GLM Denoise

- Introduction
  - Background
  - Getting started
- Run GLMdenoise
  - Define design matrix
    - From mrVista dataTYPES
    - From parfiles
  - Load preprocessed data
  - Define HRF
  - Run it!
- Check results
- Helper code for analyses
  - Make denoised nifti runs
  - Make parameter maps
  - Compute linear contrast
  - Get ROI data

Introduction

Reference paper: Kay et al (2013), Frontiers in Neuroscience

Documentation: GLMdenoise

Background

GLMdenoise is a variant of the standard GLM that uses PCA-derived noise regressors to improve model fit and SNR. The technique is extensively discussed in Kay et al (2013), with examples and FAQ on Kendrick's personal website. In brief, the technique identifies two populations of voxels: "signal" voxels that are driven by the stimulus and "noise" voxels that are not. Noise voxels are defined as the subset of voxels with cross-validated R^2 < 0 (i.e. the model doesn't explain any variance, indicate variance in the signal from these voxels is not driven by the stimulus). Principle component (PC) analysis is done on these noise voxels, identifying 20 noise PCs. These PCs are added to the GLM as noise regressors, and the cross-validated R^2 computed. The "optimal" number of PCs to include is determined as the minimum number of PCs required to improve the R^2 within 95% of the maximum possible. This conservative number of PCs is to minimize risk of overfitting the model. The final model is fit to the data, bootstrapping across the runs. The median bootstrap is taken as the beta estimate and half the 68% CI as the standard error.

The steps for running GLMdenoise mostly consists of organizing the data

1. Define the stimulus (design) matrix, either directly from parfiles or from the mrVista variable dataTYPES.
2. Load the preprocessed functional data
3. [Optional] Define the canonical hrf
4. Run GLMdenoise
5. [Optional] Pass the denoised data niftis into mrVista so that the gui can be used for further analyses

Getting started

First, download GLMdenoise code from Kendrick's personal website or from the GitHub repository (preferred). To clone the git repository, open a terminal:

```
# Navigate to git repository
cd ~/matlab/git

# Download GLMdenoise repository
git clone git://github.com/kendrickkay/GLMdenoise

# Also download KNK utils if haven't previously
git clone git://github.com/kendrickkay/knkutils
```

Second, in matlab, run example1.m to check everything is set up correctly and to familiarize self with code. The code itself is extensively documented, as is the outputs of example1.
Run GLMdenoise

Example scripts running GLMdenoise, making design matrix

Navigate to session

Navigate to the session directory and open a a hidden Inplane view.

```
sessDir = '/Volumes/server/Projects/Gamma_BOLD/wl_subj004';
cd(sessDir)

% Open hidden inplane in order to define global variables
vw = initHiddenInplane;
vw = viewSet(vw, 'cur dt', 'Original');
```

Define design matrix

Create a design matrix for each run of the experiment. The size of each design matrix is #TRs x #Conditions, and has 0s in all cells except for onset of new condition. Each design matrix is stored in a cell array of size 1 x #Runs in experiment. This code creates a figure displaying the design matrix; check these images against your parfiles to make sure everything is correct. This is critical!

From mrVista dataTYPEs

If the standard GLM, as implemented in mrVista GUI, has already been fitted, get the design matrix from the GUI.

```
```
% Define experiment length
nScans = 8; % number of scans in gamma bold experiment
nConds = 9; % number of conditions excluding fixation
nFrames = 148; % number of frames (TRs, volumes) per scan

% make design matrix
design = cell(1, nScans);
figure('Name', 'Design matrices', 'Color', [1 1 1], 'Position', [212 189 1021 1157]);

% for every scan in the experiment, create a design matrix
for scan = 1:nScans

    % get parfile params
    trials = er_concatParfiles(vw, scan);

    % get event related (glm) params
    params = er_getParams(vw, scan);

    % set HRF to 0 since it will be iteratively derived in denoise code.
    params.glmHRF = 0;
    params.glmHRF_params = [];

    % create matrix of zeros with dimensions #TRs x nConditions, so 148 x 9
    X = zeros(nFrames(scan), nConds);

    % fill in design matrix with 1's indicating when each new block starts
    for i = 1:nConds
        ind = trials.onsetFrames(trials.cond==i) + params.onsetDelta;
        ind = ind(ind>0 & ind<nFrames(scan));
        X(ind,i) = 1;
    end

    % Plot design matrix to check against parfiles
    subplot(3, 3, scan);
    imagesc(X);
    colormap(gray);
    colorbar;
    xlabel('Conditions');
    ylabel('Time points');
    title(['scan #' num2str(scan)]);

    % add to design matrix
    design{scan} = X;
end

From parfiles

Alternatively, make the design matrix directly from the parfiles.

Code
function X = makeDesignMatrixFromParfile(parfile, TR)

[onsets, conds, labels] = readParFile(parfile);

% create matrix of zeros with dimensions #TRs x nConditions
nTRs = 131; % 135 total, 4 removed in preprocessing
nConds = 11; % 5 contrast levels x 2 positions + 1 zero contrast
X = zeros(nTRs, nConds);

% fill in design matrix with 1's indicating event onset and type (e.g. event
% at tr = 6 that was condition 4 would be indicated by a 1 in row 6, col 4)
for i = 1:length(onsets)
    cur_tr = onsets(i)/TR;
    cur_cond = conds(i) + 1;
    X(cur_tr, cur_cond) = 1;
end

% check;
figure; imagesc(X);
colormap(gray); colorbar;
xlabel('Conditions');
ylabel('Time points');

Load preprocessed data
This data should have already gone through preprocessing. It should be (1) motion corrected, (2) slice-timing corrected, (3) field map corrected (if applicable), and (4) have initial few volumes chopped.

% Get EPI file names
dataDir = fullfile(sessDir, 'Raw', 'both');
epis = matchfiles(fullfile(dataDir, 'run*.nii.gz'));

% Read in the data
data = cell(1, nScans);
for ii = 1:nScans
    tmp = niftiRead(epis{ii});
    data{ii} = tmp.data;
end

Check the size of the data, making sure it has correct number of voxels (columns 1 and 2), number of slices (column 3), number of TRs (column 4). Check that design matrix has correct number of TRs (column 5) and number of conditions, excepting fixation (column 6).
for ii = 1:nScans
    disp([size(data{ii}) size(design{ii})]);
end

% Should look like:
% 104 x 80 voxels per slice
% 24 slices
% 148 TRs (in data, then in design matrix)
% 9 conditions, excluding fixation
%  104    80    24   148   148     9
%  104    80    24   148   148     9
%  104    80    24   148   148     9
%  104    80    24   148   148     9
%  104    80    24   148   148     9
%  104    80    24   148   148     9
%  104    80    24   148   148     9
%  104    80    24   148   148     9
%  104    80    24   148   148     9

Define HRF

Define the initial seed for the HRF. The HRF will then be iteratively fit in the first part of the GLMdenoise algorithm.

Code

% Define HRF
stimdur = 12; % seconds. Length of block
hrf = getcanonicalhrf(stimdur, tr);

% plot HRF
figure('Name', 'canonical hrf', 'Color', [1 1 1]);
plot(hrf);

title('canonical hrf (spm diff of gammas)');
xlabel('Time from condition onset (s)');
ylabel('Response (arbitrary units)');

Run it!

This process will take 10-15 minutes and is fairly memory intensive.

[results, denoiseddata] = GLMdenoisedata(design, data, stimdur, tr, hrf,[],[],'GLMdenoisefigures');
Check results

GLMdenoise automatically makes a number of figures describing each step in the denoising procedure. These figures will be stored in ‘GLMdenoisefigures.’ Here are a few good figures to look at, as described by KNK’s example code. Open these figures in the finder window, not matlab.

- **Check estimated HRF**: open HRF.png. The initial HRF is the canonical HRF convolved by stimulus (block) length. The canonical HRF is the initial seed used to iteratively fit the estimated HRF, which is then used throughout GLMdenoise for all the scans. Check that estimated HRF doesn’t look crazy.

  ![Initial and estimated HRF](figure)

- **Check noise pool and PC voxels**: open NoisePool.png and PCvoxel.png in same window so can toggle between them. Noise pool shows the voxels that were used to derive noise regressors because they had no signal related to the experiment. Noise voxels are mostly white matter and possibly some voxels far from the surface coil. PC voxels are the signal voxels, which were used to fit the GLM. The noise voxels and PC voxels should be **different** voxel populations, so toggle between the images to make sure. Below, the left image is the noise pool and the right image is the PC voxels. Note that a lot of voxels outside the brain are included in the PC pool. This is okay as they’ll be thresholded out later.

  ![Noise & signal voxels](Brain)
Check cross validation ($R^2$): open PCcrossvalidation00.png and PCcrossvalidationXX.png, where XX is the selected number of PC noise regressors (found from results.pcnnum in matlab). Toggle between the two images. The variance explained should increase.
following denoising, so there should be more and brighter orange in PCcrossvalidationXX.png. The amount of improvement will be reflected in the difference in the two. Below, the image on left is with no noise regressors (PCcrossvalidation00.png), and the image on right is with selected number of regressors (PCcrossvalidationXX.png).

- Variance explained improvement (Parameter map)
Check improvement in variance exp ($R^2$): Open PCscatterXX.png where XX is the selected number of PC noise regressors. The red points indicate voxels used in the GLM denoising, while the green points are excluded voxels. More points above the $x=y$ lines indicates an improvement in variance explained. Voxels further away from $x=y$ show more improvement than those closer to it. Generally want all the points to be above $x=y$ line.

Variance explained improvement (figure)

Check improvement in SNR: open SNRcomparebeforeandafter.png. Interpretation as above, but with SNR instead of $R^2$.

SNR improvement (figure)
Helper code for analyses

Unlike Vistasoft-implemented GLM, the GLMdenoise just gives you back a denoised data set (denoiseddata.mat) and the results of the GLM (denoisedresults.mat). Visualizing the results and doing further analyses requires some work. Below are a few typical things you might want to do with the GLMdenoise outputs. Sample code is included. At some point, might consider making these experiment-specific scripts into functions that can be used across projects.

Make denoised nifti runs

Example script for making denoised nifti runs.

GLMdenoise returns a full denoised data set that can be written to niftis and passed back into mrVista. Note as discussed in GLMdenoise FAQ, the error estimates in the denoised data set will be invalid. Statistical analyses on the denoised data (e.g. t-tests, another GLM) will therefore be invalid as well.

Make parameter maps

Example script for making variance explained, beta, standard error, and SNR maps.

Create a variance explained map, a beta map, a standard error map, and an SNR map. This is useful for visualizing on a mesh, defining ROIs, and getting ROI data. As written, the beta map contains a map for each predictor in each scan in the data type. This works out when you happen to have the same number of scans as betas. Otherwise, either (1) create a GLM maps datatype with as many scans as predictors, or (2) save individual maps for each beta.

Compute linear contrast
**Example function for computing linear contrasts**

Compare betas for two sets of predictors (active set, control set) through a linear contrast. This script takes advantage of bootstrapping by computing contrast for each bootstrap, then taking median and 68% CI as the contrast estimate and standard error respectively. The resulting contrast can be saved as a parameter map for visualization.

**Get ROI data**

**Example script for getting ROI data by loading a parameter map**

Getting data (e.g. betas) from just the voxels in an ROI is surprisingly tricky. ROIs are frequently defined in the gray view in anatomical voxels or in the inplane in inplane anatomy voxels, but GLMdenoise is in functional voxels. The voxel indexing for the GLMdenoise outputs is also different from the indexing for ROI voxels. The most straightforward and least prone method for getting ROI data has been to load a map with the desired data in mrVista, and then use the getCurDataROI function. The disadvantage is that it is slow and impractical if you want all the bootstraps for a voxel, not just a single data value. In such cases, it is best to transform the ROI inplane coordinates using `ip2functionalCoords.m`, and then get the data straight from the GLMdenoise outputs.