *Representational similarity analysis (RSA) provides an alternative view by plotting the activity patterns in the space defined by different voxel activities. The distances between activity patterns serves here as the sufficient statistic, which is fully defined by the second moment matrix.*

of activity profiles. This can be justified by considering a read-out neuron that receives input from a population of neurons. From the standpoint of this neuron, it does not matter which neuron has which activity profile (as long as it can adjust input weights), and which features were chosen to describe these activity profiles - all that matters is what information can read out from the code. Thus, from this perspective it may be argued that the formulation of specific feature sets and the fitting of feature weights for each voxel are unnecessary distractions.

Therefore, our approach of pattern component modeling (PCM) abstracts from specific activity patterns. This is done by summarizing the data using a suitable summary statistic (Fig. 1a), that describes the shape of the activity profile distribution (Fig. 1c). This critical characteristic of the distribution is the covariance matrix of the activity profile distribution or - more generally - the second moment. The second moment determines how well we can linearly decode any feature from the data. If, for example, activity measured for two experimental conditions is highly correlated in all voxels, then the difference between these two conditions will be very difficult to decode. If however, the activities are uncorrelated, then decoding will be very easy. Thus, the second moment is a central statistical quantity that determines the representational content of the brain activity patterns of an area (Diedrichsen and Kriegeskorte, 2017).

Similarly, a representational model is formulated in PCM not by its specific feature set, but by its predicted second moment matrix. If two feature sets have the same second moment matrix, then the two models are equivalent. Thus, PCM makes hidden equivalences between encoding models explicit. To evaluate models, PCM simply compares the likelihood of the data under the distribution predicted by the model. To do so, we rely on an generative model of brain activity data, which fully specifies the distribution and relationship between the random variables. Specifically, true activity profiles are assumed to have a multivariate Gaussian distribution and the noise is also assumed to be Gaussian, with known covariance structure. Having a fully-specified generative model allows us to calculate the likelihood of data under the model, averaged over all possible values of the feature weights. This results in the so-called model evidence, which can be used to compare different models directly, even if they have different numbers of features. In summarizing the data using a sufficient statistic, PCM is closely linked to representation similarity analysis (RSA), which characterizes the second moment of the activity profiles in terms of the distances between activity patterns (Fig. 1d, also see Diedrichsen and Kriegeskorte (2017)).

By removing the requirement to fit and cross-validate individual voxel weights, PCM enables the user to concentrate on a different kind of free parameter, namely model parameters that determine the shape of the distribution of activity profiles. From the perspective of encoding models, these would be hyper-parameters that change the form of the feature or regression matrix. For example, we can fit the distribution of activity profiles using a weighted combination of 3 different feature sets (Fig. 1a). Such component models (see section **??**) are extremely useful if we hypothesize that a region cares about different groups of features (i.e. colour, size, orientation), but we do not know how strongly each feature is represented. In encoding models, this would be equivalent to providing a separate scaling factor to different parts of the feature matrix. Most encoding models, however, use a single model feature matrix, making them equivalent to a fixed PCM model.

In this manual we will show how to use the PCM toolbox to estimate and compare flexible representational models. We will present the fundamentals of the generative approach taken in PCM and outline different ways in which flexible representational models with free parameters can be specified. We will then discuss methods for model fitting and for model evaluation. We will also walk in detail through three illustrative examples from our work on finger representations in primary sensory and motor cortices, also providing recipe code for the examples presented in the paper.

Overview

The toolbox provides function for model fitting, model comparison, and some basic visualization. What the toolbox does *not* provide are functions to extract the required data from the first-level GLM or raw data, or to run search-light or ROI analyses. We have omitted these function as they strongly depend on the analysis package used for the basic imaging analysis. Some useful tools for the extraction of multivariate data from the standard first-level GLM, please see the RSA-toolbox and Surfing toolbox.

The functions provided in the toolbox can be categorized into different categories:

Basic likelihood and optimization

These are the functions that perform the core statistical functions of the toolbox.

Function	Comment
`pcm_likelihood`	Likelihood of a single data set under a model
`pcm_likelihoodIndivid`	pcm_likelihood with optional random block effect
`pcm_likelihoodGroup`	Likelihood of a group data set under a model
`pcm_NR`	Newton Raphson optimisation
`pcm_NR_diag`	Newton Raphson for diagonalized component models
`pcm_NR_free`	Newton Raphson for a free model
`pcm_EM`	Expectation-Maximization
`pcm_minimize`	Conjugate gradient descent

Model Evaluation

These functions are higher level functions that perform fitting and crossvalidation of either individual data set or group data sets.

Function	Comment
`pcm_fitModelIndivid`	Fits G and noise parameter to individual data
`pcm_fitModelIndividCrossval`	Within-subject crossvalidation of models
`pcm_fitModelGroup`	Fit common G to all subjects
`pcm_fitModelGroupCrossval`	Between-subject crossvalidation of models

Utility functions

Function	Comment
`pcm_checkderiv`	Checks derivate of a nonlinear model
`pcm_estGcrossval`	Cross-validated estimate of G
`pcm_indicatorMatrix`	Generates indicator matrices
`pcm_vararginoptions`	Dealing with variable options to functions
`pcm_optimalAlgorithm`	Makes a recommendation for the best optimisation algorithm
`pcm_getStartingval`	Estimates starting values for optimisation
`pcm_estimateU`	Calculates voxel-pattern estimates under the current model
`pcm_prepFreeModel`	Sets up free model (freechol)

Function	Comment
`pcm_makePD`	Forces covariance estimate to be semipositive
`pcm_generateData`	Generates data under a specific model for simulations

Visualization functions

Function	Comment
`pcm_classicalMDS`	Multidimensional scaling
`pcm_plotModelLikelihood`	Displays marginal likelihood for different models

Recipes

Function	Comment
`pcm_recipe_finger`	Example of a fixed and component models
`pcm_recipe_correlation`	Example of feature model
`pcm_recipe_nonlinear`	Example for non-linear model

Model specification

Generative model

Central to PCM is a generative model of the measured brain activity data $\mathbf{Y}$, a matrix of N x P activity measurements, referring to N time points (or trials) and P voxels. The data can refer to the minimally preprocessed raw activity data, or to already deconvolved activity estimates, such as those obtained as beta weights from a first-level time series model. $\mathbf{U}$ is the matrix of true activity patterns (a number of conditions x number of voxels matrix) and $\mathbf{Z}$ the design matrix. Also influencing the data are effects of no interest $\mathbf{B}$ and noise:

$$\begin{gathered}\mathbf{Y} = \mathbf{ZU} + \mathbf{XB} + \epsilon \\ \mathbf{u}_p \sim N(\mathbf{0}, \mathbf{G}) \\ \epsilon_p \sim N(\mathbf{0}, \mathbf{S}\sigma^2)\end{gathered}$$

There are a five assumptions in this generative model. First, the activity profiles ($\mathbf{u}_p$, columns of $\mathbf{U}$) are considered to be a random variable drawn from a normal distribution. Representational models therefore do not specify the exact activity profiles of specific voxels, but simply the characteristics of the distribution from which they originate. Said differently, PCM is not interested in which voxel has which activity profiles - it ignores their spatial arrangement. This makes sense considering that activity patterns can vary widely across different participants(Ejaz et al., 2015) and do not directly impact what can be decoded from a region. For this, only the distribution of these activity profiles in this region is considered.

The second assumption is that the mean of the activity profiles (across voxels) is the same across conditions, and that it is modeled using the effects of no interests . Therefore, we most often model in $\mathbf{X}$ the mean of each voxel across conditions. While one could also artificially remove the mean of each condition across voxels (Walther et al., 2016), this approach would remove differences that, from the persepctive of decoding and representation, are highly meaningful (Diedrichsen and Kriegeskorte, 2017).

The third assumption is that the activity profiles come from a multivariate Gaussian distribution. This is likely the most controversial assumption, but it is motivated by a few reasons: First, for fMRI data the multi-variate Gaussian is often a relatively appropriate description, especially if the mean of each voxel across conditions has been removed by the model. Secondly, the definition causes us to focus on the mean and covariance matrix, $\mathbf{G}$, as sufficient statistics, as these completely determine the Gaussian. Thus, even if the true distribution of the activity profiles is better described by a non-Gaussian distribution, the focus on the second moment is sensible as it characterizes the linear decodability of any feature of the stimuli.

Fourthly, the model assumes that different voxels are independent from each other. If we used raw data, this assumption would be clear violated, given the strong spatial correlation of noise processes in fMRI data. To reduce these dependencies we typically uses spatially pre-whitened data, which is divided by a estimate of the spatial covariance matrix (Diedrichsen and Kriegeskorte, 2017; Walther et al., 2016). One complication here is that spatial pre-whitening usually does not remove spatial dependencies completely, given the estimation error in the spatial covariance matrix (Diedrichsen et al., 2016).

Finally, we assume that the noise of each voxel is Gaussian with a temporal covariance that is known up to a constant term σ^2. Given the many additive influences of various noise sources on fMRI signals, Gaussianity of the noise is, by the central limit theorem, most likely a very reasonable assumption, which is commonly made in the fMRI literature. The original formulation of PCM used a model which assumed that the noise is also temporally independent and identically distributed (i.i.d.) across different trials, i.e. $\mathbf{S} = \mathbf{I}$. However, as pointed out recently (Cai et al., 2016), this assumption is often violated in non-random experimental designs with strong biasing consequences for estimates of the covariance matrix. If this is violated, we can either assume that we have a valid estimate of the true covariance structure of the noise (S), or we can model different parts of the noise structure (see section **??**).

When we fit a PCM model, we are not trying to estimate specific values of the the estimates of the true activity patterns $\mathbf{U}$. This is a difference to encoding approaches, in which we would estimate the values of $\mathbf{U}$ by estimating the feature weights $\mathbf{W}$. In PCM, we want to assess how likely the data is under any possible value of $\mathbf{U}$, as specified by the prior distribution. Thus we wish to calculate the marginal likelihood

$$p\left(\mathbf{Y}|\theta\right) = \int p\left(\mathbf{Y}|\mathbf{U},\theta\right) p\left(\mathbf{U}|\theta\right) d\mathbf{U}.$$

This is the likelihood that is maximized in PCM in respect to the model parameters θ. For more details, see mathematical and algorithmic details.

Model types

Fixed models

In fixed models, the second moment matrix $\mathbf{G}$ is exactly predicted by the model. The simplest and most common example is the Null model, which states that $\mathbf{G} = \mathbf{0}$. This is equivalent to assuming that there is no difference between the activity patterns measured under any of the conditions. The Null-model is useful if we want to test whether there are any differences between experimental conditions.

Fixed models also occur when the representational structure can be predicted from some independent data. An example for this is shown in section **??**, where we predict the structure of finger representations directly from the correlational structure of finger movements in every-day life (Ejaz et al., 2015). Importantly, fixed models only predict the the second moment matrix up to a proportional constant. The width of the distribution will vary with the overall signal-to-noise-level (assuming we use pre-whitened data). Thus, when evaluating fixed models we allow the predicted second moment matrix to be scaled by an arbitrary positive constant.

Example

An empirical example to for a fixed representational model comes from Ejaz et al (2015). Here the representational structure of 5 finger movements was compared to the representational structure predicted by the way the muscles are activated during finger movements (Muscle model), or by the covariance structure of natural movements of the 5 fingers. That is the predicted second moment matrix is derived from data completely independent of our imaging data.

Models are stored in structures, with the field `type` indicating the model type. To define a fixed model, we simple need to load the predicted second moment matrix and define a model structure as follows (see `pcm_recipe_finger`):

```
M.type = 'fixed'; % Type set to fixed
M.numGparams = 0; % Number of parameters
M.Gc = Model(1).G_cent; % This is the predicted second moment matrix from the behavioural data
M.name = 'muscle';% This is the name of the
```

When evaluating the likelihood of a data set under the prediction, the pcm toolbox still needs to estimate the scaling factor and the noise variance, so even in the case of fixed models, an iterative maximization of the likelihood is required (see below).

Component models

A more flexible model is to express the second moment matrix as a linear combination of different components. For example, the representational structure of activity patterns in the human object recognition system in inferior temporal cortex can be compared to the response of a convolutional neural network that is shown the same stimuli (Khaligh-Razavi and Kriegeskorte, 2014). Each layer of the network predicts a specific structure of the second moment matrix and therefore constitutes a fixed model. However, the real representational structure seems to be best described by a mixture of multiple layers. In this case, the overall predicted second moment matrix is a linear sum of the weighted components matrices:

$$\mathbf{G} = \sum_h \exp(\theta_h)\mathbf{G}_h.$$

The weights for each component need to be positive - allowing negative weights would not guarantee that the overall second moment matrix would be positive definite. Therefore we use the exponential of the weighing parameter here, such that we can use unconstrained optimization to estimate the parameters.

For fast optimization of the likelihood, we require the derivate of the second moment matrix in respect to each of the parameters. Thus derivative can then be used to calculate the derivative of the log-likelihood in respect to the parameters (see section 4.3. Derivative of the log-likelihood). In the case of linear component models this is easy to obtain.

$$\frac{\partial \mathbf{G}}{\partial \theta_h} = \exp(\theta_h)\mathbf{G}_h$$

Example

In the example `pcm_recipe_finger`, we have two fixed models, the Muscle and the natural statistics model. One question that arises in the paper is whether the real observed structure is better fit my a linear combination of the natural statistics and the muscle activity structure. So we can define a third model, which allows any arbitrary mixture between the two type.

```
M.type = 'component';
M.numGparams = 2;
M.Gc(:,:,1) = Model(1).G_cent;
```

```
M.Gc(:,:,2) = Model(2).G_cent;
M.name = 'muscle + usage';
```

Feature models

A representational model can be also formulated in terms of the features that are thought to be encoded in the voxels. Features are hypothetical tuning functions, i.e. models of what activation profiles of single neurons could look like. Examples of features would be Gabor elements for lower-level vision models (Kay et al., 2008), elements with cosine tuning functions for different movement directions for models of motor areas (Eisenberg et al., 2010), and semantic features for association areas (Huth et al., 2016). The actual activity profiles of each voxel are a weighted combination of the feature matrix $\mathbf{u}_p = \mathbf{M}\mathbf{w}_p$. The predicted second moment matrix of the activity profiles is then $\mathbf{G} = \mathbf{M}\mathbf{M}^T$, assuming that all features are equally strongly and independently encoded, i.e. $E\left(\mathbf{w}_p\mathbf{w}_p^T\right) = \mathbf{I}$. A feature model can now be flexibly parametrized by expressing the feature matrix as a weighted sum of linear components.

$$\mathbf{M} = \sum_h \theta_h \mathbf{M}_h$$

Each parameter θ_h determines how strong the corresponding set of features is represented across the population of voxels. Note that this parameter is different from the actual feature weights $\mathbf{W}$. Under this model, the second moment matrix becomes

$$\mathbf{G} = \mathbf{U}\mathbf{U}^T/P = \frac{1}{P}\sum_h \theta_h^2 \mathbf{M}_h\mathbf{M}_h^T + \sum_i \sum_j \theta_i\theta_j \mathbf{M}_i\mathbf{M}_j^T.$$

From the last expression we can see that, if features that belong to different components are independent of each other, i.e. $\mathbf{M}_i\mathbf{M}_j = \mathbf{0}$, then a feature model is equivalent to a component model with $\mathbf{G}_h = \mathbf{M}_h\mathbf{M}_h^T$. The only technical difference is that we use the square of the parameter θ_h, rather than its exponential, to enforce non-negativity. Thus, component models assume that the different features underlying each component are encoded independently in the population of voxels - i.e. knowing something about the tuning to feature of component A does not tell you anything about the tuning to a feature of component B. If this cannot be assumed, then the representational model is better formulated as a feature model.

By the product rule for matrix derivatives, we have

$$\frac{\partial \mathbf{G}}{\partial \theta_h} = \mathbf{M}_h\mathbf{M}(\theta)^T + \mathbf{M}(\theta)\,\mathbf{M}_h^T$$

Example

In the example `pcm_recipe_correlation`, we want to model the correlation between the patterns for the left hand and the corresponding fingers for the right hand.

There two features to simulate the common pattern for the left and right hand movements, respectively (θ_d, θ_e). For the fingers of the contra-lateral hand we have one feature for each finger, with the feature component weighted by θ_a. The same features also influence the patterns for the ipsilateral hand with weight θ_b. This common component models the correspondence between contra and ipsilateral fingers. Finally, the component weighted by θ_c encodes unique encoding for the ipsilateral fingers.

```
M.type       = 'feature';
M.numGparams = 5;
M.Ac(:,1:5 ,1)  = [eye(5);zeros(5)];      % Contralateral finger patterns   (a)
M.Ac(:,1:5 ,2)  = [zeros(5);eye(5)];      % Mirrored Contralateralpatterns  (b)
M.Ac(:,6:10,3)  = [zeros(5);eye(5)];      % Unique Ipsilateral pattterns    (c)
M.Ac(:,11  ,4)  = [ones(5,1);zeros(5,1)]; % Hand-specific component contra  (d)
```

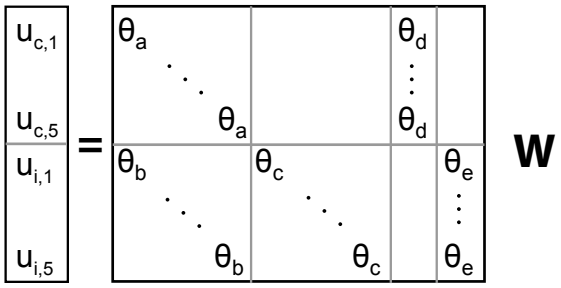

Figure 2: *Feature model to model correlation.*

```
M.Ac(:,12  ,5)  = [zeros(5,1);ones(5,1)]; % Hand-specific component ipsi    (e)
M.name       = 'correlation';
M.theta0=[1 1 0.5 0.1 0.1 ]';       % Starting values
```

Nonlinear models

The most flexible way of defining a representational model is to express the second moment matrix as a non-linear (matrix valued) function of the parameters, $\mathbf{G} = F(\theta)$. While often a representational model can be expressed as a component or feature model, sometimes this is not possible. One example is a representational model in which the width of the tuning curve (or the width of the population receptive field) is a free parameter (Dumoulin and Wandell, 2008). Such parameters would influence the features, and hence also the second-moment matrix in a non-linear way. Computationally, such non-linear models are not much more difficult to estimate than component or feature models, assuming that one can analytically derive the matrix derivatives $\partial\mathbf{G}/\partial\theta_h$.

For this, the user needs to define a function that takes the parameters as an input and returns $\mathbf{G}$ the partial derivatives of $\mathbf{G}$ in respect to each of these parameters. The derivates are returned as a (KxKxH) tensor, where H is the number of parameters.

```
[G,dGdtheta]=fcn(theta,data,…)
```

Note that this function is repeatedly called by the optimization routine and needs to execute fast. That is, any computation that does not depend on the current value of θ should be performed outside the function and then passed to it.

Example

In the example `pcm_recipe_nonlinear`, we define how the representational structure of single finger presses of the right hand ($\mathbf{G}$) scales as the number of presses increases. To achieve this, we can simply allow for a scaling component (θ_f) for each pressing speed (f). In the recipe, we have four pressing speeds. Therefore, we use $\mathbf{G}$ from one pressing speed to model the $\mathbf{G}$s of the remaining three pressing speeds. For one pressing speed, $\mathbf{G}$ is a 5x5 matrix, where each dimension corresponds to one finger. To speed up the optimization routine, we set $\mathbf{G}(1,1)$ to one. The parameters in $\mathbf{G}$ are then free to vary with respect to $\mathbf{G}(1,1)$.

```
M.type       = 'nonlinear';
M.name       = 'Scaling';
M.modelpred  = @ra_modelpred_scale;
M.numGparams = 17;            % 14 free theta params in G because G(1,1) is set to 1, and 3 free scaling param
M.theta0    = [Fx0; scale_vals];  % Fx0 are the 14 starting values from G, scale_vals are 3 starting scaling
```

Free models

The most flexible representational model is the free model, in which the predicted second moment matrix is unconstrained. Thus, when we estimate this model, we would simply derive the maximum-likelihood estimate of the second-moment matrix. This can be useful for a number of reasons. First, we may want an estimate of the second moment matrix to derive the corrected correlation between different patterns, which is less influenced by noise than the simple correlation estimate (Cai et al., 2016; Diedrichsen et al., 2011). Furthermore, we may want to estimate the likelihood of the data under a free model to obtain a noise ceiling - i.e. an estimate of how well the best model should fit the data (see section **??**).

In estimating an unconstrained $\mathbf{G}$, it is important to ensure that the estimate will still be a positive definite matrix. For this purpose, we express the second moment as the square of an upper-triangular matrix, $\mathbf{G} = \mathbf{A}\mathbf{A}^T$ (Cai et al., 2016; Diedrichsen et al., 2011). The parameters are then simply all the upper-triangular entries of $\mathbf{A}$.

Example

To set up a free model, the model type needs to be set to `freechol`, and you need to provide the number of conditions. The function `pcm_prepFreeModel` then quickly calculates the row and column indices for the different free parameters, which is a useful pre-computation for subsequent model fitting.

```
M.type       = 'freechol';
M.numCond    = 5;
M.name       = 'noiseceiling';
M            = pcm_prepFreeModel(M);
```

For a quick and approximate noise ceiling, you can also set the model type to `freedirect`. In the case, the fitting algorithms simply use the crossvalidated second moment to determine the parameters - basically the starting values of the complete model. This may lead to a slightly lower noise ceiling, as full optimization is avoided in the interest of speed.

Noise assumptions

The noise is assumed to come from a multivariate normal distribution with covariance matrix $\mathbf{S}\sigma^2$. What is a reasonable noise structure to assume? First, the data can usually be assumed to be independent across imaging runs. If the data are regression estimates from a first-level model, and if the design of the experiment is balanced, then it is usually also permissible to make the assumption that the noise is independent within each imaging run $\mathbf{S} = \mathbf{I}$, (Diedrichsen et al., 2011). Usually, however, the regression coefficients from a single imaging run show positive correlations with each other. This is due to the fact that the regression weights measure the activation during a condition as compared to a resting baseline, and the resting baseline is common to all conditions within the run (Diedrichsen et al., 2011). To account for this, it is recommended to remove the mean pattern across by modeling the run mean as a fixed effects. This can be done by setting the option `runEffect = 'fixed'` in the model fittting routines mentioned below. This effectively accounts for any uniform correlation between activity estimates withn a run.

Usually, assuming equal correlations of the activation estimates within a run is only a rough approximation to the real co-varince structure. A better estimate can be obtained by using an estimate derived from the design matrix and the estimated temporal autocorrelation of the raw signal. As pointed out recently (Cai et al., 2016), the particular design can have substantial influence on the estimation of the second moment matrix. This is especially evident in cases where the design is such that the trial sequence is not random, but has an invariant structure (where trials of one condition are often to follow trials of another specific condition). The accuracy of our approximation hinges critically on the quality of our estimate of the temporal auto-covariance structure of the true noise. Note that it has been recently demonstrated that especially for high sampling rates, a simple autoregressive model of the noise is insufficient (Eklund et al., 2012). In all optimisation

routine, a specific noise covariance structure can be specified by setting the option `'S'` in the model fitting routines to the NxN covariance estimate.

The last option is to estimate the covariance structure of the noise from the data itself. This can be achieved by introducing random effects into the generative model equation in section **??**, which account for the covariance structure across the data. One example used here is to assume that the data are independent within each imaging run, but share an unknown covariance within each run, which is then estimated as a part of the covariance matrix (Diedrichsen et al., 2011). While this approach is similar to just removing the run mean from the data as a fixed effect, it is a good strategy if we actually want to model the difference of each activation pattern against the resting baseline. When treating the mean activation pattern in each run as a random effect, the algorithm finds a compromise between how much of the shared pattern in each run to ascribe to the random run-to-run fluctuations, and how much to ascribe to a stable mean activation. This can be done by setting the option `runEffect = 'random'` in the model fitting routines. This approach is taken for example in `pcm_recipe_nonlinear` as here the resting baseline is of interest and should not be artificially removed.

Model fitting and crossvalidation

Details of the different optimization routines that maximize the likelihood can be found in section on Mathematical and Algorithmic details. Currently, the toolbox either uses `minimize` (conjugate gradient descent) or `pcm_NR` (Newton-Raphson). Newton-Raphson can be substantially faster in cases in which there are relatively few free parameters. The optimisation routine can be set for each model seperately by setting the field `M.fitAlgorithm`.

The following routines are wrapper functions around the actual optimisation routines that fit models to individual or group data. Noise and (possibly) scale paramaters are added to the fit for each subject. To compare models of different complexity, 2 types of crossvalidation are implemented.

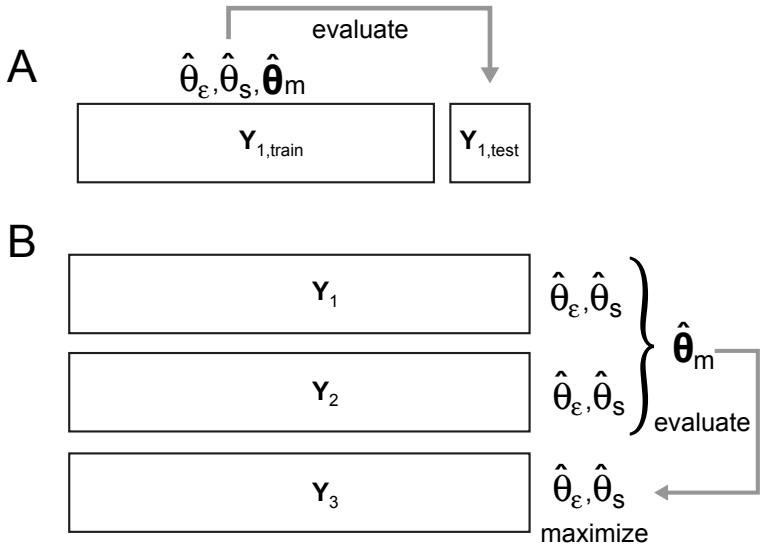

Figure 3: *Model crossvalidation schemes.* (**A**) *Within-subject crossvalidation where the model is fit on N-1 partitions and then evaluated on the left-out partition N.* (**B**) *Group crossvalidation were the model is fit to N-1 subjects and then evaluated on a left-out subject. In all cases, an individual scaling and noise parameters is fit to each subject to allow for different signal-to-noise levels.*

Fitting to individual data sets

Models can be fitted to each data set individually, using the function `pcm_fitModelIndivid`. Individual fitting makes sense for models with a single component (fixed models), which can be evaluated without crossvalidation. It also makes sense when the main interest are the parameters of the fit from each individual.

```
function [T,theta_hat,G_pred]=pcm_fitModelIndivid(Y,M,partitionVec,conditionVec,varargin);
```

The input arguments are:

```
% INPUT:
%        Y: {#Subjects}.[#Conditions x #Voxels]
%            Observed/estimated beta regressors from each subject.
%            Preferably multivariate noise-normalized beta regressors.
%
%        M: {#Models} Cell array with structure that defines model(s).
%
%   partitionVec: {#Subjects} Cell array with partition assignment vector
%                   for each subject. Rows of partitionVec{subj} define
%                   partition assignment of rows of Y{subj}.
%                   Commonly these are the scanning run #s for beta
%                   regressors.
%                   If a single vector is provided, it is assumed to me the
%                   same for all subjects
%
%   conditionVec: {#Subjects} Cell array with condition assignment vector
%                   for each subject. Rows of conditionVec{subj} define
%                   condition assignment of rows of Y{subj}.
%                   If a single vector is provided, it is assumed to me the
%                   same for all subjects
%                   If the (elements of) conditionVec are matrices, it is
%                   assumed to be the design matrix Z, allowing the
%                   specification individualized models.
```

Optional inputs are:

```
% OPTION:
%   'runEffect': How to deal with effects that may be specific to different
%                imaging runs:
%                  'random': Models variance of the run effect for each subject
%                            as a seperate random effects parameter.
%                  'fixed': Consider run effect a fixed effect, will be removed
%                            implicitly using ReML.
%
%   'isCheckDeriv: Check the derivative accuracy of theta params. Done using
%                   'checkderiv'. This function compares input to finite
%                   differences approximations. See function documentation.
%
%   'MaxIteration': Number of max minimization iterations. Default is 1000.
%
%   'S',S         : (Cell array of) NxN noise covariance matrices -
%                   otherwise independence is assumed
```

And the outputs are defined as:

```
%   T:      Structure with following subfields:
%       SN:                 Subject number
```

```
%       likelihood:         log likelihood
%       noise:              Noise parameter
%       run:                Run parameter (if run = 'random')
%       iterations:         Number of interations for model fit
%       time:               Elapsed time in sec
%
%   theta{m}     Cell array of estimated model parameters, each a
%                 #params x #numSubj matrix
%   G_pred{m}     Cell array of estimated G-matrices under the model
```

The output can be used to compare the likelihoods between different models. Alternatively you can inspect the individual fits by looking at the parameters (theta) or the fitted second moment matrix (G_pred).

Example

In `pcm_recipe_correlation`, we fit the feature model described above and then use the fitted parameters to determine the predicted correlation.

```
[D,theta,G_hat] = pcm_fitModelIndivid(Data,M,partVec,condVec,'runEffect','fixed');

% Get the correlations from the parameters for Model1
var1        = theta{1}(1,:).^2;
var2        = theta{1}(2,:).^2+theta{1}(3,:).^2;
cov12       = theta{1}(1,:).*theta{1}(2,:);
r_model1    = (cov12./sqrt(var1.*var2))';
```

Fitting to individual data sets with cross-validation across partitions

Crossvalidation within subject (Fig. 3a) is the standard for encoding models and can also be applied to PCM-models.

Example

To be provided

Fitting to group data sets

The function `pcm_fitModelGroup` fits a model to a group of subjects. All parameters that change the **G** matrix, that is all `M.numGparams`, are shared across all subjects. To account for the individual signal-to-noise level, by default a separate signal strength (s_i) and noise parameter ($\sigma^2_{\epsilon,i}$) are fitted for each subject. That is, for each individual subject, the predicted covariance matrix of the data is:

$$\mathbf{V_i} = s_i \mathbf{Z}\mathbf{G}\mathbf{Z}^\mathbf{T} + \mathbf{S}\sigma^\mathbf{2}_{\epsilon,\mathbf{i}}.$$

To prevent scaling or noise variance parameters to become negative, we actually optimise the log of these parameter, such that

$$s_i = exp(\theta_{s,i})$$
$$\sigma^2_{\epsilon,i} = exp(\theta_{\epsilon,i}).$$

The output `theta` for each model contains not only the `M.numGparams` model parameters, but also the noise parameters for all the subjects, then (optional) the scale parameters for all the subjects, and (if the runEffect

is set to random) the run-effect parameter for all the subjects. The resultant scaling parameter for each subject is additionally stored under `T.scale` in the output structure. The fitting of an additional scale parameter can be switched off by providing the optional input argument `pcm_fitModelGroup(...,'fitScale',0)`. Often, it speeds up the computation to perform a group fit first, so it can serve as a starting value for the crossvalidated group fit (see below). Otherwise the input and output parameters are identical to `pcm_fitModelIndivid`.

Note that the introduction of the scale parameter introduces a certain amount of parameter redundancy. For example, if a model has only one single component and parameter, then the overall scaling is simply `s*theta`. One can remove the redundancy by forcing one model parameter to be 1 - in practice this, however, is not necessary.

Fitting to group data sets with cross-validation across participants

PCM allows also between-subject crossvalidation (Fig. 3b). The model parameters that determine the representational structure are fitted to all the subjects together, using separate noise and scale parameters for each subject. Then the model is evaluated on the left-out subjects, after maximizing both scale and noise parameters. Function pcm_fitModelIndividCrossval.m implements these steps.

The function `pcm_fitModelGroupCrossval` implements these steps. As an additional input parameter, one can pass the parameters from the group fit as `(...,'groupFit',theta,...)`. Taking these as a starting point can speed up convergence.

Example

The function `pcm_recipe_finger` provides a full example how to use group crossvalidation to compare different models. Three models are being tested: A muscle model, a usage model (both a fixed models) and a combination model, in which both muscle and usage can be combined in any combination. Because the combination model has one more parameter than each single model, crossvalidation becomes necessary. Note that for the simple models, the simple group fit and the cross-validated group fit are identical, as in both cases only a scale and noise parameter are optimized for each subject.

```
% Model 1: Null model for baseline: here we use a model which has all finger
% Patterns be independent - i.e. all finger pairs are equally far away from
% each other
M{1}.type       = 'component';
M{1}.numGparams = 1;
M{1}.Gc         = eye(5);
M{1}.name       = 'null';

% Model 2: Muscle model: derived from covariance structure of muscle
% activity during single finger movements
M{2}.type       = 'component';
M{2}.numGparams = 1;
M{2}.Gc         = Model(1).G_cent;
M{2}.name       = 'muscle';

% Model 3: Natural statistics model: derived from covariance structure of
% natual movements
M{3}.type       = 'component';
M{3}.numGparams = 1;
M{3}.Gc         = Model(2).G_cent;
M{3}.name       = 'usage';
```

```
% Model 4: Additive mixture between muscle and natural stats model
M{4}.type       = 'component';
M{4}.numGparams = 2;
M{4}.Gc(:,:,1)  = Model(1).G_cent;
M{4}.Gc(:,:,2)  = Model(2).G_cent;
M{4}.name       = 'muscle + usage';

% Model 5: Free model as Noise ceiling
M{5}.type       = 'freechol';
M{5}.numCond    = 5;
M{5}.name       = 'noiseceiling';
M{5}           = pcm_prepFreeModel(M{5});

% Fit the models on the group level
[Tgroup,theta] = pcm_fitModelGroup(Y,M,partVec,condVec,'runEffect','fixed','fitScale',1);

% Fit the models through cross-subject crossvalidation
[Tcross,thetaCr] = pcm_fitModelGroupCrossval(Y,M,partVec,condVec,'runEffect','fixed','groupFit',theta,'f

% Provide a plot of the crossvalidated likelihoods
subplot(2,3,[3 6]);
T = pcm_plotModelLikelihood(Tcross,M,'upperceil',Tgroup.likelihood(:,5));
```

Data visualisation

Second Moment matrices

One important way to visualize both the data and the model prediction is to show the second moment matrix as a colormap, for example using the command `imagesc`. The predicted second moment matrix for each mode is being returned by the fitting routines (see above) and can therefore directly visualized. A useful estimate for the empirical second moment matrix is the cross-validated estimate obtained using `pcm_estGCrossval`. Note that if you removed the block-effect using the runEffect option 'fixed' then you need to also remove it from the data to have a fair comparison. A simple way to do so is to center the estimate - i.e., to make the mean of the rows and columns of the second moment matrix zero. This is equivalent to subtracting out the mean pattern.

```
G_hat=pcm_estGCrossval(Y,partVec,condVec);
H = eye(5)-ones(5)/5;  % Centering matrix
imagesc(H*Gm*H');
```

Note also that you can transform a second moment matrix into a representational dissimilarity matrix (RDM) using the following equivalence (see Diedrichsen & Kriegeskorte, 2016):

```
C   = pcm_indicatorMatrix('allpairs',[1:numCond]');
RDM = squareform(diag(C*G*C'));
```

The RDM can also be plotted as the second-moment matrix. The only difference is that the RDM does not contain information about the baseline.

Multidimensional scaling

Another important way of visualizing the second moment matrix is Multi-dimensional scaling (MDS), an important technique in representational similarity analysis. When we look at the second moment of a population code, the natural way of performing this is classical multidimensional scaling. This technique

plots the different conditions in a space defined by the first few eigenvectors of the second moment matrix - where each eigenvector is weighted by the $sqrt(\lambda)$.

Importantly, MDS provides only one of the many possible 2- or 3-dimensional views of the high-dimensional representational structure. That means that one should never make inferences from this reduced view. It is recommended to look at as many different views of the representational structure as possible to obtain a unbiased impression. For high dimensional space, you surely will find *one* view that shows exactly what you want to show. There are a number of different statistical visualisation techniques that can be useful here, including the 'Grand tour' which provides a movie that randomly moves through different high-dimensional rotations.

To enable the user to take a slightly more guided approach to visualizing, the function `pcm_classicalMDS` allows the specification of a contrast. The resultant projection is the subspace, in which the contrast is maximally represented. For example we can look at the finger patterns by looking at the difference between all the fingers:

```
C = pcm_indicatorMatrix('allpairs',[1:5]');
COORD=pcm_classicalMDS(G,'contrast',C);
plot(COORD(:,1),COORD(:,2),'o');
```

The following command then provides a different projection, optimized for showing the difference between thumb and index and thumb and middle finger. The other fingers are then projected into this space

```
C=[1 -1 0 0 0;1 0 -1 0 0]';
[COORD,l]=pcm_classicalMDS(Gm,'contrast',C);
```

Using different contrast makes especially sense when more conditions are studied, which can be described by different features or factors. These factors can then be used to provide a guide by which to rotate the representational structure. Note that when no contrast is provided, the functions attempts to represent the overall second moment matrix as good as possible. If the second moment is not centered, then the function chooses the projection that best captures the activity of each condition relative to baseline (rest). This often results in a visualisation that is dominated by one mean vector (the average activity pattern).

Plotting model evidence

Another approach to visualize model results is to plot the model evidence (i.e. marginal likelihoods). The marginal likelihoods are returned from the modeling routines in arbitrary units, and are thus better understood after normalizing to a null model at the very least. The lower normalization bound can be a null model, and upper bound is often a noise ceiling. This technique simply plots scaled likelihoods for each model fit.

```
T = pcm_plotModelLikelihood(Tcross,M,'upperceil',Tgroup.likelihood(:,5));
```

Visualising the voxel-feature weights

Finally, like in an encoding model, we can also estimate and visualise the estimated voxel-feature weights or activity patterns, that are most likely under the current model. The PCM toolbox provides two ways of formulating an encoding model: First, the model features can be fixedly encoded in the design matrix $\mathbf{Z}$ and the data is modeled as:

$$\mathbf{Y} = \mathbf{ZU} + \boldsymbol{\epsilon}$$

.

In this case, $\mathbf{U}$ can be interpreted as the voxel-feature weights, and PCM estimates the variances and covariance of these feature maps, encoded in $\mathbf{G}(\boldsymbol{\theta})$. You can get the BLUP estimates of $\mathbf{U}$ under model M by calling

```
U=pcm_estimateU(M,theta,Data,Z,X);
```

Secondly, in feature models, $\mathbf{Z}$ simply provides the basic asssignment of trials to conditions. The features themselves are encoded in the feature matrix $\mathbf{A}$, which itself can flexibly be changed based on the parameters.

$$\mathbf{Y} = \mathbf{Z}\mathbf{A}(\boldsymbol{\theta})\mathbf{W} + \boldsymbol{\epsilon}$$

For this case, you can obtain the best estimate of the voxel-feature weights and the currently best feature matrix by calling:

```
[W,A]=pcm_estimateW(M,theta,Data,Z,X);
```

Feature weights can then be visualized using standard techniques either in the volume or on the surface. Routines to do this are not included in the PCM toolbox.

Mathematical and Algorithmic details

Likelihood

In this section, we derive the likelihood in the case that there are no fixed effects. In this case the distribution of the data would be

$$\mathbf{y} \sim N(0, \mathbf{V})$$
$$\mathbf{V} = \mathbf{Z}\mathbf{G}\mathbf{Z}^{\mathbf{T}} + \mathbf{S}\sigma_{\epsilon}^{\mathbf{2}}$$

To calculate the likelihood, let us consider at the level of the single voxel, namely, $\mathbf{Y} = [\mathbf{y_1}, \mathbf{y_2}, ..., \mathbf{y_p}]$. Then the likelihood over all voxels, assuming that the voxels are independent (e.g. effectively pre-whitened) is

$$p(\mathbf{Y}|\mathbf{V}) = \prod_{i=1}^{P} (2\pi)^{-\frac{N}{2}} |\mathbf{V}|^{-\frac{1}{2}} exp\left(-\frac{1}{2}\mathbf{y}_i^T \mathbf{V}^{-1}\mathbf{y}_i\right)$$

When we take the logarithm of this expression, the product over the individual Gaussian probabilities becomes a sum and the exponential disappears:

$$L = \ln\left(p(\mathbf{Y}|\mathbf{V})\right) = \sum_{i=1}^{P} \ln p(\mathbf{y_i})$$

$$= -\frac{NP}{2}\ln(2\pi) - \frac{P}{2}\ln(|\mathbf{V}|) - \frac{1}{2}\sum_{i=1}^{P} \mathbf{y}_i^T \mathbf{V}^{-1}\mathbf{y}_i$$

$$= -\frac{NP}{2}\ln(2\pi) - \frac{P}{2}\ln(|\mathbf{V}|) - \frac{1}{2} trace\left(\mathbf{Y}^T\mathbf{V}^{-1}\mathbf{Y}\right)$$

Using the trace trick, which allows trace $(\mathbf{ABC}) =$ trace $(\mathbf{BCA})$, we can obtain a form of the likelihood that does only depend on the second moment of the data, $\mathbf{YY^T}$,as a sufficient statistics:

$$L = -\frac{NP}{2}\ln(2\pi) - \frac{P}{2}\ln(|\mathbf{V}|) - \frac{1}{2} trace\left(\mathbf{Y}\mathbf{Y}^T\mathbf{V}^{-1}\right)$$

Restricted likelihood

In the presence of fixed effects (usually effects of no interest), we have the problem that the estimation of these fixed effects depends iterativly on the current estimate of $\mathbf{V}$ and hence on the estimates of the second moment matrix and the noise covariance.

$$\hat{\mathbf{B}} = \left(\mathbf{X}^T\mathbf{V}^{-1}\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{V}^{-1}\mathbf{Y}$$

Under the assumption of fixed effects, the distribution of the data is

$$\mathbf{y_i} \sim N\left(\mathbf{X}\mathbf{b_i}, \mathbf{V}\right)$$

To compute the likelihood we need to remove these fixed effects from the data, using the residual forming matrix

$$\mathbf{R} = \mathbf{X}\left(\mathbf{X^T}\mathbf{V^{-1}}\mathbf{X}\right)^{-\mathbf{1}}\mathbf{X^T}\mathbf{V^{-1}}\mathbf{r_i} = \mathbf{R}\mathbf{y_i}$$

For the optimization of the random effects we therefore also need to take into account the uncertainty in the random effects estimates. Together this leads to a modified likelihood - the restricted likelihood that we which to optimize.

$$L_{ReML} = -\frac{NP}{2}\ln\left(2\pi\right) - \frac{P}{2}\ln\left(|\mathbf{V}|\right) - \frac{1}{2}trace\left(\mathbf{Y}\mathbf{Y}^T\mathbf{R^T}\mathbf{V^{-1}}\mathbf{R}\right) - \frac{P}{2}\ln|\mathbf{X^T}\mathbf{V^{-1}}\mathbf{X}|$$

Note that the third term can be simplified by noting that

$$\mathbf{R^T}\mathbf{V^{-1}}\mathbf{R} = \mathbf{V^{-1}} - \mathbf{V^{-1}}\mathbf{X}(\mathbf{X}\mathbf{V^{-1}}\mathbf{X})^{-\mathbf{1}}\mathbf{X^T}\mathbf{V^{-1}} = \mathbf{V^{-1}}\mathbf{R} = \mathbf{V_R^{-1}}$$

First derivatives of the log-likelihood

Next, we find the derivatives of L with respect to each hyper parameter θ_i, which influence G. Also we need to estimate the hyper-parameters that describe the noise, at least the noise parameter σ_ϵ^2. To take these derivatives we need to use two general rules of taking derivatives of matrices (or determinants) of matrices:

$$\frac{\partial \ln\left(\mathbf{V}\right)}{\partial\theta_i} = trace\left(\mathbf{V}^{-1}\frac{\partial\mathbf{V}}{\partial\theta_i}\right)$$

$$\frac{\partial\mathbf{V}^{-1}}{\partial\theta_i} = \mathbf{V}^{-1}\left(\frac{\partial\mathbf{V}}{\partial\theta_i}\right)\mathbf{V}^{-1}$$

Therefore the derivative of the log-likelihood in Eq. **??**. in respect to each parameter is given by:

$$\frac{\partial L_{ML}}{\partial\theta_i} = -\frac{P}{2}trace\left(\mathbf{V}^{-1}\frac{\partial\mathbf{V}}{\partial\theta_i}\right) + \frac{1}{2}trace\left(\mathbf{V}^{-1}\frac{\partial\mathbf{V}}{\partial\theta_i}\mathbf{V}^{-1}\mathbf{Y}\mathbf{Y}^T\right)$$

First derivatives of the restricted log-likelihood

First, let's tackle the last term of the restricted likelihood function:

$$l = -\frac{P}{2}\ln|\mathbf{X}^T\mathbf{V}^{-1}\mathbf{X}|$$

$$\frac{\partial l}{\partial\theta_i} = -\frac{P}{2}trace\left(\left(\mathbf{X}^T\mathbf{V}^{-1}\mathbf{X}\right)^{-1}\mathbf{X}^T\frac{\partial\mathbf{V}^{-1}}{\partial\theta_i}\mathbf{X}\right)$$

$$= \frac{P}{2} trace\left(\left(\mathbf{X}^T\mathbf{V}^{-1}\mathbf{X}\right)^{-1} \mathbf{X}^T\mathbf{V}^{-1}\frac{\partial \mathbf{V}}{\partial \theta_i}\mathbf{V}^{-1}\mathbf{X}\right)$$

$$= \frac{P}{2} trace\left(\mathbf{V}^{-1}\mathbf{X}\left(\mathbf{X}^T\mathbf{V}^{-1}\mathbf{X}\right)^{-1} \mathbf{X}^T\mathbf{V}^{-1}\frac{\partial \mathbf{V}}{\partial \theta_i}\right)$$

Secondly, the derivative of the third term is

$$l = -\frac{1}{2} trace\left(\mathbf{V}_R^{-1}\mathbf{Y}\mathbf{Y}^T\right)$$

$$\frac{\partial l}{\partial \theta_i} = \frac{1}{2} trace\left(\mathbf{V}_R^{-1}\frac{\partial \mathbf{V}}{\partial \theta_i}\mathbf{V}_R^{-1}\mathbf{Y}\mathbf{Y}^T\right)$$

The last step is not easily proven, except for diligently applying the product rule and seeing a lot of terms cancel. Putting these two results together with the derivative of the normal likelihood gives us:

$$\frac{\partial (L_{ReML})}{\partial \theta_i} = -\frac{P}{2} trace\left(\mathbf{V}^{-1}\frac{\partial \mathbf{V}}{\partial \theta_i}\right)$$

$$+\frac{1}{2} trace\left(\mathbf{V}_R^{-1}\frac{\partial \mathbf{V}}{\partial \theta_i}\mathbf{V}_R^{-1}\mathbf{Y}\mathbf{Y}^T\right)$$

$$+\frac{P}{2} trace\left(\mathbf{V}^{-1}\mathbf{X}\left(\mathbf{X}^T\mathbf{V}^{-1}\mathbf{X}\right)^{-1} \mathbf{X}^T\mathbf{V}^{-1}\frac{\partial \mathbf{V}}{\partial \theta_i}\right)$$

$$= -\frac{P}{2} trace\left(\mathbf{V}_R^{-1}\frac{\partial \mathbf{V}}{\partial \theta_i}\right) + \frac{1}{2} trace\left(\mathbf{V}_R^{-1}\frac{\partial \mathbf{V}}{\partial \theta_i}\mathbf{V}_R^{-1}\mathbf{Y}\mathbf{Y}^T\right)$$

Derivates for specific parameters

From the general term for the derivative of the log-likelihood, we can derive the specific expressions for each parameter. In general, we model the co-variance matrix of the data $\mathbf{V}$ as:

$$\mathbf{V} = s\mathbf{Z}\mathbf{G}(\boldsymbol{\theta}_h)\mathbf{Z}^T + S\sigma_\epsilon^2 s = exp(\theta_s)\sigma_\epsilon^2 = exp(\theta_\epsilon)$$

Where θ_s is the signal scaling parameter, the θ_ϵ the noise parameter. We are using the exponential of the parameter, to ensure that the noise variance and the scaling will always be strictly positive. When taking the derivatives, we use the simple rule of $\partial exp(x)/\partial x = exp(x)$. Each model provides the partial derivaratives for $\mathbf{G}$ in respect to the model parameters (see above). From this we can easily obtain the derviative of $\mathbf{V}$

$$\frac{\partial \mathbf{V}}{\partial \theta_h} = \mathbf{Z}\frac{\partial \mathbf{G}(\boldsymbol{\theta_h})}{\partial \theta_h}\mathbf{Z}^T exp(\theta_s).$$

The derivate in respect to the noise parameter

$$\frac{\partial \mathbf{V}}{\partial \theta_\epsilon} = \mathbf{S} exp(\theta_\epsilon).$$

And in respect to the signal scaling parameter

$$\frac{\partial \mathbf{V}}{\partial \theta_s} = \mathbf{Z}\mathbf{G}(\boldsymbol{\theta}_h)\mathbf{Z}^T exp(\theta_s).$$

Conjugate Gradient descent

One way of optiminzing the likelihood is simply using the first derviative and performing a conjugate-gradient descent algorithm. For this, the routines `pcm_likelihoodIndivid` and `pcm_likelihoodGroup` return the negative log-likelihood, as well as a vector of the first derivatives of the negative log-likelihood in respect to the parameter. The implementation of conjugate-gradient descent we are using here based on Carl Rassmussen's excellent function `minimize`.

Newton Raphson algorithm

A alternative to conjugate gradients, which can be considerably faster, are optimisation routines that exploit the matrix of second derivatives of the log-liklihood. The local curvature information is then used to "jump" to suspected bottom of the bowl of the likelihood surface. The negative expected second derivative of the restricted log-likelihood, also called Fisher-information can be calculated efficiently from terms that we needed to compute for the first derivative anyway:

$$\mathbf{F}_{i,j}(\theta) = -E\left[\frac{\partial^2}{\partial\theta_i\partial\theta_j}L_{ReML}\right] = \frac{P}{2}trace\left(\mathbf{V}_R^{-1}\frac{\partial\mathbf{V}}{\partial\theta_i}\mathbf{V}_R^{-1}\frac{\partial\mathbf{V}}{\partial\theta_j}\right).$$

The update then uses a slightly regularized version of the second derviate to compute the next update on the parameters.

$$\boldsymbol{\theta}^{u+1} = \boldsymbol{\theta}^u - \left(\mathbf{F}(\boldsymbol{\theta}^u) + \mathbf{I}\lambda\right)^{-1}\frac{\partial L_{ReML}}{\partial\boldsymbol{\theta}^u}.$$

Because the update can become unstable, we are regularising the Fisher information matrix by adding a small value to the diagonal, similar to a Levenberg regularisation for least-square problems. If the likelihood increased, λ is decreases, if the liklihood accidentially decreased, then we take a step backwards and increase λ. The algorithm is implemented in `pcm_NR` .

Choosing an optimisation algorithm

While the Newton-Raphson algorithm can be considerably faster for many problems, it is not always the case. Newton-Raphson usually arrives at the goal with many fewer steps than conjugate gradient descent, but on each step it has to calculate the matrix second derviatives, which grows in the square of the number of parameters . So for highly-parametrized models, the simple conjugate gradient algorithm is better. You can set for each model the desired algorithm by setting the field `M.fitAlgorithm = 'NR';` for Newton-Raphson and `M.fitAlgorithm = 'minimize';` for conjugate gradient descent. If no such field is given, then fitting function will call `M=pcm_optimalAlgorithm(M)` to obtain a guess of what will be the best algorithm for the problem. While this function provides a good heuristic strategy, it is recommended to try both and compare both the returned likelihood and time. Small differences in the likelihood (< 0.1) are due to different stopping criteria and should be of no concern. Larger differences can indicate failed convergence.

Acceleration of matrix inversion

When calculating the likelihood or the derviatives of the likelihood, the inverse of the variance-covariance has to be computed. Because this can become quickly very costly (especially if original time series data is to be fitted), we can exploit the special structure of $\mathbf{V}$ to speed up the computation:

$$\begin{aligned}\mathbf{V}^{-1} &= \left(s\mathbf{Z}\mathbf{G}\mathbf{Z}^T + \mathbf{S}\sigma_\varepsilon^2\right)^{-1} \\ &= \mathbf{S}^{-1}\sigma_\varepsilon^{-2} - \mathbf{S}^{-1}\mathbf{Z}\sigma_\varepsilon^{-2}\left(s^{-1}\mathbf{G}^{-1} + \mathbf{Z}^T\mathbf{S}^{-1}\mathbf{Z}\sigma_\varepsilon^{-2}\right)^{-1}\mathbf{Z}^T\mathbf{S}^{-1}\sigma_\varepsilon^{-2} \\ &= \left(\mathbf{S}^{-1} - \mathbf{S}^{-1}\mathbf{Z}\left(s^{-1}\mathbf{G}^{-1}\sigma_\varepsilon^2 + \mathbf{Z}^T\mathbf{S}^{-1}\mathbf{Z}\right)^{-1}\mathbf{Z}^T\mathbf{S}^{-1}\right)/\sigma_\varepsilon^2\end{aligned}$$

With pre-inversion of $\mathbf{S}$ (which can occur once outside of the iterations), we make a $N \times N$ matrix inversion into a $K \times K$ matrix inversion.

References

Cai, M.B., Schuck, N.W., Pillow, J., Niv, Y., 2016. A bayesian method for reducing bias in neural representational similarity analysis, in: Advances in Neural Information Processing Systems. pp. 4952–4960. doi:10.1101/073932

deCharms, R.C., Zador, A., 2000. Neural representation and the cortical code. Annu Rev Neurosci 23, 613–47. doi:10.1146/annurev.neuro.23.1.613

DiCarlo, J.J., Cox, D.D., 2007. Untangling invariant object recognition. Trends Cogn Sci 11, 333–41. doi:10.1016/j.tics.2007.06.010

DiCarlo, J.J., Zoccolan, D., Rust, N.C., 2012. How does the brain solve visual object recognition? Neuron 73, 415–34. doi:10.1016/j.neuron.2012.01.010

Diedrichsen, J., Kriegeskorte, N., 2017. Representational models: A common framework for understanding encoding, pattern-component, and representational-similarity analysis. PLoS Comput Biol 13, e1005508. doi:10.1371/journal.pcbi.1005508

Diedrichsen, J., Ridgway, G.R., Friston, K.J., Wiestler, T., 2011. Comparing the similarity and spatial structure of neural representations: A pattern-component model. Neuroimage 55, 1665–78. doi:10.1016/j.neuroimage.2011.01.044

Diedrichsen, J., Yokoi, A., Arbuckle, S.A., 2017. Pattern component modeling: A flexible approach for understanding the representational structure of brain activity patterns. Neuroimage. doi:10.1016/j.neuroimage.2017.08.051

Diedrichsen, J., Zareamoghaddam, H., Provost, S., 2016. The distribution of crossvalidated mahalanobis distances. ArXiv.

Dumoulin, S.O., Wandell, B.A., 2008. Population receptive field estimates in human visual cortex. Neuroimage 39, 647–60. doi:10.1016/j.neuroimage.2007.09.034

Eisenberg, M., Shmuelof, L., Vaadia, E., Zohary, E., 2010. Functional organization of human motor cortex: Directional selectivity for movement. J Neurosci 30, 8897–905. doi:30/26/8897 [pii] 10.1523/JNEUROSCI.0007-10.2010

Ejaz, N., Hamada, M., Diedrichsen, J., 2015. Hand use predicts the structure of representations in sensorimotor cortex. Nat Neurosci 18, 1034–40. doi:10.1038/nn.4038

Eklund, A., Andersson, M., Josephson, C., Johannesson, M., Knutsson, H., 2012. Does parametric fMRI analysis with spm yield valid results? An empirical study of 1484 rest datasets. Neuroimage 61, 565–78. doi:10.1016/j.neuroimage.2012.03.093

Haxby, J.V., Gobbini, M.I., Furey, M.L., Ishai, A., Schouten, J.L., Pietrini, P., 2001. Distributed and overlapping representations of faces and objects in ventral temporal cortex. Science 293, 2425–30. doi:10.1126/science.1063736

Huth, A.G., Heer, W.A. de, Griffiths, T.L., Theunissen, F.E., Gallant, J.L., 2016. Natural speech reveals the semantic maps that tile human cerebral cortex. Nature 532, 453–8. doi:10.1038/nature17637

Kay, K.N., Naselaris, T., Prenger, R.J., Gallant, J.L., 2008. Identifying natural images from human brain activity. Nature 452, 352–5. doi:10.1038/nature06713

Khaligh-Razavi, S.M., Kriegeskorte, N., 2014. Deep supervised, but not unsupervised, models may explain

it cortical representation. PLoS Comput Biol 10, e1003915. doi:10.1371/journal.pcbi.1003915

Kriegeskorte, N., 2011. Pattern-information analysis: From stimulus decoding to computational-model testing. Neuroimage 56, 411–21. doi:10.1016/j.neuroimage.2011.01.061

Norman, K.A., Polyn, S.M., Detre, G.J., Haxby, J.V., 2006. Beyond mind-reading: Multi-voxel pattern analysis of fMRI data. Trends Cogn Sci 10, 424–30. doi:10.1016/j.tics.2006.07.005 [doi]

Pereira, F., Mitchell, T., Botvinick, M., 2009. Machine learning classifiers and fMRI: A tutorial overview. Neuroimage 45, S199–209. doi:10.1016/j.neuroimage.2008.11.007

Walther, A., Nili, H., Ejaz, N., Alink, A., Kriegeskorte, N., Diedrichsen, J., 2016. Reliability of dissimilarity measures for multi-voxel pattern analysis. Neuroimage 137, 188–200. doi:10.1016/j.neuroimage.2015.12.012