FMRI Data Quality

Pablo Velasco

Version 1.5 – Jan 24, 2014

Contents

1 Purpose

This tool examines the quality of functional data. It checks the spatial and temporal statistics of the voxel intensity in a series of ROIs defined by the user, and generates an `html` report with the quantitative quality indicators (Signal-to-Noise Ratio, Signal-to-Fluctuation-Noise Ratio and Signal-to-Ghost Ratio; defined below); as well as a spike detection report and plots of the average image intensity time-course for those ROIs. It also includes a function to correct for the spikes detected in the data.

2 Introduction

Temporal stability during fMRI acquisition is of vital importance because the Blood Oxygenation Level Dependent (BOLD) effects of interest are only a few percent in magnitude. To accurately measure such small signal changes, an MR system must have intrinsic image time series fluctuation levels much lower than these expected signal changes [1].

This document describes the data analysis protocol suggested by CBI to check for the quality of functional data. It is designed to be run on functional scans **before** any data processing (before motion correction, smoothing, etc). By running it on **all your data series** you will be able to tell beforehand if your data are not worth being analyzed, saving you precious time.

3 Description of the analysis

The code loops through the repetitions[1], reads the data (it works for both NIfTI and DICOM formats) and calculates the spatial mean and standard deviation —for each repetition— of the signal in a set of regions of interest (ROIs), some of which are in the object, some in the ghost and some in the background. The location of the ROIs can be calculated automatically by another function (also provided with the package), based on the image dimensions and phase encoding (PE) direction, and can be modified manually by the user.

After obtaining the temporal series of the spatial means and standard deviations for each ROI, a series of statistical indicators are computed, which will give the user a quantitative index of the data quality:

- **SNR$_0$**: Signal-to-Noise Ratio, from [6]. It takes as "signal" the average voxel intensity in all the ROIs defined in the object, averaged across time. It takes as "noise" the standard deviation (across space) of the signal in the ROIs defined in the background, and then averaged across time:

$$\text{SNR}_0 = \frac{<(<S_{\text{object}}>_x)>_t}{1.53\times <[std_x(S_{\text{background}})]>_t} \tag{1}$$

  The 1.53 factor is used to correct for the fact that we took the absolute value of a complex image with 0 mean.

---

[1]You can skip some volumes from the beginning of the run, to allow for signal stabilization.

- **SFNR**: Signal-to-Fluctuation-Noise Ratio, from [1, 2]. It takes as "signal" the same as for the SNR_0 above. It takes as "fluctuation noise" the (temporal) standard deviation of the (spatial) mean in the same ROIs, after the slow drift (quadratic) has been removed from the temporal series.

$$\text{SFNR} = \frac{< (< S_{\text{object}} >_x) >_t}{std_t (< S_{\text{object}} >_x)}$$

- **SGR**: Signal-to-Ghost Ratio, from [4]. It takes as "signal" the average voxel intensity from an ROI inside the object, and then averaged across time. It takes as "ghost" the average voxel intensity from an ROI in the ghost, corresponding to the previous ROI in the object (that is, half a field of view away), and then averaged across time:

$$\text{SGR} = \frac{< (< S_{\text{object}-\text{linked}-\text{to}-\text{ghost}} >_x) >_t}{< (< S_{\text{ghost}} >_x) >_t}$$

Then, it plots the signal time-courses for each ROI group, in which you can check visually for the quality of your data. Next, it searches for spikes in the signal intensity in a set of ROIs in the background. Finally, it writes a `html` report file which includes some information about the scan, the figures, the statistical indicators and the spike detection results. If any spikes had been detected during the analysis, this tool saves the repetition numbers and the spiky slices in a file, allowing the user to correct the data with an additional function provided in this tool.

Here we describe in a little more detail each of the functions:

3.1 roiCorners.m

`roiCorners.m` is a function used to automatically generate the default location of a series of ROIs for the data quality check. If only the name of the data file is passed, it uses the default name `rois.txt` to store the ROI locations. If the ROIs file already exists, it doesn't overwrite it and it just displays the location or the ROIs (see Sec. 3.3).

The default locations depend on the the acquisition matrix (*e.g.*, 64 x 64), the number of slices and PE direction (obtained from the data header). The locations are based on the assumption that the object will be in the middle of the field of view and that there is background in all four edges of the image. The ROIs are classified into four sections (object, ghost, background and 'spikes'). These are the default ROIs (see Fig. 1):

- **Object**:
  - 20x20 center of Object (`object_maxroi`: red box in the center of the object in Fig. 1).
  - 10x10 in Object, the region corresponding to 10x10 ROI in Ghost (that is, Ny/2 away from the ghost in the PE direction). (`object_linked_to_ghost`: green box right below `object_maxroi` in Fig. 1).

- **Ghost**:
  - 1st part on the right (or top, depending on the PE direction) of the central slice, centered at 1/4th of the image along the readout dimension (`ghost`: navy box at the right of the image in Fig. 1).
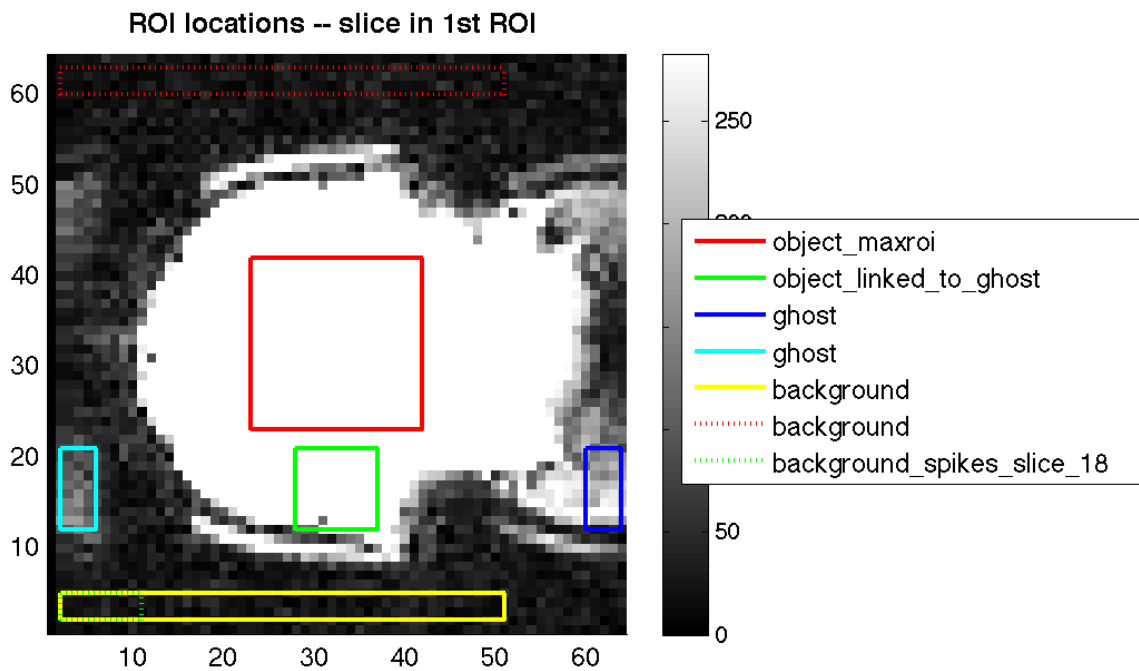
3

Figure 1: Example of the default locations of the ROIs in the central slice from a representative run (the image intensity has been windowed so that the ghost shows up clearly). This type of plot allows the user to check whether the default placement of the ROIs is correct or if you need to move some of them.

- 2nd part on the left (or bottom, depending on the PE direction) of the central slice, centered at 1/4th of the image along the readout dimension (`ghost`: light blue box at the left of the image in Fig. 1).

- **Background**:

  - 2x50 along one of the edges of the image where there is no ghost, in the central slice (`background`: yellow box at the bottom of the image in Fig. 1).

  - 2x50 at the other edge of the image where there is no ghost, in the central slice (`background`: red dotted box at the top of the image in Fig. 1).

- **Spikes**:

  - in every slice, a small (5x10) ROI in the background, to detect spikes (For the central slice:`background_spikes_slice_18`: green dotted box at the bottom left of the image in Fig. 1).

Since **it is very important that you check that the ROIs are in the correct locations**, after the ROIs are saved into a text file, `roiCorners` calls the function `displayROIs` (Section 3.3) to display them in a figure, like Fig. 1.

If you need to modify the default location of any of the ROIs, open the file with a text editor. You will see the different ROIs organized by region, and the entry for each one is preceded by a label and a short description of the ROI, followed by a line with the (*i,j,k*) indices of one of the corners of the ROI, and in, the next line, the coordinates of the opposite corner. Note that the *i,j,k* indices start counting from 1, not from 0.

Here is a sample of the first few lines of a ROI file:

```
=== OBJECT ===

OBJECT_MAXROI: 20x20 center of Object
23,23,18
42,42,18

OBJECT_linked_to_Ghost: 10x10 in Object - region corresponding to 10x10 ROI in Ghost
12,28,18
21,37,18


=== GHOST ===

GHOST: 10x10 in Ghost - 1st part: top of slice
12,60,18
21,64,18

...
```

3.2   importROIs.m

`importROIs.m` is a function to read the ROI corners from a ROI file. It loops through all the lines in the file and stores the information in four structure arrays (`object`, `ghost`, `background` and

5

`spikes`); with each of the elements of the arrays containing the following fields:

- `name`: name of the ROI
- `type`: the type of the ROI: `object`, `ghost`, `background` or `spikes`
- `corner1`: indices (*i,j,k*) of the first ROI corner.
- `corner2`: indices (*i,j,k*) of the second (opposite) ROI corner.
- `nvox`: total number of voxels in this ROI
- `meants`: time-series of the mean voxel intensity across the ROI.
- `stdts`: time-series of the standard deviation of the voxel intensities across the ROI.

Since `importROIs` only reads the information contained in the ROI file, the last two fields (`meants` and `stdts`) are returned empty. They will be used by `checkdataquality.m` (Section 3.4) to store the corresponding ROI statistics for each time-point.

3.3 displayROIs.m

`displayROIs.m` is a function used to display the location of some of the most relevant ROIs with respect to the image.

It first calls `importROIs.m` (Section 3.2) to read the ROIs locations, and then gets the first volume of data in the image file to generate the image. With these, it selects the the slice corresponding to the first corner of the first ROI in the ROIs file. It creates a figure showing that slice as well as colored rectangles showing all the ROIs present in that specific slice. An example of such an image is Fig. 1.

3.4 checkdataquality.m

`checkdataquality.m` is the function which does the statistical computations and generates the images.

As with `roiCorners.m`, the first thing `checkdataquality` does is to read the file header, to obtain the acquisition matrix (*e.g.*, 64 x 64), the number of slices and repetitions, and the PE direction. If the `*-header.txt` file created at extraction from CBI-webdb is present in the same folder as the data file, `checkdataquality` reads the date the data were collected as well as the coil that was used.

Then, `checkdataquality` calls `importROIs.m` to get the location of the ROIs. Once the function knows where the ROIs are, it loops through the repetitions (a.k.a. volumes, measurements or time-points), reading one volume at a time and, for each of the ROIs, it calculates the mean voxel intensity (and the standard deviation) across the ROI. It also calculates the mean voxel intensity of all ROIs in each of the three regions: `object`, `ghost` and `background`. The user can specify how many time-points to exclude from the beginning of the run (the default is zero).

When all the repetitions have been read, it calculates the 3 statistical indicators described in Sec. 3 and plots 3 graphs, with the time courses of the average voxel intensity in the ROIs in the `object`,

`background` and `ghost`, respectively. In them, you can observe the size of the fluctuations, and the intensity in the background, compared to the image intensity.

Finally, `checkdataquality` runs a "*search for spikes*". In it, for each of the ROIs in the `spikes` section of the ROI file, the function does a search for peaks in the intensity. A time point is tagged as a 'peak' when the signal is bigger than 'threshold' standard deviations above the mean (across time). The threshold is set to 10, which seems to be very effective at catching spikes, without giving many false positives. Internally, it saves the spike candidates in a structure called `peaks`, with fields:

- `ts`: timepoints where a spike was detected (for that ROI);
- `nspk`: number of spikes (again, for that ROI);
- `slices`: slices included in that ROI;
- `name`: name of the ROI where those spikes were found.

The spike detection part of `checkdataquality` also plots the time courses of the mean signal for each of the `spikes` ROI. The data would be noisy, but normally without any big peak (for example, see Fig. 2). An example of a time-course with "big peaks" is given in Fig. 3. In the plot you can evaluate visually how big the detected peaks are. If the "spike detection" part picked up any spike candidates, the script will also generate a graph for each of the candidates in which it will show the image of the corresponding slice for the bad timepoint (in the middle) and the adjacent timepoints both before and after the bad timepoint (Fig. 4).
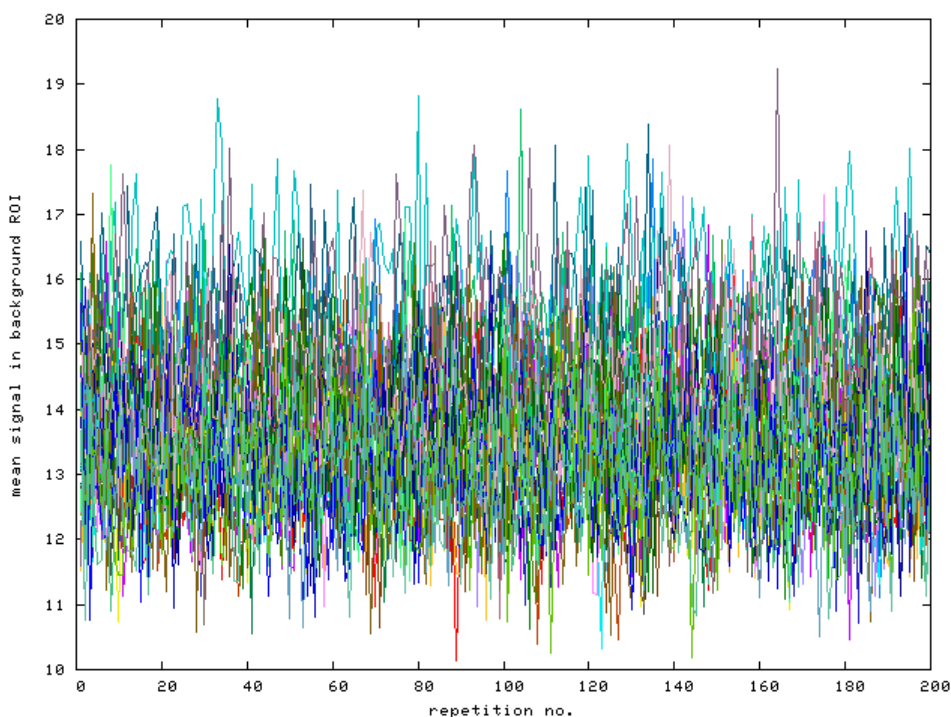
Figure 2: Example of the time-course in the `spikes` ROIs for a normal run. Note how the noise fluctuation size is very constant across time and the baseline is similar for all ROIs.

Background Timeseries for Spike detection
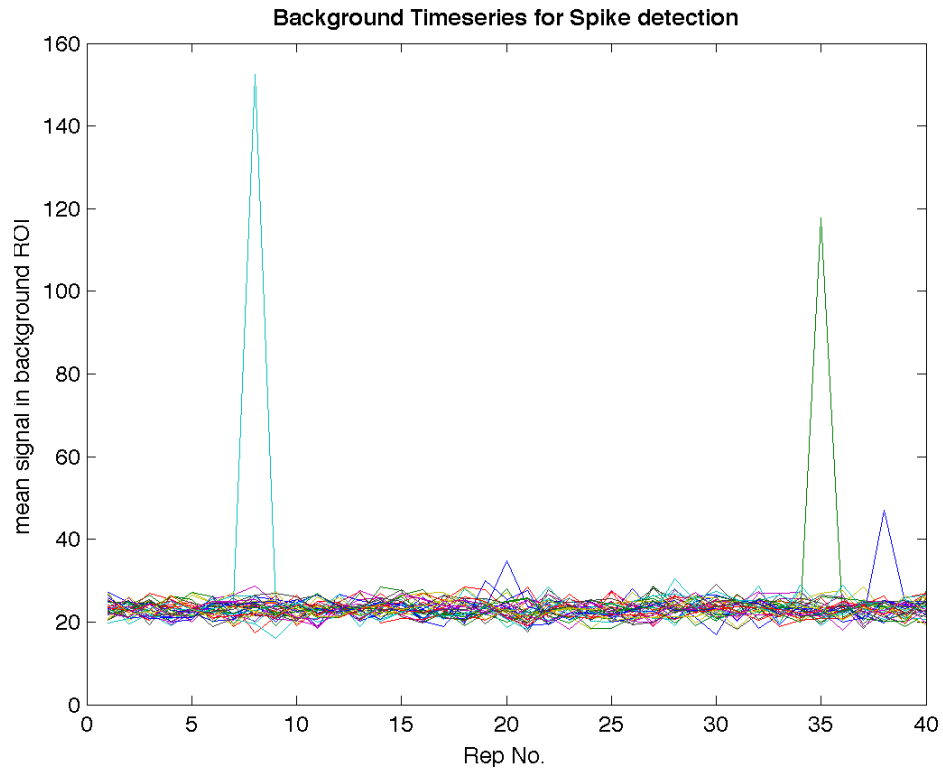
Figure 3: Example of the time-course in the `spikes` ROIs for a "spiky" run. There are three peaks standing up above the baseline level.

Slice: 2 (in "Spikes ROI": BACKGROUND_SPIKES_SLICE_2)

t = 33   t = 34   t = 35   t = 36   t = 37

Figure 4: Images of the slice showing a peak in the intensity at time-point 35 in Fig. 3, and the same slice for the two timepoints before and after. Note the corduroy pattern across the image, typical of spikes.

3.5 dataQReport.m

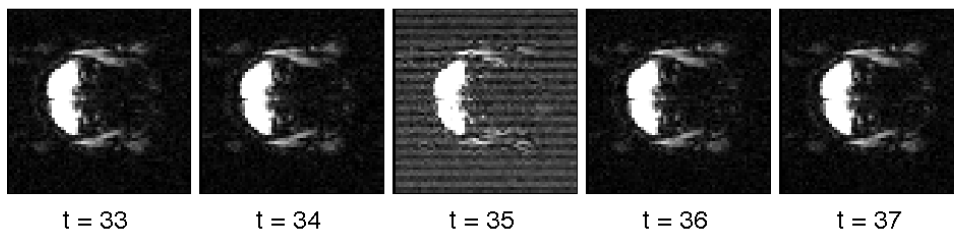`dataQReport.m` is the function that generates an `html` report of the results from `checkdataquality`. The function creates a folder (inside the same folder as the data being checked) called `qReport_datafName`, where `datafName` is the name of the data file being analyzed (with no extension).

The first step is to run `checkdataquality` and to save the images created (as `png` files). Then, `dataQReport` writes out the `html` file, line by line, which includes:

- The coil used to collect the data.

- The date of the study.

- The plot with the ROI locations (see Section 3.3).

- The SNR "indicators".

- The time-series plots for the intensity in the three ROI sections: `object`, `background` and `ghost`.

- The results of the "spike detection": Total number of potential spikes, and the time-point and ROI where they occurred; the time-series plot of the intensity in all the `spikes` ROIs and, in the case any possible spikes are detected, the plot of the slice and time-point where they were detected (and the 2 previous and posterior time-points for that same slice, for comparison).

Finally, `dataQReport` copies the ROI file used to generate the report in the same folder containing the report. The idea is that if the user thinks the CBI staff should take a look at the results, he/she can zip the whole folder that was created and email to the corresponding person. Then, the zip file will contain all the information the CBI staff would need to analyze the data.

`dataQReport.m` returns a structure with the following fields:

- `snr0`

- `sfnr`

- `sgr`

- `sDate`: scan date

- `coil`: coil used for the scan

- `peaks`: structure containing the information about spikes in the data (see 3.4).

3.6 correcting_spikes.m

A new version (v 2.0) of `correcting_spikes.m` has been included with the distribution. This function corrects bad timepoint-slice pairs from a fMRI run, by substituting them with an average of the intensity for that defective slice across a given number of timepoints before and after the bad one.

In this new version, the function has been modified so that it takes as input arguments the output of `dataQReport`. These are the arguments:

- `datafNameIn`: file with the original data.

- `spikesfName`: Matlab file with the structure (generated by `dataQReport`) containing the ROI name, the number of spikes for that ROI and the timepoints at which those spikes were detected (the `*_spikes.mat` file inside the folder where the `html` report was saved).

- `datafNameOut`: file (for NIfTI image) or folder (for DICOM images) where to store the corrected data.

- `adj_tpoints`: (optional) number of timepoints before and after the bad timepoint to be averaged to substitute the bad slice (default value: 2).

`correcting_spikes` only includes in the average the data from the same slice corresponding to adjacent timepoints only if they are themselves free from spikes (i.e., not included in the file `spikesfName`).

4 Discussion

This tool calculate several statistical indicators which address different aspects of the data quality. Each has its relevance, and it is very important to monitor their values across different runs within the same session and across different sessions and subjects. After analyzing several runs, one can get an idea of what values to expect for a given imaging protocol, and get a feeling of what a "bad run" is.

In general, for the case under consideration, which is that of functional imaging, one should pay special attention to the values reported for **SFNR**. There are several contributions to the temporal intensity fluctuations at any given voxel: the functional activity itself, the electronics noise (mostly white noise from the receiver), physiological changes (respiration and heart beat), scanner noise (B0 drift due to heating, ...), etc. Since the goal of checking the data quality is to know how good our data are going to be for the functional analysis, one wants to make sure that the fluctuations not due to functional activity are as small as possible. Therefore, one wants to estimate the size of the temporal noise (temporal fluctuations in the signal) with respect to the average temporal signal using a large number of voxels. For each time point, using the average intensity in a large number of voxels will average the BOLD contributions from different functional parts of the brain, so the variations due to functional activity over time will be significantly washed out, giving only an average functional contribution. Most of the electronics white noise will be averaged out as well, since its contribution is spatially independent. This way, the main source of temporal variation of the average intensity will come from the scanner noise (which is what we want to estimate) and from the physiological activity (which can be recorded and subtracted from each voxel time-course, improving the statistical power of the functional analysis of the data). Therefore, SFNR —which is the indicator tuned to measure this aspect of the noise— is probably the most relevant indicator of the quality of our functional data.

The 1.53 factor in Eq. 1 is based on the assumption that the noise distribution in the background is Rayleigh. This is only valid for data collected using a single coil. If the data were collected using a different number of coils, the factor would be different [3]. But, for purposes of SNR_0 comparisons across runs and across subjects, it doesn't make any difference.

Always check the ROI location: if an ROI happens to be too close to the edge of the head,

10

even a little bit of motion could artificially introduce fluctuations in the ROI signal, decreasing the SNR numbers, but with no real effect in the actual quality of the data for our functional analysis. You should also be careful when using the first and last row or column in a slice, since they can be artificially filled with zeros during image reconstruction, corrupting the statistical calculations. Again, check the image with the ROI locations to see if the intensity is zero or not; or, even better if you are not sure, stay away from the slice edges.

The idea of using the slice specified by the first ROI in the ROIs file for the plot of the ROI locations is that normally that first ROI will correspond to the region of the image the user is most interested in and, therefore, wants to know the placement of the ROIs with respect to the image in that slice. If you want to look at the location of the ROIs in any other slice you can manually edit the ROI file so that the *k* index of the first corner of the first ROI is the slice you want (note: we start counting from 1, not from 0) and then run `displayROIs` again (or run `roiCorners` passing the name of the modified ROI file as a second argument).

The spike detection algorithm uses a robust linear fit to determine the mean and standard deviation of the background noise level. Since Matlab's `robustfit` function requires the Statistics Toolbox, we include our own implementation, `CBIrobustfit`, with the package.

Note that the "spike detection" analysis picks up any surge in the background noise. This can be coming from actual spikes or from other noise source, but in general it means "problems with the data". If the script doesn't print any spike-candidates, one can be sure there are no spikes in the data. If it does print some, the user have to look at the plots of the specific slice/time-point in the data (which will be printed in the report) to see what the problem is. Since the script picked that point, there will be some background irregularities. You have to see how big those irregularities are compared to your background signal and compared to the normal fluctuations in the time-course of the `object` ROIs: it is possible that they are insignificantly small.

For the `correcting_spikes` function: once the data quality report has detected the presence of spikes in the data, the user should go to those slices and see if the image is good enough for the functional analysis. If the data for that slice-timepoint are bad, the best thing to do is to try to exclude them from the analysis. The procedure to do it depends on the type of analysis; as an example, if when working with a General Lineal Model, a possibility is to remove that datapoint from both the data vector and design matrix; or to include an extra column (one per bad datapoint) in the design matrix. This new column will be full of zeros except for a one corresponding to the defective datapoint. Both approaches are equivalent, and will yield a least-square fit in which the defective datapoint is not taken into consideration.

Another possibility is to correct for the defective data point (using `correcting_spikes`), by averaging a certain number of datapoints in front and behind the bad one (for the same slice). If the user decides to follow this option the information being used to fit that timepoint will not contain information from the specific timepoint, but an average from the adjacent timepoints. How good or bad this approximation is, is an empirical question: one would have to compare the results of the analysis with and without averaging.

Ideally, one would want to save the images as some kind of vector graphics. Unfortunately, the only vector graphics supported by Matlab are postscript and `pdf`, which will not show up in an html file.

A note for CBI users extracting their data as "NIfTI BrainVoyager". When selecting this option, a full series of NIfTI `.hdr`-`.img` pairs will be created, as well as the original 4D NIfTI file. Since

this data quality tool doesn't work with 3D NIfTI files, the 4d file should be the data file passed as argument to `dataQReport`.

5 Usage

This is the step-by-step procedure you should follow to check your data:

1. Extract your data the usual way (from our CBI- WebDB).

2. Open Matlab and, for each of your runs:

   (a) Run:

   ```
   roiCorners(datafName,roifName)
   ```

   where `datafName` is the name of the file containing the data, and `roifName` is the name of the file where you want to save your ROIs (by default, it saves the ROIs in a file called `rois.txt`).

   (b) Examine the location of the ROIs. Make sure all the ROIs are where they are supposed to be. Specifically, check that the ROIs that are supposed to be on the background do not include any ghost (that's why the maximum intensity in the image was set so low: to show clearly the ghost).

   (c) If changes need to be made in the location of any ROI, edit `roifName` and save the changes. Then, run again:

   ```
   roiCorners(datafName,roifName)
   ```

   This time the script will just display the ROI locations that you just introduced.

   (d) Run:

   ```
   dataQReport(datafName,roifName)
   ```

   If you want to exclude from the analysis a few time-points at the beginning of the run:

   ```
   dataQReport(datafName,roifName,tExclude)
   ```

   (e) Look at the report generated.

   (f) It is also highly advisable to visually inspect a movie of data. There are many viewers for this, like FSL or OsiriX.

3. If the report looks fine, you can go ahead with your usual data processing stream. If you see something odd in the data, please contact the CBI staff **as soon as possible**. Send an email indicating what the problem was, the accession number and run number, and a zip file of the `qReport` folder.

4. If the report tool finds any spikes, you might want to run:

   ```
   correcting_spikes(datafNameIn,spikesfName,datafNameOut)
   ```

with `datafNameIn` the same data file on which you ran the report, `spikesfName` the name of the `mat` file where `dataQReport` saved the spikes found (the `*_spikes.mat` file inside the folder where the `html` report was saved) and `datafNameOut` the file (for NIfTI files) or the directory (for DICOM files) where the corrected data should be saved.

6 Dependencies

Besides the functions distributed in this package, you will need:

- If your data are DICOM, you need Matlab's Image Processing Toolbox (it is included in both the CNS and the NYU ITS network licenses).

- If your data are NIfTI, you need in your path our CBI NIfTI Matlab libraries (http://cbi.nyu.edu/software/niftimatlab.php).

7 Example data

Together with the code, you can find three different data-sets:

- `normalData.nii`: What you can expect in a normal data-set.

- `ghostyData.nii`: Data showing a relatively high ghost: the `SGR` is very low. However, the ghost intensity doesn't fluctuate too much over time.

- `spikyData.nii`: Data-set which contained a few spikes. You can observe the typical corduroy pattern in the time-points with spikes.

You can copy those data-sets to a directory where you have writing privileges and follow the steps above to check the quality of the different runs. Then, observe the differences between them.

We also provide the html reports generated by running the data quality tool on each of the three example data sets.

8 Where to find the latest version

You can download the latest version of the code and example data from the CBI software page at http://cbi.nyu.edu/software/dataQuality.php

The software is installed in the CBI Image Processing Lab. You will need to add the following to your MATLAB path:

- /CBI/MatlabTools/dataQuality

- /CBI/MatlabTools/niftimatlab

If you have any problems downloading the code or example data, send an email to software@cbi.nyu.edu.

If you have problems running the code or if you have any questions or comments about the tool, send an email to pablo.velasco@nyu.edu.

Appendix A: Spikes

From [5]

Because of the Physics involved in MRI, the data are collected in k-space (or data space), and the MR image is created by a mathematical transformation (equivalent to an inverse Fourier transform) of the data (see Fig. 5, top). If a malfunction of the electronics happens during data acquisitioin, a "spike" may appear in the data. A typical, "clean" spike will last for a very short time, producing a very localized signal increase in the data. When one transforms these data into a MR image, the spike will be transformed into a periodic noise, with a spatial frequency given by the k-value where the spike appeared. A key fact here is that the localized noise in k-space is transformed into spatial noise distributed all over the image space, with a well defined pattern (see Fig. 5, bottom).

Appendix B: Representative values at 3 mm isotropic voxels

Following you can find representative indicator values obtained while scanning at 3 mm. isotropic voxels. Note that the specific values will depend on many things, like where you place your ROIs (especially when using surface coils), reconstruction type (online vs. offline), the length of the run, the bandwidth, the specific task, TR, TE, FA, etc. As a rule of thumb, the "raw" *noise*[2] should increase with increasing bandwidth as $\sqrt{BW}$, and the temporal *noise* should decrease with increasing number of volumes as $\sqrt{N}$.

Table 1: Representative values of data quality indicators for several sessions (separated by horizontal lines). Included are also the number of volumes and the bandwidth used for each run. In the notes, 4Ch: 4-channel occipital array

SNR_0	SFNR	SGR	Nt	BW (Hz/Pixel)	Notes
35.87	437.66	21.71	150	4885	
36.07	378.56	22.03	150	4885	
35.94	336.28	21.59	150	4885	
35.72	422.31	20.42	150	4885	
35.80	360.14	20.38	150	4885	
35.64	304.14	20.98	150	4885	
35.71	256.41	20.92	150	4885	
35.86	389.70	20.29	150	4885	
40.19	353.49	23.53	150	4885	
39.91	356.37	23.32	150	4885	
39.80	369.80	22.87	150	4885	
40.15	361.08	23.08	150	4885	

---

[2]this is only true if the electronics noise dominates

Table 1: (continued)

SNR$_0$	SFNR	SGR	Nt	BW (Hz/Pixel)	Notes
39.86	352.07	22.61	150	4885	
40.25	401.14	23.02	150	4885	
40.12	435.09	23.10	150	4885	
39.98	304.16	23.13	150	4885	
29.63	293.70	21.00	174	4110	
32.38	250.30	17.50	174	4110	
32.52	260.01	17.92	174	4110	
32.50	233.96	17.70	174	4110	
32.62	226.04	17.71	174	4110	
32.52	173.57	17.78	174	4110	
32.56	181.73	17.80	174	4110	
32.69	173.00	17.73	174	4110	
32.77	165.65	17.65	174	4110	
29.82	178.76	18.42	192	4110	
29.62	299.35	18.68	174	4110	
29.34	251.38	18.31	174	4110	5 spikes
29.68	206.22	18.77	174	4110	
29.57	226.23	18.70	174	4110	
29.75	201.03	18.86	174	4110	
29.54	239.31	18.65	174	4110	
29.45	241.72	18.71	192	4110	1 spike
33.25	512.98	24.64	60	4110	
33.38	433.44	24.10	60	4110	
33.93	419.49	24.00	60	4110	
32.35	392.42	24.44	60	4110	
33.74	440.53	24.11	60	4110	
33.51	527.37	24.34	60	4110	
34.57	493.44	24.50	60	4110	
41.64	450.41	21.09	60	4110	
41.66	670.26	21.25	60	4110	
41.68	341.18	21.08	60	4110	
41.81	461.42	20.96	60	4110	
41.59	364.69	20.91	60	4110	
41.58	331.81	20.64	60	4110	
41.60	372.16	20.58	60	4110	
39.78	489.56	21.81	60	4110	
40.03	435.75	21.82	60	4110	
40.27	411.89	21.72	60	4110	1 spike
40.22	346.25	21.30	60	4110	
40.16	366.56	21.49	60	4110	
40.52	339.34	21.14	60	4110	
40.01	380.39	21.12	60	4110	slight motion
42.69	362.61	28.31	60	4110	117 spikes
38.93	402.02	26.09	60	4110	121 spikes
29.94	173.68	18.36	60	4110	282 spikes

Table 1: (continued)

SNR$_0$	SFNR	SGR	Nt	BW (Hz/Pixel)	Notes
26.03	139.84	16.91	60	4110	254 spikes
15.48	40.97	10.06	31	4110	122 spikes
94.49	327.81	14.63	120	4110	4Ch
94.84	272.45	14.92	120	4110	4Ch
95.67	259.96	15.59	120	4110	4Ch
96.57	312.41	15.66	120	4110	4Ch
96.16	384.88	15.72	120	4110	4Ch
95.42	263.48	15.70	120	4110	4Ch
95.63	366.65	15.85	120	4110	4Ch
64.56	250.26	17.16	120	4110	4Ch
64.14	219.03	18.95	120	4110	4Ch
65.71	212.17	18.74	120	4110	4Ch
66.08	166.46	18.15	120	4110	4Ch
65.22	221.77	17.69	120	4110	4Ch
65.33	192.17	17.79	120	4110	4Ch
65.03	209.05	18.15	120	4110	4Ch
65.30	154.13	18.64	120	4110	4Ch
66.98	302.94	22.77	120	4885	4Ch
75.80	305.17	22.94	120	4885	4Ch
50.52	459.14	23.49	120	4885	4Ch, some motion
46.60	333.99	22.29	120	4885	4Ch, some motion
39.74	254.47	22.20	120	4885	4Ch
36.99	360.12	22.40	120	4885	4Ch, some motion
29.97	419.24	22.68	120	4885	4Ch, some motion
27.99	347.31	21.91	120	4885	4Ch
28.51	357.73	20.57	120	4885	4Ch, some motion
38.03	280.76	16.21	320	4110	4Ch, 1 spike
38.01	294.41	16.38	330	4110	4Ch
38.53	292.08	16.51	330	4110	4Ch, 1 spike
39.05	264.91	16.63	330	4110	4Ch, 1 spike
39.11	319.56	16.67	330	4110	4Ch
39.39	268.44	16.34	330	4110	4Ch
39.42	281.92	16.24	330	4110	4Ch
39.64	293.26	16.30	320	4110	4Ch, 5 spikes
53.52	188.92	16.20	290	4110	4Ch
53.80	178.32	16.63	290	4110	4Ch
53.85	181.67	16.61	290	4110	4Ch
53.86	146.83	16.32	290	4110	4Ch
54.15	173.66	16.46	290	4110	4Ch
54.03	172.10	16.44	290	4110	4Ch
54.31	193.44	16.32	290	4110	4Ch
54.37	213.94	16.27	290	4110	4Ch, 3 spikes
53.31	294.21	15.34	320	4110	4Ch
54.47	186.09	16.38	320	4110	4Ch, 1 spike
36.01	413.11	40.92	246	4110	multi-echo

Table 1: (continued)

SNR_0	SFNR	SGR	Nt	BW (Hz/Pixel)	Notes
29.78	336.29	13.40	128	4165	multi-echo
38.01	198.63	23.93	225	4885	1 spike
35.44	463.41	24.54	200	4885	17 spikes
31.42	49.61	21.72	168	4885	30 spikes

References

[1] L Friedman and G H Glover. Report on a multicenter fmri quality assurance protocol. *J Magn Reson Imaging*, 23:827–839, 2006.

[2] Foland L Glover GH. Scanner quality assurance for longitudinal or multicenter fmri studies. In International Society for Magnetic Resonance Imaging, editor, *12th Annual Meeting of the International Society for Magnetic Resonance Imaging (ISMRM)*, 2004.

[3] SO Rice. Mathematical analysis of random noise. *Bell System Technical Journal*, 23:282–332, JUL 1944.

[4] A Simmons, E Moore, and S C Williams. Quality control for functional magnetic resonance imaging using automated data analysis and shewhart charting. *Magn Reson Med*, 41(6):1274–1278, 1999 Jun.

[5] Pablo Velasco. Spike detection/correction. Technical report, New York University, Center for Brain Imaging, April 2006.

[6] R M Weisskoff. Simple measurement of scanner stability for functional nmr imaging of activation in the brain. *Magn Reson Med*, 36:643–645, 1996.
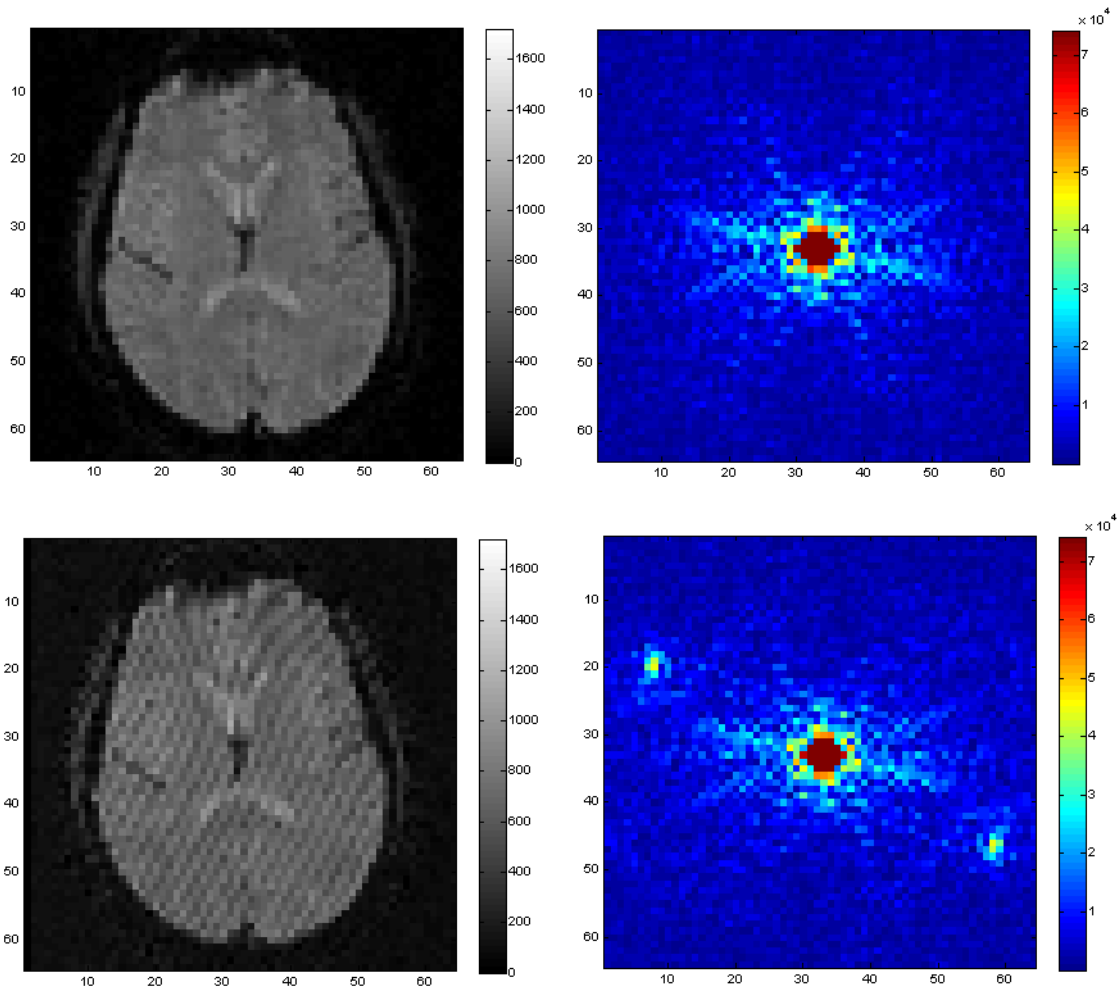
Figure 5: Top: A normal EPI slice (left) together with the corresponding reconstruction (by 2D Fast Fourier Transform - fft2) of what the k-space data would have looked like. One can go back and forth between them by fft2. Bottom: Same, for a "spiky" EPI slice. Note the spike in k-space, with its symmetric copy, which is the origin of the noise in the image.