GLMsingle

Release 0.0.1

Prince, J.S., Charest, I., Kurzawski, J.W., Pyles, J.A., Tarr, M.J., Ka

Mar 14, 2024

CONTENTS:

1 MATLAB doc 1

2 Python doc 9

3 WIKI 11

4 Indices and tables 23

Python Module Index 25

MATLAB Module Index 27

Index 29

CHAPTER

ONE

MATLAB DOC

matlab.**GLMestimatesingletrial**(*design*, *data*, *stimdur*, *tr*, *outputdir*, *opt*)

USAGE:

```
[results,resultsdesign] = GLMestimatesingletrial(design,data,stimdur,tr,outputdir,
↪opt)
```

<design> is the experimental design. There are two possible cases:

1. A where A is a matrix with dimensions time x conditions. Each column should be zeros except for ones indicating condition onsets.
2. {A1 A2 A3 ... An} where each of the A's are like the previous case. The different A's correspond to different runs, and different runs can have different numbers of time points. However, all A's must have the same number of conditions.

Note that we ultimately compute single-trial response estimates (one estimate for each condition onset), and these will be provided in chronological order. However, by specifying that a given condition occurs more than one time over the course of the experiment, this information can and will be used for cross-validation purposes.

<data> is the time-series data with dimensions X x Y x Z x time or a cell vector of elements that are each X x Y x Z x time. XYZ can be collapsed such that the data are given as a 2D matrix (units x time), which is useful for surface-format data. The dimensions of <data> should mirror that of <design>. For example, <design> and <data> should have the same number of runs, the same number of time points, etc. <data> should not contain any NaNs. We automatically convert <data> to single format if not already in single format.

<stimdur> is the duration of a trial in seconds. For example, 3.5 means that you expect the neural activity from a given trial to last for 3.5 s.

<tr> is the sampling rate in seconds. For example, 1 means that we get a new time point every 1 s. Note that <tr> applies to both <design> and <data>.

<outputdir> (optional) is a directory to which files will be written. (If the directory does not exist, we create it; if the directory already exists, we delete its contents so we can start fresh.) If you set <outputdir> to NaN, we will not create a directory and no files will be written. If you provide {outputdir figuredir}, we will save the large output files to <outputdir> and the small figure files to <figuredir> (either or both can be NaN). Default is 'GLMestimatesingletrialoutputs' (created in the current working directory).

<opt> (optional) is a struct with the following optional fields:

*** MAJOR, HIGH-LEVEL FLAGS ***

<wantlibrary> (optional) is:

- 0 means use an assumed HRF
- 1 means determine the best HRF for each voxel using the library-of-HRFs approach

- Default: 1.

<wantglmdenoise> (optional) is

- 0 means do not perform GLMdenoise
- 1 means perform GLMdenoise
- Default: 1.

<wantfracridge> (optional) is

- 0 means do not perform ridge regression
- 1 means perform ridge regression
- Default: 1.

<chunknum> (optional) is the number of voxels that we will process at the same time. This number should be large in order to speed computation, but should not be so large that you run out of RAM. Note that the <chunknum> that you choose does not affect any of the results or outputs; it merely affects execution time and RAM usage. Default: 50000.

<xvalscheme> (optional) is a cell vector of vectors of run indices, indicating the cross-validation scheme. For example, if we have 8 runs, we could use {[1 2] [3 4] [5 6] [7 8]} which indicates to do 4 folds of cross-validation, first holding out the 1st and 2nd runs, then the 3rd and 4th runs, etc. Default: {[1] [2] [3] ... [n]} where n is the number of runs.

<sessionindicator> (optional) is 1 x n (where n is the number of runs) with positive integers indicating the run groupings that are interpreted as "sessions". The purpose of this input is to allow for session-wise z-scoring of single-trial beta weights for the purposes of hyperparameter evaluation. For example, if you are analyzing data aggregated from multiple scan sessions, you may want beta weights to be z-scored per voxel within each session in order to compensate for any potential gross changes in betas across scan sessions. Note that the z-scoring has effect only INTERNALLY: it is used merely to calculate the cross-validation performance and the associated hyperparameter selection; the outputs of this function do not reflect z-scoring, and the user may wish to post-hoc apply z-scoring. Default: 1*ones(1,n) which means to interpret all runs as coming from the same session.

*** I/O FLAGS ***

<wantfileoutputs> (optional) is a logical vector [A B C D] indicating which of the four model types to save to disk (assuming that they are computed).

- A = 0/1 for saving the results of the ONOFF model
- B = 0/1 for saving the results of the FITHRF model
- C = 0/1 for saving the results of the FITHRF_GLMDENOISE model
- D = 0/1 for saving the results of the FITHRF_GLMDENOISE_RR model
- Default: [1 1 1 1] which means save all computed results to disk.

<wantmemoryoutputs> (optional) is a logical vector [A B C D] indicating which of the four model types to return in the output <results>. The user must be careful with this, as large datasets can require a lot of RAM. If you do not request the various model types, they will be cleared from memory (but still potentially saved to disk). Default: [0 0 0 1] which means return only the final type-D model.

*** GLM FLAGS ***

<extraregressors> (optional) is time x regressors or a cell vector of elements that are each time x regressors. The dimensions of <extraregressors> should mirror that of <design> (i.e. same number of runs, same number of time points). The number of extra regressors does not have to be the same

across runs, and each run can have zero or more extra regressors. If [] or not supplied, we do not use extra regressors in the model.

<maxpolydeg> (optional) is a non-negative integer with the maximum polynomial degree to use for polynomial nuisance functions, which are used to capture low-frequency noise fluctuations in each run. Can be a vector with length equal to the number of runs (this allows you to specify different degrees for different runs). Default is to use round(L/2) for each run where L is the duration in minutes of a given run.

<wantpercentbold> (optional) is whether to convert amplitude estimates to percent BOLD change. This is done as the very last step, and is accomplished by dividing by the absolute value of 'meanvol' and multiplying by 100. (The absolute value prevents negative values in 'meanvol' from flipping the sign.) Default: 1.

*** HRF FLAGS ***

<hrftoassume> (optional) is time x 1 with an assumed HRF that characterizes the evoked response to each trial. We automatically divide by the maximum value so that the peak is equal to 1. Default is to generate a canonical HRF (see getcanonicalhrf.m). Note that the HRF supplied in <hrftoassume> is used in only two instances: (1) it is used for the simple ONOFF type-A model, and (2) if the user sets <wantlibrary> to 0, it is also used for the type-B, type-C, and type-D models.

<hrflibrary> (optional) is time x H with H different HRFs to choose from for the library-of-HRFs approach. We automatically normalize each HRF to peak at 1. Default is to generate a library of 20 HRFs (see getcanonicalhrflibrary.m). Note that if <wantlibrary> is 0, <hrflibrary> is clobbered with the contents of <hrftoassume>, which in effect causes a single assumed HRF to be used.

*** DIAGNOSTIC MODEL (FIR) FLAGS ***

<firdelay> (optional) is the total time duration in seconds over which to estimate the run-wise FIR model (where we assume an ONOFF design matrix in which all conditions are collapsed together). Default: 30.

<firpct> (optional) is a percentile threshold. We average the FIR model R2 values across runs and then select voxels that pass this threshold. These voxels are used for the FIR timecourse summaries. Default: 99.

*** MODEL TYPE A (ONOFF) FLAGS ***

(none)

*** MODEL TYPE B (FITHRF) FLAGS ***

<wantlss> (optional) is 0/1 indicating whether "least-squares-separate" estimates are desired. If 1, then the type-B model will be estimated using the least-squares- separate method (as opposed to ordinary least squares). Default: 0.

*** MODEL TYPE C (FITHRF_GLMDENOISE) FLAGS ***

<numpcstotry> (optional) is a non-negative integer indicating the maximum number of PCs to enter into the model. Default: 10.

<brainthresh> (optional) is [A B] where A is a percentile for voxel intensity values and B is a fraction to apply to the percentile. These parameters are used in the selection of the noise pool. Default: [99 0.1].

<brainR2> (optional) is an R^2 value (percentage). After fitting the type-A model, voxels whose R^2 is below this value are allowed to enter the noise pool. Default is [] which means to automatically determine a good value.

<brainexclude> (optional) is X x Y x Z (or XYZ x 1) with 1s indicating voxels to specifically exclude when selecting the noise pool. 0 means all voxels can be potentially chosen. Default: 0.

<pcR2cutoff> (optional) is an R^2 value (percentage). To decide the number of PCs to include, we examine a subset of the available voxels. Specifically, we examine voxels whose type-A model R^2 is above <pcR2cutoff>. Default is [] which means to automatically determine a good value.

<pcR2cutoffmask> (optional) is X x Y x Z (or XYZ x 1) with 1s indicating all possible voxels to consider when selecting the subset of voxels. 1 means all voxels can be potentially selected. Default: 1.

<pcstop> (optional) is

- A: a number greater than or equal to 1 indicating when to stop adding PCs into the model. For example, 1.05 means that if the cross-validation performance with the current number of PCs is within 5% of the maximum observed, then use that number of PCs. (Performance is measured relative to the case of 0 PCs.) When <pcstop> is 1, the selection strategy reduces to simply choosing the PC number that achieves the maximum. The advantage of stopping early is to achieve a selection strategy that is robust to noise and shallow performance curves and that avoids overfitting.
- -B: where B is the number of PCs to use for the final model. B can be any integer between 0 and opt.numpcstotry. Note that if -B case is used, cross-validation is NOT performed for the type-C model, and instead we blindly use B PCs.
- Default: 1.05.

*** MODEL TYPE D (FITHRF_GLMDENOISE_RR) FLAGS ***

<fracs> (optional) is a vector of fractions that are greater than 0 and less than or equal to 1. We automatically sort in descending order and ensure the fractions are unique. These fractions indicate the regularization levels to evaluate using fractional ridge regression (fracridge) and cross-validation. Default: fliplr(.05:.05:1). A special case is when <fracs> is specified as a single scalar value. In this case, cross-validation is NOT performed for the type-D model, and we instead blindly use the supplied fractional value for the type-D model.

<wantautoscale> (optional) is whether to automatically scale and offset the model estimates from the type-D model to best match the unregularized estimates. Default: 1.

This function computes up to four model outputs (called type-A (ONOFF), type-B (FITHRF), type-C (FITHRF_GLMDENOISE), and type-D (FITHRF_GLMDENOISE_RR)), and either saves the model outputs to disk, or returns them in <results>, or both, depending on what the user specifies.

There are a variety of cases that you can achieve. Here are some examples:

- wantlibrary=1, wantglmdenoise=1, wantfracridge=1 [Default]
  - A = simple ONOFF model
  - B = single-trial estimates using a tailored HRF for every voxel
  - C = like B but with GLMdenoise regressors added into the model
  - D = like C but with ridge regression regularization (tailored to each voxel)
- wantlibrary=0

  A fixed assumed HRF is used in all model types.
- wantglmdenoise=0, wantfracridge=0

  Model types C and D are not computed.
- wantglmdenoise=0, wantfracridge=1

  Model type C is not computed; model type D is computed using 0 GLMdenoise regressors.
- wantglmdenoise=1, wantfracridge=0

Model type C is computed; model type D is not computed.

- wantlss=1

  Model type B is computed, but using least-squares-separate instead of OLS. Other model types, if computed, use OLS.

Note that if you set wantglmdenoise=1, you MUST have repeats of conditions and an associated cross-validation scheme (<opt.xvalscheme>), UNLESS you specify opt.pcstop = -B. In other words, you can perform wantglmdenoise without any cross-validation, but you need to provide opt.pcstop = -B.

Note that if you set wantfracridge=1, you MUST have repeats of conditions and an associated cross-validation scheme (<opt.xvalscheme>), UNLESS you specify a single scalar opt.fracs. In other words, you can perform wantfracridge without any cross-validation, but you need to provide opt.fracs as a scalar.

*** OUTPUTS: ***

We return model results in the output variable <results>. These results are saved to disk in files called 'TYPEA…', 'TYPEB…', and so on. There are various outputs for each of the four model types:

<modelmd> is either

1. the HRF (time x 1) and ON-OFF beta weights (X x Y x Z)
2. the full set of single-trial beta weights (X x Y x Z x TRIALS)

<R2> is model accuracy expressed in terms of R^2 (percentage).

<R2run> is R2 separated by run

<meanvol> is the mean of all volumes

<FitHRFR2> is the R2 for each of the different HRFs in the library

<FitHRFR2run> is separated by run

<HRFindex> is the 1-index of the best HRF

<HRFindexrun> is HRFindex separated by run

<noisepool> indicates voxels selected for the noise pool

<pcregressors> indicates the full set of candidate GLMdenoise regressors that were found

<glmbadness> is the cross-validation results for GLMdenoise

<pcvoxels> is the set of voxels used to summarize GLMdenoise cross-validation results

<xvaltrend> is the summary GLMdenoise cross-validation result on which pcnum selection is done

<pcnum> is the number of PCs that were selected for the final model

<FRACvalue> is the fractional ridge regression regularization level chosen for each voxel

<rrbadness> is the cross-validation results for the ridge regression

<scaleoffset> is the scale and offset applied to RR estimates to best match the unregularized result

Note that not all outputs exist for every model type.

We also return design-related results in the output variable <resultsdesign>. These results are saved to disk to a file called 'DESIGNINFO…'. The outputs include:

<design> is as specified by the user (with possibly some minor regularization)

<stimdur> is as specified by the user

<tr> is as specified by the user

<opt> is as specified by the user (with possibly some minor regularization)

<designSINGLE> is a single-trial design matrix corresponding to <design>

<stimorder> is a row vector indicating which condition (1-indexed) each trial (in chronological order) belongs to

<numtrialrun> is a row vector with the number of trials in each run

<condcounts> is a row vector with the number of trials associated with each condition

<condinruns> is a row vector with the number of runs that each condition shows up in

<endbuffers> is a row vector with the number of seconds after the last trial onset in each run

We also return diagnostic FIR-related results — these are saved to disk to a file called 'RUNWISEFIR...'. The outputs include:

<firR2> is the R2 of the FIR model for each run (X x Y x Z x run).

<firtcs> is the estimated FIR timecourse for each run (X x Y x Z x 1 x time x run). Note that the first time point is coincident with trial onset and the time points are at the sampling rate corresponding to <tr>.

<firavg> is the estimated FIR timecourse in each run (time x run). These are obtained by calculating the median timecourse across the "best" voxels (see opt.firpct).

<firgrandavg> is the average of <firavg> across runs (time x 1).

*** FIGURES: ***

If <outputdir> is set appropriately, we will generate a variety of useful figures and save them to disk. Note that if you provide your data in 3D format (e.g. X x Y x Z x T), we will be able to write out a number of additional useful slice inspections that you will not get if you provide your data in collapsed format (e.g. XYZ x T).

betaviz_type[B,C,D].png - an image visualization of betas obtained under the type-B, type-C, and type-D models. The matrix dimensions are 1,000 voxels x trials. We choose 1,000 voxels equally spaced in descending order from the 100th to 75th percentiles of the R^2 values produced by the ONOFF model. The colormap is cmapsign4.m (blueish colors to black to reddish colors) from -X to X where X is the 99th percentile of the absolute value of the betas in the first model that is actually computed (typically, this will be the type-B model).

dmetric_type[B,C,D].png - a "deviation from zero" metric calculated based on the betas obtained under the type-B, type-C, and type-D models. We use a hot colormap ranging between the min and max of the values obtained for the first model that is computed (typically, this will be the type-B model).

FRACvalue.png - chosen fractional ridge regression value (copper colormap between 0 and 1)

HRFindex.png - 1-index of chosen HRF (jet colormap between 1 and the number of HRFs in the library)

meanvol.png - simply the mean across all data volumes

noisepool.png - voxels selected for the noise pool (white means selected)

onoffR2_vs_HRFindex.png - scatter plot of the R^2 of the ONOFF model against the chosen HRF index. All voxels are shown. A small amount of jitter is added to the HRF index in order to aid visibility.

onoffR2.png - R^2 of the ONOFF model (sqrt hot colormap between 0% and 100%)

onoffR2hist.png - depicts the finding of an automatic threshold on the ONOFF model R^2. This is used in determining the noise pool (but can be overridden by opt.brainR2).

pcvoxels.png - voxels used to summarize GLMdenoise cross-validation results (white means selected)

runwiseFIR_R2_runXX.png - for each run, the R^2 of the diagnostic FIR model (sqrt hot colormap between 0% and 100%)

runwiseFIR_R2_runavg.png - simply the average of the R^2 across runs

runwiseFIR.png - Upper left shows run-wise FIR estimates. The estimates reflect the mean FIR timecourse averaged across a set of "best" voxels (see opt.firpct). The mean of these mean FIR timecourses across runs is indicated by the thick red line. Upper right shows FIR amplitudes at the peak time observed in the grand mean timecourse (indicated by the dotted black line). Bottom left shows the HRFs in the library as colored lines and the "assumed HRF" as a thick black line. Note that these reflect any user-specified customization (as controlled via opt.hrftoassume and opt.hrflibrary).

runwiseFIR_summaryvoxels.png - the voxels that were used for runwiseFIR.png

typeD_R2_runXX.png - the R^2 of the final type-D model computed using data from individual runs (sqrt hot colormap between 0% and 100%)

typeD_R2.png - the R^2 of the final type-D model (using all data)

xvaltrend.png - shows the cross-validation performance for different numbers of GLMdenoise regressors. Note that the y-axis units are correct but not easy to interpret.

CHAPTER

TWO

PYTHON DOC

class glmsingle.glmsingle.**GLM_single**(*params=None*)

Bases: object

fit(*design*, *data*, *stimdur*, *tr*, *outputdir=None*, *figuredir=None*)

2.1 Arguments:

<design> is the experimental design. There are two possible cases: 1. A where A is a matrix with dimensions time x conditions.

Each column should be zeros except for ones indicating condition onsets.

2. **[A1, A2, ... An] where each of the A's are like the previous case.**
   The different A's correspond to different runs, and different runs can have different numbers of time points. However, all A's must have the same number of conditions.

Note that we ultimately compute single-trial response estimates (one estimate for each condition onset), and these will be provided in chronological order. However, by specifying that a given condition occurs more than one time over the course of the experiment, this information can and will be used for cross-validation purposes.

<data> is the time-series data with dimensions X x Y x Z x time or a
list vector of elements that are each X x Y x Z x time. XYZ can be collapsed such that the data are given as a 2D matrix (units x time), which is useful for surface-format data. The dimensions of <data> should mirror that of <design>. For example, <design> and <data> should have the same number of runs, the same number of time points, etc. <data> should not contain any NaNs. We automatically convert <data> to single format if not already in single format. <stimdur> is the duration of a trial in seconds. For example, 3.5 means that you expect the neural activity from a given trial to last for 3.5 s.

<tr> is the sampling rate in seconds. For example, 1 means that we get
a new time point every 1 s. Note that <tr> applies to both <design> and <data>.

<outputdir> (optional) is a directory to which data will be written.
(If the directory does not exist, we create it; if the directory already exists, we delete its contents so we can start fresh.) If you set <outputdir> to None, we will not create a directory and no files will be written. Default is 'GLMestimatesingletrialoutputs' (created in the current working directory).

<figuredir> (optional) is a directory to which figures will be written.
(If the directory does not exist, we create it; if the directory already exists, we delete its contents so we can start fresh.) If you set <figuredir> to None, we will not create a directory and no files will be written. Default is 'GLMestimatesingletrialfigures' (created in the current working directory).

2.2 Returns:

There are various outputs for each of the four model types:

<modelmd> is either

(1) the HRF (time x 1) and ON-OFF beta weights (X x Y x Z)

(2) the full set of single-trial beta weights (X x Y x Z x TRIALS)

<R2> is model accuracy expressed in terms of R^2 (percentage).

<R2run> is R2 separated by run

<meanvol> is the mean of all volumes

<FitHRFR2> is the R2 for each of the different HRFs in the library

<FitHRFR2run> is separated by run

<HRFindex> is the 1-index of the best HRF

<HRFindexrun> is HRFiniex separated by run

<noisepool> indicates voxels selected for the noise pool

<pcregressors> indicates the full set of candidate GLMdenoise
regressors that were found

<glmbadness> is cross-validation results for GLMdenoise

<pcvoxels> is the set of voxels used to summarize GLMdenoise
cross-validation results

<xvaltrend> is the summary GLMdenoise cross-validation result on which
pcnum selection is done

<pcnum> is the number of PCs that were selected for the final model

<FRACvalue> is the fractional regularization level chosen for each
voxel

<scaleoffset> is the scale and offset applied to RR estimates to best
match the unregularized result

CHAPTER

THREE

WIKI

3.1 Basic information

GLMsingle is introduced and described in the following paper:

Prince, J.S., Charest, I., Kurzawski, J.W., Pyles, J.A., Tarr, M., Kay, K.N. Improving the accuracy of single-trial fMRI response estimates using GLMsingle. *eLife* (2022).

GLMsingle is an analysis technique (an algorithm) for obtaining accurate estimates of single-trial beta weights in fMRI data.

If you have questions or discussion points, please use the Discussions feature of this github repository, or if you find a bug, please let us know by raising a Github Issue.

3.2 Example scripts

We provide a number of example scripts that demonstrate usage of GLMsingle.

You can browse these example scripts here:

- Python - example 1
- Python - example 2
- MATLAB - example 1
- MATLAB - example 2

3.3 FAQ

3.3.1 What are the main things that GLMsingle does?

The main components of GLMsingle include:

1. a "library of HRFs" technique where an empirically derived set of HRF timecourses (from the subjects in the NSD dataset) are used as potential HRFs for each voxel in your dataset,
2. the GLMdenoise technique where data-driven nuisance regressors are obtained and added into the GLM (using cross-validation for determining how many nuisance regressors to add), and
3. ridge regression as a way to improve robustness of single-trial beta estimates. The technique relies on heavy amounts of computation, but is implemented in a relatively efficient manner, and could therefore serve as a go-to

tool for deriving beta estimates from many types of fMRI experimental data, and especially for condition-rich designs with few repeats per condition.

3.3.2 How does GLMsingle achieve denoising?

We can consider each of the three components.

1. Mismodeling the timecourse of a voxel can lead to suboptimal results (i.e. results that are, in a sense, "noisy"); GLMsingle attempts to remedy this by using a well regularized HRF selection strategy where a simple "index" is learned for each voxel.
2. fMRI data suffer from very large amounts of spatially correlated noise (e.g. due to head motion, physiological noise, etc.) — by deriving these noise sources from the data themselves, GLMsingle is able to provide some amount of "modeling" out the noise.
3. Finally, fMRI designs often involve substantial overlap of the response across successive trials. From a statistical perspective, this is going to hurt estimation efficiency. Ridge regression is a method that induces some amount of shrinkage bias to help improve out-of-sample generalization.

Note that these three components are heterogenous with regards to the idea of "fitting" the data. For the HRF component, the presumption is that we probably have at least enough data to estimate an HRF index for each voxel; hence the philosophy is to indeed fit the data. For the GLMdenoise component, the goal is to try to add regressors to the model to better fit the data, while acknowledging that at some point, overfitting is going to occur (and the algorithm attempts to determine that point). For the ridge regression component, things are a bit different. The point of ridge regression is to allow for the possibility that due to measurement noise, single-trial beta estimates are inaccurate and we want to actually limit the extent to which we fit the data. Thus, somewhat counter-intuitively, ridge regression *increases* the residuals of the model fit.

3.3.3 I'm concerned about the noise pool. What if there are signals in there?

This is an interesting issue to think about. The short answer is that GLMsingle guards against improper use of nuisance regressors through cross-validation. If, for some reason, there are valid experimental signals being learned from the noise pool, GLMsingle will tend to not use these signals since they will likely degrade the cross-validation performance.

Note that even if the noise pool includes "good" voxels, improvement in beta estimates are still possible. This is something demonstrated in Figure 6 of Kay et al. Frontiers 2013, where we deliberately included the entire brain in the noise pool (just to see what would happen). The intuition is that results will depend on the specific mixture of noise and signal being learned in the data-derived nuisance regressors. If the nuisance regressors have a little bit of signal mixed in, they can still be useful as an approximate model of the noise.

3.3.4 In GLMsingle, the GLMdenoise and ridge regression (RR) components of the method require experimental conditions to repeat across runs. How should I think about whether this is appropriate for my experiment?

In order to determine the hyperparameter settings, it is indeed necessary for some amount of repeated trials to exist in the data that are given to GLMsingle. It is true that, in a sense, any component of the fMRI data that does not repeat for the repeated trials associated with a condition is thought of a "noise" by the cross-validation process. Hence, there is some potential risk that components of interest might be removed. However, for the most part, we believe that the risk of this is actually minimal. This can be seen by thinking about the nature of the effects of GLMdenoise and ridge regression. GLMdenoise is simply trying to capture variance that appears to be spatially/globally correlated across voxels in the dataset; hence, its flexibility is actually quite limited. Ridge regression can only dampen the instabilities of beta estimates that persists across temporally nearby trials and in a uniform manner for all trials in the experiment; hence, its flexibility is also quite limited.

3.3.5 I noticed that GLMsingle needs some conditions to repeat across runs. But my experiment is not fully balanced in this regard (or I have just a few repeats). Is this a problem?

In general, we don't think that perfect balancing is critical. The reason lies in thinking about the nature of the technique — the repeats are simply used to set the hyperparameters (number of nuisance regressors; fractional regularization amount for each voxel). In our experience, even just a handful of repeats appears to be sufficient to robustly guide the estimation of the hyperparameters. Thus, even if you can code only a subset of the trials in your experiment as condition repeats, this may be sufficient to guide the hyperparameter estimation, and the overall results from GLMsingle might be quite good. (In fact, there are very few repeats in the NSD and BOLD5000 datasets, which are the main two datasets demonstrated in the GLMsingle pre-print.)

3.4 Pre-processing choices

GLMsingle should be applied to pre-processed fMRI data. But which types of pre-processing are suitable? Please review the GLMsingle manuscript for detailed information, but here we comment on some types of pre-processing that have come up in conversations with users.

- Skull stripping / brain masking – This is totally fine. In fact, one should be able to run GLMsingle even on surface-prepared data (where voxels not in gray matter are already excluded).
- High-pass temporal filtering – This is unnecessary. In terms of high-pass filtering (where low frequency time series components are removed), GLMsingle automatically includes a set of polynomials to model the baseline signal level which may drift over the course of each run. Thus, one does not need to high-pass filter the fMRI data in pre-processing. (However, if you do include it, it actually won't make a big difference, since the polynomials will essentially model the components that were removed. But note that this assumes that the mean of each voxel has at least been retained, since GLMsingle uses this spatial information.)
- Low-pass temporal filtering – This is not very typical, and also not necessary. The HRF timecourse and ridge regression components of GLMsingle are intended to compensate for high temporal frequency noise in the data.
- Projecting out nuisance components – Pipelines often remove/project-out nuisance components (e.g. motion regressors, ICA derived noise, low-rank approaches like NORDIC, etc.) from fMRI time series in the course of pre-processing. While you can do this, and GLMsingle will likely still work, this is not quite recommended. This is because such pre-filtering approaches risk bias. And, GLMsingle's approach is to attempt to learn these types of nuisance components from the data themselves, so it is a bit ironic to use both approaches simultaneously. Moreover, it is possible that using a pre-filtering approach will actually cause strange complications with GLMsingle, so beware.
- Spatial smoothing – This is fine, if you want to do this as a pre-processing step. Note that you might want to consider running GLMsingle on non-smoothed data, and then decide whether to apply spatial smoothing at a later analysis step (e.g. on the single-trial betas delivered by GLMsingle).
- Registration / spatial changes – It is typical to register fMRI data to an atlas space, or to a subject's anatomy, and/or to correct for head displacements within and across runs from a subject. These types of spatial changes are all fine, and do not fundamentally interact with the issues dealt with by GLMsingle.
- Grand intensity scaling – Some pipelines may scale a given dataset to help normalize units across subjects and/or sessions. This is likely fine, as long as it is a scaling (and not an additive offset). (Scaling preserves percent signal change, and preserves most of the important properties of a dataset.) However, we would have some worry if the scaling were applied differently for different voxels/brain regions and/or differently for different runs within a session, as such aggressive normalization may corrupt signals in undesirable ways.

3.5 Things to watch out for when applying GLMsingle

3.5.1 Edge / head-motion effects

During motion correction, parts of your volume that move outside of the imaged field-of-view will no longer have any valid data. Typical approaches are to "zero out" such areas. These areas are typically on the edge slices of your acquisition. You will want to make sure that this is done carefully and that, for example, a voxel doesn't sometimes have valid data and sometimes does not have valid data. The safest approach is to completely zero out the data for a voxel that does not have full data for all of the runs that you are analyzing with GLMsingle.

3.5.2 Data from multiple scan sessions

BOLD response amplitudes can change substantially across scan sessions (days) for a single individual. Thus, if you are applying GLMsingle to data concatenated across days, this may pose some problems (since the amplitude changes will appear to be "noise" to GLMsingle and may confuse it). One approach is to use the `sessionindicator` input (see below). Another approach is to just apply GLMsingle separately to each scan session. Note that this approach is effective, but does push the issue down the line: you will eventually have to decide whether and how you will want to normalize betas from different scan sessions.

3.5.3 Filenames

If you use the input option `wantlibrary=0` to use a canonical HRF (instead of the library of HRFs), the filename that is written is still "`FITHRF`" (even though the results reflect what the user specified). The reason is just due to the internal code architecture (fitting with a library consisting of one HRF is still treated as "`FITHRF`").

3.6 Tips on usage

3.6.1 My experiment design is not quite synchronized with my fMRI data.

GLMsingle requires the design matrix and the fMRI data to be synchronized at some level of granularity. For example, if the design matrix is specified at 1-s increments, the fMRI data need to also be prepared at 1-s increments. Sometimes, users have experiments where events do not occur at exactly the time points of the fMRI data. In order to accommodate these scenarios, one can perform some upsampling/resampling of the fMRI data and/or the design matrix in order to get the two objects to match. (You might find https://github.com/Charestlab/pyslicetime/blob/master/slicetime/tseriesinterp.py and/or https://github.com/cvnlab/knkutils/blob/master/timeseries/tseriesinterp.m useful for changing the sampling rate of fMRI time series data.)

One drawback is the increase in memory and disk space if upsampling is performed. However, keep in mind that one could just upsample to a moderate level (e.g. 1 s) and then perform a little bit of "rounding the design matrix to the nearest TR/volume". Rounding might introduce fairly negligible levels of inaccuracy given the sluggishness of the HRF.

3.6.2 How do I deal with experimental conditions with different durations?

GLMsingle is designed to estimate betas (response amplitudes) for trials that are expected to share similar timecourse shapes. For example, if an experiment has brief events (e.g. 2-s long), GLMsingle convolves an HRF with a 2-s long square wave to generate the expected timecourse shape, and the idea is to estimate betas that modulate the amplitude of this timecourse shape for the different experimental conditions. If an experiment has long events (e.g. 16-s long blocks), GLMsingle will similarly convolve an HRF with a 16-s long square wave to generate the expected timecourse shape. The timecourse shapes in the two situations are very different.

A major challenge is how to interpret betas when they reflect amplitudes of timecourses reflecting different durations. This is a tricky general problem, and GLMsingle is not particularly suited to resolving that situation.

Note that within a range, the expected timecourse shape undergoes fairly modest changes. For example, the timecourse shape resulting with convolution of a fixed HRF with a 1-s square wave is quite similar to the result of convolution of that fixed HRF with a 2-s square wave. Certainly, the amplitude is very different (as expected); but the shape is fairly similar. So, one approach is to code events generically as, say, 1.5-s in duration, and this will allow the different activity induced by the 1-s and 2-s events to show up in the estimated response amplitude (betas).

Another potential idea is to code a long event as a string of successive short events. For example, suppose that you have some trials that are 2-s in duration, and that occasionally you have a trial that is 6-s in duration. If you are willing to believe that the 6-s duration event is homogeneous in its neural activity over time, you could code this event as a series of 3 2-s events, and label each of these events as reflecting the same condition. In this way, you still conform to the idea that the timecourse of the hemodynamic response is fixed and that you expect the timecourse shape to summate over time. Once you get the individual single-trial betas out, you can simply, for example, average over the 3 individual betas, if you like.

3.6.3 Can I exert control over the noise pool voxels?

The default behavior is to automatically select noise pool voxels based on passing a simple signal intensity threshold (i.e. a simple mask that ignores out-of-brain voxels) and on having very low amounts of BOLD variance related to the experiment (i.e. a "R2" threshold). If you want to specifically control the voxels that are used for the noise pool, you can achieve this by setting `brainthresh` to [99 0] (which allows all voxels to pass the intensity threshold), `brainR2` to 100 (which allows potentially all voxels in) and then setting `brainexclude` to be all voxels that you do NOT want in the noise pool.

3.6.4 What's the deal with `sessionindicator`?

If your experiment wants to analyze/combine data from multiple distinct scan sessions (multiple scanner visits), there is the possibility that after basic pre-processing, there may be substantial abrupt differences in percent BOLD signal change across responses observed on different days. The `sessionindicator` option allows you to tell the code how runs are grouped, and it will internally use this information to z-score responses within sessions in order to better estimate cross-validation performance. This normalization is just done internally (under the hood) and does not propagate to the final outputs.

3.6.5 Why do the betas look crazy outside the brain?

GLMsingle's approach is to estimate percent signal change (PSC) by dividing estimated response amplitudes by the mean signal intensity at each given voxel. In voxels that have very little MR signal (e.g. voxels that are outside the brain), the PSC values might blow up. This is fine and not a reason for concern, as you should simply ignore the data from voxels outside of the brain. Alternatively, if you want to apply a brain mask to your data prior to GLMsingle, that would be fine too.

Likewise, given GLMsingle's algorithmic procedures, it will attempt to identify the optimal HRF even for voxels outside of the brain. This is of course somewhat nonsensical if there is no real signal to begin with. The user can/should simply ignore HRF estimates for these voxels.

3.6.6 How do I interpret the fractional ridge regression values?

In cases where there are strong BOLD responses from the experiment, you should notice that some voxels in the brain have large fractions (i.e. fractions near 1). This indicates that very little regularization is recommended for these voxels. However, if signal-to-noise ratio is weak, the optimal fractions might be near 0. By default, the smallest fraction that is evaluated is 0.05. Hence, it might be the case that most (or all) voxels might have the optimal fraction selected to be 0.05. The interpretation of this, if it occurs, is that heavy regularization of single-trial betas is necessary to improve generalization performance. This does indicate that signals appear to be weak; however, it does not necessarily indicate that there is a problem with the data or analysis per se.

Furthermore, if you inspect the typeD_R2 and the FRACvalue outputs from GLMsingle, we have noticed that these metrics tend accentuate high-SNR voxels and therefore make it somewhat difficult to distinguish low-SNR voxels from surrounding noise voxels. Bear in mind that even if a voxel doesn't "show up" when inspecting these metrics, the single-trial betas that are estimated for the voxel can still have meaningful signals contained within them. We suggest that ultimate determination of whether voxels contain signals should be guided by the user's analyses of the single-trial betas that are provided by GLMsingle.

3.7 Designing design matrices

3.7.1 How should I design my design matrix?

For some experiments, thinking about how to setup the design matrix is a major challenge that deserves careful thought. For example, consider an experiment where there are occasional one-back events, and these events are actually not of interest. Moreover, suppose you do not want to assume that the brain response to these one-back events are somehow similar to other trials in the experiment. Then, what you could do is to code these one-back events as unique conditions that have only one presentation. For example, if there are 30 one-back events, you could just add 30 new columns to your design matrix and indicate the onset of each one-back event in one of the columns. Then, after you obtain betas from GLMsingle, you can either just ignore all of the betas associated with those columns, or use them for some purpose!

Note that it is okay if your final design matrices have one or more blank columns. For example, perhaps the trials associated with a given condition do not occur in a given run, but rather occur in other runs.

3.7.2 Do I need blanks?

In general, you should definitely include blanks or dead time in your experiment. And you should not explicitly code those periods in the design matrix. The reason is that GLMsingle uses a set of polynomials per run to model the baseline signal, and the baseline signal is estimated based on sections of your experiment that are not explicitly coded as experimental events. Note that the interpretation of the single-trial beta amplitudes is that they are evoked responses in the BOLD signal above and beyond whatever the baseline signal is (and this baseline signal may drift to some extent up and down over the course of the run).

If you try to use a design matrix where "everything" is coded, this is problematic because of the inability to estimate the baseline signal level. In these cases, consider omitting the coding of some of the experimental events (e.g. "fixation periods"); in doing so, the idea is that you are trying to estimate changes in the BOLD response **relative** to what the BOLD signal level is during these omitted periods.

3.7.3 What is the minimum trial separation?

Since the BOLD response is very sluggish, the response to two successive trials can overlap a lot. And, the more closely spaced your trials, the larger the overlap. It is an empirical question as to how effectively GLMsingle can separate the response amplitudes to closely spaced trials. Clearly, the closer the spacing, the harder it is to accurately estimate responses from nearby trials; however, the counteracting factor here is that with more closely spaced trials, a larger number of trials can be conducted (which counteracts the loss of power).

We do not yet know what the "minimum trial separation" is. Good results were obtained in the Natural Scenes Dataset with a trial separation of 4 s — in that experiment, an image was shown for 3 s, and then there was a 1 s gap before the next image. Note that the gap here is not very relevant from GLMsingle's point of view. It would be perfectly fine to have, e.g., continuous "mini blocks" that last for a 4-s trial and then immediately move on to the next 4-s trial.

We would guess that going faster than 4 s would be risky. Perhaps 3 s would still deliver reasonable results, but faster than that would seem risky (but is an open question). Keep in mind that in GLMsingle, you code the onsets of trials, so what we are talking about here is the separation between the onsets of successive trials. Also, keep in mind that a separate issue is the `stimdur` input, which controls the duration of the expected hemodynamic response to a given trial. For example, in a situation where successive trials are spaced 6 seconds apart, if the `stimdur` input is 6 seconds, this has different collinearity properties compared to the situation where the `stimdur` input is 1 second.

3.7.4 What about trial subcomponents?

In some cognitive experiments, there are multiple stages to a trial. For example, one might get a cue, preparatory period, a stimulus presentation, another cue to indicate to make a response, and/or an explicit motor (button press) period.

It is up to the experimenter how to design the modeling approach. From GLMsingle's point of view, it does not distinguish (nor care about) these subcomponents. One approach is to code each subcomponent that you wish to model as a distinct condition. For example, you could code the cue for condition A as "condition A1" and the stimulus for condition A as "condition A2". The theory and interpretation is that you are attempting to estimate a separate response amplitude associated with the cue (and its associated brain activity) and the stimulus (and its associated brain activity). As usual, you will have to make some choices as to whether to treat different presentations of a given type of trial as counting as "repeats" or not.

Alternatively, it is totally valid to treat all subcomponents of a trial as contributing to a single response amplitude. From a statistical point of view, this is an easier and more straightforward approach. Ultimately, it depends on your analysis approach and what types of information you are hoping to extract from the data.

3.8 Additional questions/discussion

3.8.1 What does GLMsingle think a "signal" is?

GLMsingle treats responses evoked by an experimental condition as a signal if they tend to replicate across repeated trials of that condition. If the evoked response is zero, then clearly there is no signal. If the evoked response has some variability across trials, that is okay. The general goal is to attempt to analyze a set of data in order to improve the replicability of the condition responses.

3.8.2 What are the metrics of quality that are relevant here?

The philosophy behind GLMsingle lies primarily in cross-validation. Internal to the algorithm is the use of cross-validation to determine the two main "hyperparameters" that are used — one is the number of nuisance regressors (which are derived via PCA on a noise pool of voxels) to add into the GLM model, and the other is the amount of ridge regularization to be applied to each voxel. The idea is that the hyperparameters that best generalize to the unregularized single-trial betas in some left-out data are the appropriate parameters to use.

More generally, the quality metric being invoked here is "reliability" or "reproducibility". Typically, in an experiment you have experimental conditions that are presented multiple times. The assumption is that the signal induced by a given condition should reproduce across different trials.

The concept of noise ceiling (described at length in the NSD data paper) is essentially a sophisticated method to quantify reliability. A simpler metric that can assess reliability is simply taking the responses obtained for a voxel and checking its reliability across different trials. For example, you could split the data into two halves and correlate the signal estimated on one half with the signal estimated on the other half.

3.8.3 I noticed that GLMsingle involves some internal cross-validation. Is this a problem for decoding-style analyses where we want to divide the data into a training set and a test set?

It is true that GLMsingle makes use of all of the data that it is presented with: for example, if you give it 10 runs of time-series data, its outputs are dependent (in complex ways) on all of the data. However, we don't think that this poses any major "circularity"-type problems. All that the algorithm knows about are the onsets of your experimental conditions and some specification of when you think that conditions repeat, insofar that you expect some component of the response to be reproducible across trials. The repeats are used (in leave-one-run-out cross-validation) to determine the setting of the hyperparameters. GLMsingle has no access to your scientific hypotheses, and it is hard to see how it could bias the single-trial beta estimates in favor of one hypothesis over another. There is one exception, however. If the primary outcome you are trying to demonstrate is that responses to the same condition are reliable across runs, then that is, in a sense, exactly what the algorithm is trying to achieve — so in that scenario, you might not want to give all of the data to GLMsingle. Instead, you could divide your data into two halves (for example), independently give each half to GLMsingle, and then test your hypothesis that indeed responses are reliable across the two halves.

3.8.4 How does R2 start becoming meaningful in the full FITHRF_GLMDENOISE_RR version?

For a single-trial design matrix, note that in a sense the predictors are extremely flexible and are happy to capture essentially all or almost all of the variance in the time-series data from a voxel, even if that voxel contains no actual signal. Thus, for `ASSUMEHRF` or `FITHRF` or `FITHRF_GLMDENOISE` type models, the R2 values from these models are more or less meaningless (everything looks "good"). However, the RR technique (as a direct consequence of the fact that it will shrink betas to 0 in accordance to the cross-validated generalizability of the single-trial beta estimates) will essentially leave unperturbed the good voxels that have good SNR and aggressively shrink the bad voxels with little or no SNR. As a consequence, the variance explained by the beta estimates that are produced by ridge regression will be directly related to the "goodness" of the voxel as determined by the cross-validation procedure. Thus, the ridge regression results will have high R2 values for the voxels that seem to have reproducible beta estimates across runs. (One note: The R2 values reflect the explanatory power of the model with the shrunken beta weights. However, by default we apply a post-hoc scaling and offset to the (potentially shrunken) betas from each voxel to match the overall mean of the unregularized betas. Thus, there is a minor mismatch, in that the R2 does not quite correspond to the post-hoc scaled/offset betas.)

Note that it is for these reasons that we write a figure inspection of the typeD model R2 values (typeD_R2.png), which includes RR. The R2 values from the other models are not very informative.

3.8.5 Why does HRF selection improve beta estimates?

The extent to which using a better HRF improves GLM outcomes (like beta estimates) depends on how different the shape of the timecourse of the chosen HRF is from some alternative default HRF. If the shape is only slightly different, beta estimates will only slightly change. If the shape is radically different, the beta estimates will change substantially.

In the brain, hemodynamic timecourses can vary due to variations in vasculature distributions (e.g. close to a vein, far from a vein). If the imaging resolution is high, these variations can be substantial. How much impact timecourse variations have can also depend on the experiment. In general, block designs tend to create experimental predictors that are less affected by HRF variations than event-related designs. Thus, for event-related designs especially, getting the HRF right becomes more critical for achieving accurate beta estimates.

Of course, one general consideration here is that the ability to achieve benefits from HRF selection does depend on signal to noise ratio and the amount of data available. If signal to noise is very weak, it may be practically impossible to accurately estimate HRFs. In such cases, the user can consider fixing the HRF to a canonical HRF, or perhaps to a subject-specific HRF (for example, as estimated through the diagnostic FIR model).

3.8.6 Why does ridge regression improve beta estimates?

Ridge regression imposes a shrinkage prior on regression weights, with the exact amount of regularization controlled by the hyperparameter. In regression problems where there are correlations across the predictors, ordinary least-squares estimates of regression weights are unbiased but can suffer from high variance (i.e. affected by noise). By imposing some amount of shrinkage, the ability of the estimated regression weights to be more generalizable to unseen data can be improved. However, these are general statistical concepts. In the specific case of fMRI designs involving closely spaced trials, there are high levels of positive correlations between temporally adjacent trials. In a sense, you can think of this situation as there being limited amounts of data (statistical power) to disentangle the responses due to adjacent trials. Ordinary least squares tries to estimate these responses but will generally lead to noisy beta estimates with large magnitudes (either positive and/or negative). Ridge regression allows the estimation to impose a prior that effectively downweights the noisy and limited amounts of data that inform the distinction between adjacent trials; we can view it as imposing some amount of "temporal smoothing" (pushing the beta weights from adjacent trials to be more similar in magnitude) with the goal of attaining an overall solution that has better out-of-sample performance. Also, in GLMsingle, note that a different shrinkage hyperparameter is selected for each voxel: this is important since voxels vary substantially in their signal-to-noise ratios.

3.8.7 What are the units of the single-trial beta weights, and are there any interpretation issues?

The default behavior of GLMsingle is to return beta weights in units of percent signal change (by dividing by the mean signal intensity observed at each voxel and multiplying by 100). However, there can be tricky issues having to do with the selection of the HRF. If the HRF used in a GLM is incorrect or different from the underlying HRF, there can easily be gross "gain" or "scale" differences in the obtained percent signal change units of the betas. One of the things that GLMsingle attempts to do for you is to estimate a more proper HRF timecourse (as opposed to assuming a canonical HRF). It is the case that depending on how you use GLMsingle, you may encounter some scale issues. For example, if you use GLMsingle to analyze data from one scan session and then separately analyze data from a second scan session, there will likely be some differences across the two sets of results. First, there could be real (physiological) changes in the percent signal change across different days. Second, each set of results may have a different HRF identified and used for a given voxel; thus, the percent signal change units in the betas for a given voxel will have a different ultimate meaning for the two sets of results. It is an open question how to "best" normalize or handle the betas from different scan sessions to more accurately get at the underlying neural activity. However, a quick and dirty heuristic is to normalize the betas, e.g., by z-scoring, or scaling, depending on your overall scientific goals.

3.8.8 If the HRF changes from voxel to voxel (or area to area), doesn't that pose some interpretation difficulties or confounding issues?

This is an important issue. In essence, what's at stake is the interpretation of the absolute magnitudes of beta weights that are derived from a GLM analysis. Typically, the amplitude of an evoked hemodynamic response is expressed in terms of percent signal change (PSC) (e.g. by dividing the amplitude increase by some baseline signal level and then multiplying by 100). Now, if all voxels/areas/subjects shared exactly the same hemodynamic timecourse shape (i.e. HRF) and we used this HRF in the analysis, then things would be easy. However, there are real differences in timecourse shape across voxels/areas/subjects. (And timecourses can actually also change across scan sessions for the same voxel(s), due to physiology changes.) Thus, an approach that just assumes a single fixed HRF has the advantage of being easy to think about, but already will produce biases in PSC magnitudes. (For example, if you just change the single fixed HRF used in an analysis, you will likely see that some voxel's magnitudes go up and other voxels' magnitudes go down.) Now, consider the approach of tailoring the HRF used in the analysis for individual voxels. Certainly, the interpretation has to follow carefully – the set of betas we observe from a given voxel needs to be interpreted as the amplitude of responses that appear to be present in the data *assuming* the HRF selected for that voxel. Note that in GLMsingle, the library of HRFs are normalized such that the peak HRF value is 1 for each of the HRFs, so you can conveniently think of the betas that result for a given HRF as literally the amplitude. Nonetheless, it remains an open question how much we can actually interpret differences in BOLD amplitudes across voxels/areas. For example, if one region seems to have higher PSC than another region (even within the same subject), this doesn't necessarily mean that the underlying local neural activity is actually higher in the first region. Finally, we note that one approach to "get rid" of these types of amplitude issues is to normalize (e.g. z-score) the responses you observe from a voxel (or region) over the course of a scan session. Of course, one should think carefully about the implications of such an approach for downstream analyses…

3.8.9 Let's talk about the HRF library. How do I know if the library is appropriate for my data?

The HRF library was based on a large survey of "activated" voxels in the visual-memory experiment conducted as part of the Natural Scenes Dataset (NSD). The voxels tended to be in the "back of the brain" but we did not restrict this to be the case (hence, some frontal voxels contributed too). Results were combined across 8 subjects and we attempted to generate a library that is representative of all 8 subjects.

While we think the library should work well as a generic library, certainly it could be valuable to perform detailed and further assessments of this issue. In fact, one could perform finite impulse response (FIR) modeling and derive a new

library of HRFs for a specific set of data using the procedures detailed in the NSD paper, and then hook that library into GLMsingle.

3.8.10 Why isn't the HRF selection cross-validated?

GLMdenoise has a natural "unregularized" state: do not include any derived nuisance regressors as they may introduce overfitting. Ridge regression also has a natural "unregularized" state: perform ordinary least squares estimates and do not introduce any shrinkage bias. The selection of the HRF, however, is a bit different. First, we always need some HRF (it is not as if we can have a model without some HRF timecourse). Second, we lack a baseline HRF, as the canonical HRF used in GLMsingle for the ON-OFF model is really just a very approximate HRF. The different candidates in the HRF library do not really count as different levels of regularization (for which cross-validation might be applied). Rather, each HRF candidate is itself a fully valid HRF choice. In the current approach, we are fully accepting of the idea of chooosing the candidate that works the best (i.e. maximizes the fit to the data). (However, it is certainly true that the statistical approach could be radically altered in such a way that there is some deliberate bias to choose a modal HRF and to only deviate from the modal HRF if the data allow it. This could be a workable approach, but comes with its own set of challenges, such as computational time and added complexity.)

3.8.11 What is the interpretation of the scale and offset?

By default, after the application of ridge regression, GLMsingle applies a post-hoc scale and offset to the single-trial betas obtained for a given voxel to best match what is obtained in the unregularized case. The reason for this is that due to shrinkage, the betas for a voxel are generally "shrunken" down close to 0, and therefore have some bias to be small in size. The theory is that we can undo some of the bias by applying a simple scale and offset to the beta weights. For many post-hoc uses of the beta estimates, a scale and offset is probably not a big deal, but certainly one should think hard about whether or not this matters to the analyses you are trying to do. One can always omit the scale and offset (via appropriate setting of input option `wantautoscale`) and/or avoid ridge regression altogether (via `wantfracridge`).

3.8.12 In the NSD data paper, in the calculation of the noise ceiling, the beta weights are z-scored. What's going on there?

The NSD experiment involves aggregating responses across many distinct scan sessions. Z-scoring is proposed as a potential (simple) method to reduce instabilities that exist across different scan sessions. (Obviously, z-scoring can throw away relevant information, so one should be careful in that regard.) Omitting the z-scoring is perfectly fine as an alternative approach, and one can analyze a set of data accordingly.

3.8.13 If some of my conditions happen only once, and other conditions have multiple repeats, does that somehow make the singleton conditions different or strange?

No, there shouldn't be any major worry here. The analysis components, generally speaking, treat each single trial equally. The repeats are used essentially just to determine the hyperparameters (i.e. number of nuisance regressors, amount of ridge regression shrinkage per voxel).

3.9 Basic checklist for calling GLMsingle correctly

3.9.1 Data

- The data should be your fMRI data (after minimal pre-processing).
- Are the voxel dimensions correct (e.g. XYZ dimensions)?
- Are the number of runs correct?
- Are the number of volumes in each run correct?
- Do you know your voxel size?
- Do you know what the TR associated with your prepared fMRI data is? Is it correct?
- Can you confirm that the fMRI volumes are sane? If you plot a slice from an example volume, does it look like a brain? If you compare a volume from the beginning of a run (or session) to a volume from the end of a run (and/or session), does it look quite similar (and stable)?
- Look at the time-series data for an example voxel (just choose one). Does it look like a reasonable time-series? Are the units reasonable (e.g. arbitrary raw scanner units fluctuating around 1000 or 500 or something like that?)?
- Can you confirm that if you, say, watch a movie of your brain volumes over time, that the brain imaging is stable and that there are no massive transients or artifacts?
- Are all of your runs consistently prepared and have the same format (aside from possible differences in the number of volumes in different runs)?

3.9.2 Design

- The design specifies how you think about the structure of your experiment.
- Do you have a strategy for how you are trying to model your data?
- Have you thought about the input?
- Did you code your trial onsets correctly? Double-check that each column of your design matrix has onsets matched to your fMRI volumes.
- Consider performing a careful visual inspection of the design matrix that is fed to GLMsingle. Does it match your expectation with respect to the timing of your experiment?
- Are you sure the coding of the design matrix makes sense for your experiment?
- Did you make sure to NOT code blank/null trials?
- Did you make sure that the coding of the design matrix is correct in all of your runs?

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

g
glmsingle.glmsingle, 9

MATLAB MODULE INDEX

m

matlab, 1

INDEX

F

fit() (*glmsingle.glmsingle.GLM_single method*), 9

G

GLM_single (*class in glmsingle.glmsingle*), 9
GLMestimatesingletrial() (*in module matlab*), 1
glmsingle.glmsingle
    module, 9

M

matlab (*module*), 1
module
    glmsingle.glmsingle, 9