activeBrain:
Matlab toolbox for the visualization of ECoG activations

Jan Kubánek <kubanekj@union.edu>
Czech Technical University in Prague

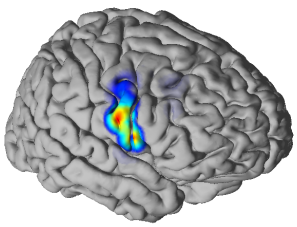
August 2005

The activeBrain Matlab toolbox was designed for internal purposes of the
BCI group at the Wadsworth Institute, Albany, NY

Abstract

The visualization of ECoG activations is implemented by the activeBrain Matlab toolbox. The purpose of he design was to provide a simple tool for displaying ECoG activations. Given a grid of ECoG electrodes, their activation values and a normalized cortex surface, this tool aims to visualize the activations by first interpolating the values at all points of the surface between the electrodes and then plotting these interpolated values in specified pseudocolors. The only values that match the activation values are those located exactly at the given electrode coordinates. The values of the intermediate points rely on the interpolating process that can be specified by using appropriate parameters. This toolbox is also capable of visualizing average activations, given electrode locations and activations for every subject. Because individual brain sizes will always be different, the electrode locations will also differ among subjects. Therefore, activeBrain implements a way to project given electrodes onto the surface of a normalized(by default Talairach) brain. The electrode locations must then be specified in the same coordinate system. If provided with samples of activation values as a time sequence, the tool further allows to capture the activations into an .avi file. A thorough commentary, remarks and examples throughout the package files should facilitate the use of this toolbox. A demo script is provided to demonstrate its capabilities and to allow a better understanding of the way activeBrain is used.

keywords: activeBrain, visualization, ECoG, activation, average, interpolation, projection, Matlab, Talairach



Contents

1 Introduction

Recently, several tools have been implemented to visualize medical data obtained by various medical imaging techniques, such as MRI, fMRI, PET, SPECT. Many of these tools are open source and thus available for research purposes [2, 3, 4, 5]. However, these tools rely on high-resolution data, usually acquired by MRI techniques. Often, a MRI voxel spans a volume of a single cubic millimeter. Provided with high resolution data, these tools can visualize brain activations quite precisely. However, the visualization of electrocorticographic(ECoG) signals(in general, activations) faces the problem of uncertainty: only the signals at implanted electrodes are known. This makes a surface visualization theoretically impossible, since only the values at the implanted electrodes can be plotted with certainty. In this case, one can assume that a signal measured on an electrode is propagated through the cortical tissue and that the signal values in the electrode surroundings will be constant or will be decreasing ("fading" in the documentation) as a function of the euclidian distance.
There is a lack of freely available tools that would aim to visualize ECoG signals and yet there are groups that investigate possible applications of ECoG signals, as it is the example of the brain-computer interface research conducted at Washington University in St. Louis and Wadsworth Center, New York Department of Health, Albany, NY [6]. The activeBrain Matlab toolbox was developed with the view of filling in the gap in current availability of open source ECoG visualization packages.

2 Methods

The `activeBrain` tool performs the following main steps:

- flatten the brain model (`coarserModel, smoothModel, hullModel`)
- project the electrodes on the flattened model (`projectElectrodes`)
- reload the dense brain model (`pial_talairach.mat`)
- compute the electrode contributions to the associated brain's surface areas (`electrodesContributions`)
- display the activations (`activateBrain`)

In order to be able to proceed, the following data must be available:

- brain model (in this demo – `pial_talairach.mat`)
- electrode locations for desired subjects (`DEMOsubj.mat`)
- activation data for these electrodes (`DEMOsubj.mat`)

2.1 Cortex model

The file `pial_talairach.mat` contains a normalized cortical model that is the only one currently supported by `activeBrain`. This brain model is defined in the `Talairach` coordinate system, which currently belongs to one of the two standardized brain comparison systems, along with the `MNI`(Montreal Neurological Institute) coordinate system. The coordinates in the `pial_talairach.mat` file define the vertices and triangles, modelling the so-called "pial surface": the pial surface is created by expanding the white matter surface so that it closely follows the gray matter-cerebrospinal fluid boundary. The model was imported into `activeBrain` from the `AFNI SUMA` dataset. `AFNI` is a set of C programs for processing, analyzing, and displaying functional MRI (FMRI) data - a technique for mapping human brain activity. `SUMA` is a program that adds cortical surface based functional imaging analysis to the `AFNI` suite of programs [2].

2.2 Electrode locations

The electrode locations should match the surface of the model. Therefore, they must be specified using the same coordinate system(`Talairach`, by default). Of course, one can transform the `Talairach` coordinates into a different coordinate system (as for example `MNI`). So that the vertices of a brain model(or coordinates of electrodes) can be transformed[1],`activeBrain` provides the `transformBrain` function. Please use `help transformBrain` for further information.
Because the electrodes will never match the cortical model exactly, they must be projected onto the model surface. For this purpose, the `Geometry` Matlab procedures [1] were used. The projection implementation mainly benefits from the triangle-line intersection algorithm described in [7]. Before performing the projection, a brain model must be flattened so that the electrodes do not possibly project into the sulci of the model. For an example on how `demo` flattens the brain, see 2. See also 4.1 for further information. There are two approaches that `activeBrain` uses for projecting the electrodes:

- projection towards a specified origin of the model
- projection using the average normal vector of the neighboring triangles

The projection using the average normal vector should be used preferably, since this type of projection does not depend on the choice of the origin. This type of projection is used in the supplied demo-please see 4.4 and figure 3. For further information on the `projectElectrodes` procedure, see 6.1.

2.3 Computation of the activations

Because the values of interelectrode activations(signals) are unknown, we have to assume that the signal is conducted through the cortical tissue and that the signal value in the electrode neighborhood will constant or decreasing("fading") as a function of the euclidian distance. This assumption is made by the `electrodesContributions` function. The supported fading is either linear or gaussian. An example of the linear fading is provided on figure 1. Here, two different activation values corresponding to two

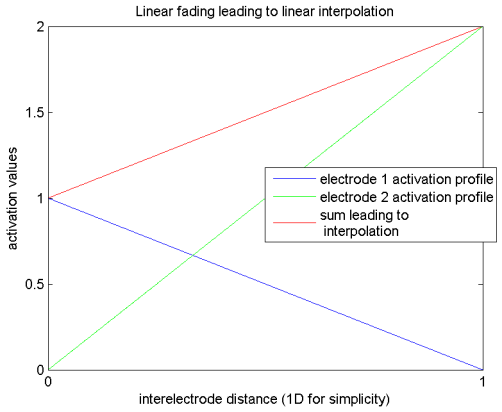


Figure 1: Linear fading leading to linear interpolation

electrodes are linearly fading to zero. The sum of the activations at every point of the cortical surface is the linearly interpolated activation value between these two electrodes. An analogical principle accompanies the gaussian type of fading. The speed and the type of fading should be set by visually inspecting the results; nevertheless, the fading distance should always be less than the interelectrode distance - so that the value at an electrode is not altered by the contributing value of another near electrode.

[1] a standardized coordinate system(`Talairach`, `MNI`) will be used in most cases; thus the transformation will normally be useful only to transform between the two standardized coordinate systems

The `electrodesContributions` function performs only the computation of the fading multipliers. The actual summation is done by `activateBrain`. The latter also performs the averaging among several subjects, if electrode locations and activations for these subjects are supplemented. For further information on the parameters that `electrodesContributions` uses, please refer to 6.2.

2.4 Displaying the activations

After the electrodes have been projected onto a flattened surface (`projectElectrodes`) and electrode contributions have been computed (`electrodesContributions`), one is ready to display the activations on the brain by running `activateBrain`. The way this procedure plots can be customized by providing two structures as its parameters (see 6.3). An interesting feature of the `cmapstruct` structure is the fading of certain pseudocolors(i.e. those with an insignificant value) into the color of the brain. One can specify which activation values are insignificant - it can be zero, a plus or a minus value. First, the colormap is set to embody a certain range of activation values. Then, one specifies a "strip"(a vector that is a part of the colormap vector) of the colormap values which one wishes to fade. This way, all the activations whose values correspond to the values of this strip will fade into the color of the brain. The middle value of the strip stands for the pseudocolor of the brain, the edges of the strip define the starting points of the fading. See 6.3 for more information.

3 Procedure

The `activeBrain` package does not require any special installation. The `demo` script will set all the directories it needs to use onto the Matlab path automatically.
The `activeBrain` package consists of the following directories and files:

```
ACTIVEBRAIN
   activateBrain           - Visualize activations on a brain model.
   electrodesContributions - Compute the coefficients of activated vertices.
   projectElectrodes       - Project electrode grids onto a brain model.
   recordBrain             - Capture activateBrain's graphical output into an avi file.
   viewBrain               - Display a brain model and/or electrode grid locations.

DEMO
   demRecordBrain - This script demonstrates how recordBrain can be called.
   demo           - The activeBrain demo.

TRANSFORMMODEL
   coarserModel   - Make a brain model coarser by calling reducepatch.
   finerModel     - Make a brain model finer by adding triangle centroids as new vertices.
   hullModel      - Smooth a brain model by computing its convex hull.
   smoothModel    - Smooth a brain model.
   transf3Dmatrix - Compute general 3D transformation matrix.
   transformBrain - Transform all vertices of a brain model.

GEOMETRY
   dmat_transpose_print              - prints a double precision matrix, transposed.
   dmat_transpose_print_some         - prints some of a double precision matrix, transposed.
   dvec_print                        - prints a real vector.
   line_exp_is_degenerate_nd         - finds if an explicit line is degenerate in ND.
   plane_normal_line_exp_int_3d      - intersection of plane and line in 3D.
   s_len_trim                        - returns the length of a nonblank character string.
```

```
triangle_contains_line_exp_3d      - finds if a line is inside a triangle in 3D.
triangle_is_degenerate_nd          - finds if a triangle is degenerate in ND.
```

4 Results: The activeBrain demo

This demo demonstrates how to use the activeBrain package functions in order to display activations on a brain model.

To display the activations, the following data must be available:

- brain model (in this demo – `pial_talairach.mat`)
- electrode locations for desired subjects (`DEMOsubj.mat`)
- activation data for these electrodes (`DEMOsubj.mat`)

Briefly, this demo proceeds as follows:

- flatten the brain model (`coarserModel, smoothModel, hullModel`)
- project the electrodes on the flattened model (`projectElectrodes`)
- reload the dense brain model (`pial_talairach.mat`)
- compute the electrode contributions to the associated brain's surface areas (`electrodesContributions`)
- display the activations (`activateBrain`)
- capture a demo activation sequence into an .avi file (`recordBrain`)

Demo contents

This demo is a run of `demo`, published by `Matlab`. The published TEX code was manually split into the following sections:

- Preprocessing a brain model
- Specifying the electrode grids
- Viewing the brain and electrode locations
- Projecting the electrodes
- Computing the electrode contributions
- Visualizing the activations

4.1 Preprocessing a brain model

A brain model must be first flattened, i.e. deprived of its sulci; otherwise, some electrodes of subjects' electrode grids would project into the sulci - which would create an uneven distribution of the electrodes; thus, the electrode grid would be deformed by the projection, creating artifacts when visualizing the activations.
A brain model is first made coarser by using the `coarserModel` function. It is used for the computation of the electrode projections; there will be almost no difference in respect of the projected electrodes between

a very dense or a coarse brain model; however, the desired shape of the model should be maintained (if the reduction is excessive, the model "shrinks").
Further, the brain model might or might not be smoothed by using the `smoothModel` function.
Finally, the brain model should be flattened by `hullModel`.
Thus, two major approaches are used to flatten a brain model:

- make the model coarser and compute its convex hull (approach A)
- make the model coarser, smooth it and compute its convex hull (approach B)

Which of the two approaches should be used depends on the location of the electrodes - i.e. on the way they are implanted and on subjective brain proportions.
The approach B maintains a better shape of the brain but the electrodes are projected slightly underneath the convex hull (not exactly on the level of sulci but a little underneath) - it is because the smoothing slightly shrinks the model used for the projection. The approach A normally delivers good visual results and does not need to compute the smoothing.
In both cases:

- reduce the model to about 10000 triangles

(the reduction is also necessary for `smoothModel`(approach B) so that it can terminate in a reasonable amount of time)

load the SUMA talairach brain:

```
load pial_talairach;
```

this creates a variable cortex

make the model coarser:

```
cortexcoarser = coarserModel(cortex, 10000);
```

Approach A
compute the convex hull:

```
cortex=hullModel(cortexcoarser);
```

this deprives the brain model of its sulci

Forget the results of the approach A and use the results of the
Approach B

use the coarse model and make it smoother:
(see also `smoothModel` for further information on the arguments used)

```
origin = [0 20 40];
smoothrad = 25;
mult = 0.5;
[cortexsmoothed] = smoothModel(cortexcoarser, smoothrad, mult);
```

compute the convex hull:

```
cortex = hullModel(cortexsmoothed);
```

4.2 Specifying the electrode grids

The electrode grids for each subject are specified by the structure `struct('electrodes', Nsubjx3matrix)` Specify which example you want to see by loading appropriate `DEMOsubj<2,3,4>`.

load a subject with 32 electrode positions (and–for later visualization of the activations - with 2 columns of activations at these electrodes):

```
load DEMOsubj;
```

this creates a variable subj; for several subjects, use a field of these structures so that also average activations can be computed
`DEMOsubj2` loads a structure field (two subjects) with similar overlaying electrode grids in order to explore the averaging capability of `activateBrain`.
`DEMOsubj3` are two subjects with only one electrode each - this demostrates the averaging more clearly than `DEMOsubj2`.
`DEMOsubj4` is only one subject with electrodes close to one another so that the interpolation of the values between them can be observed. If you want to load `DEMOsubj<2,3,4>` please modify the save command further in the code so that the subject example data is not overwritten.

4.3 Viewing the brain and electrode locations

To look at the brain model and the electrode grids, use `viewBrain`.
(see also `viewBrain`). The output of this part of code is shown on figure 2.

```
clf; set(gca, 'Color', 'none'); set(gcf, 'Color', 'w'); grid on;
viewBrain(cortex, subj, {'brain', 'electrodes'}, 0.7, 32, [110,20]);
title('Flattened brain model and electrode locations');
```

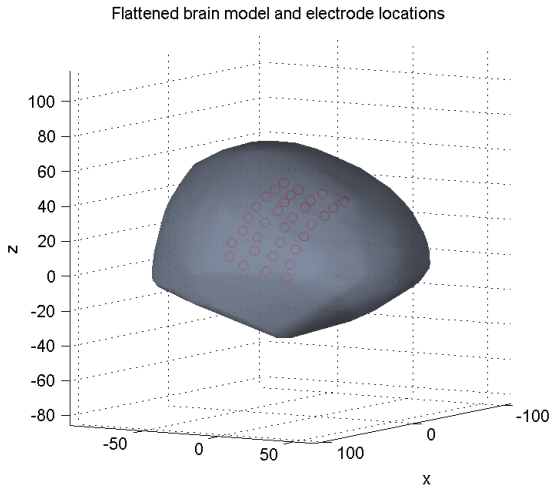


Figure 2: Flattened brain model and electrode locations

4.4 Projecting the electrodes

The electrodes are projected by calling the `projectElectrodes` function. The output of this part of code is displayed on figure 3.

project the electrodes onto the surface of the brain:

```
normdist = 25;
```

(using the normal vector projection; the normal vector is computed by averaging the normal vectors at the electrode distance `normdist` and less; see also `projectElectrodes` for a more thorough information on its arguments)

```
[ subj ] = projectElectrodes( cortex, subj, normdist);
```

You might want to save the computed electrodes projections so that this does need to be done again in the future - so that you do not need to`electrodesContributions` contribution kernel function or its parameters.

```
save DEMOsubj.mat subj;
```

look at that brain and the projected electrode grids, using `viewBrain` again:

```
clf;
set(gca, 'Color', 'none'); set(gcf, 'Color', 'w'); grid on;
viewBrain(cortex, subj, {'brain','electrodes', 'trielectrodes'}, 0.7, 32, [185,0]);
title('Flattened brain model, original electrodes and projected electrodes');
```

```
Projecting electrodes:
   processing subject 1
      processing electrode 1
           computing the distance from the electrode for all vertices
           computing the normal vector
           computing the intersection with the triangles
           (original electrode coordinates)
   32.0000   37.9000   43.1000

           (projected electrode coordinates)
   30.1113   35.4809   39.4863
                  .
                  .
                  .
      processing electrode 32
           computing the distance from the electrode for all vertices
           computing the normal vector
           computing the intersection with the triangles
           (original electrode coordinates)
   62.0000  -20.3000   11.1000

           (projected electrode coordinates)
   63.1418  -20.2083   11.2611
```

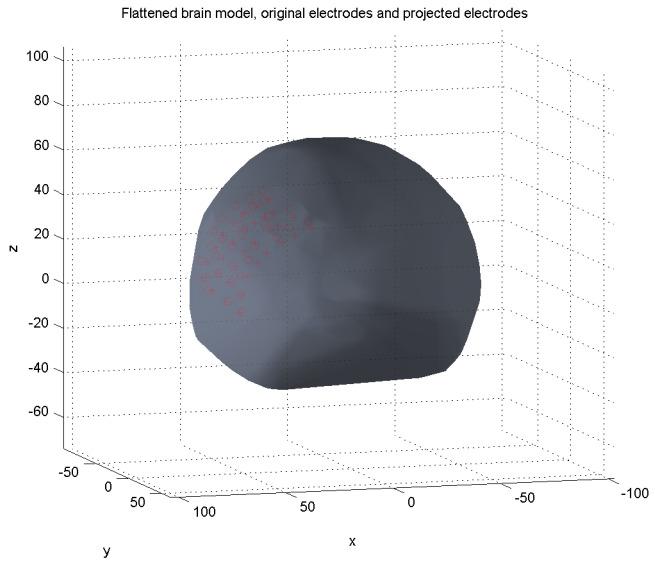


Figure 3: Flattened brain model, original electrodes and projected electrodes

4.5 Computing the electrode contributions

The electrode contributions are computed by running the `electrodesContributions` function. One should be using the fine brain again for the computation of the electrode contribution (because the final visualization `activateBrain` will be using this model).

reload the original SUMA talairach brain:

```
load pial_talairach;
```

`pial_talairachRED.mat` would also do (uses half of triangles - which is useful when visualizing the brain or capturing an .avi movie

compute the contributions of the given electrodes:

```
kernel = 'linear';
param = 10;
cutoff = 10;
```

see also `electrodesContributions` for a more thorough information on its arguments)

```
[vcontribs ] = electrodesContributions( cortex, subj, kernel, param, cutoff);
```

`param` should be set to the interelectrode distance `cut-off`: only those vertices vert whose `distance(vert, electrode) < cut-off` are considered to be altered by a near electrode (the more distant ones will not be displayed by `activateBrain`)
if you want to define a value of 0 for all vertices spanned by the electrode grid, then use a larger cutoff (the triangles will be taken into account and so displayed but their value will be 0 since `param` for the linear kernel specifies the distance at which the activations falls to zero)

```
Computing the electrode contributions to the vertices:
```

```
processing subject 1
   processing electrode 1
        computing distances
        OK
        computing contributions
        OK
             .
             .
             .
   processing electrode 32
        computing distances
        OK
        computing contributions
        OK
```

You might want to save the contributions for this subject and brain so that this does need to be done in the future (but always use the according `subj` structure and brain model when visualizing the data).

```
save DEMOvcontribs.mat vcontribs;
```

4.6 Visualizing the activations

The activations can be plotted by calling the `activateBrain` function.

set what you want to see:
(please consult `activateBrain` for further information on the following structures)

```
viewstruct.what2view = {'brain', 'activations'};
viewstruct.viewvect = [120, 10];
viewstruct.material = 'dull';
viewstruct.enablelight = 1;
viewstruct.enableaxis = 0;
viewstruct.lightpos = [200, 0, 0];
viewstruct.lightingtype = 'gouraud';
```

and how you want to see it:

```
cmapstruct.cmap = colormap('Jet'); close(gcf); %because colormap creates a figure
cmapstruct.basecol = [0.7, 0.7, 0.7];
cmapstruct.fading = true;
cmapstruct.ixg2 = floor(length(cmapstruct.cmap) * 0.15);
cmapstruct.ixg1 = -cmapstruct.ixg2;
cmapstruct.enablecolormap = true;
cmapstruct.enablecolorbar = false;
```

Run `activateBrain`
NOTE:
Do not forget to use the same brain model as the one used for the precomputation of the contributions (that is, the fine one). Also, use the `subj` that is the output of `projectElectrodes`).

```
ix = 2;
cmapstruct.cmin = min(subj(1).activations(:,ix));
cmapstruct.cmax = max(subj(1).activations(:,ix));
```

```
clf; set(gca, 'Color', 'none'); set(gcf, 'Color', 'w'); grid off;
activateBrain( cortex, vcontribs, subj, ix, cmapstruct, viewstruct );
```

```
Computing the activations....done
Displaying....done
```

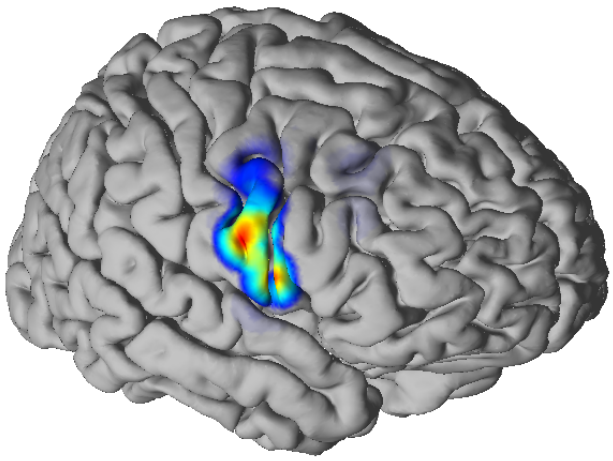


Figure 4: Demo one-subject activations

5 Conclusions

The activateBrain Matlab toolbox was designed under the MathWorks Matlab 7.0 environment. This version enables JIT acceleration of for loops. All activateBrain loops are written in such a way that the acceleration can be used. The code was further checked by M-Lint Code Checker and profiled by Matlab profiler to ensure maximum p-code performance, which is a vital aspect of processing brain models with high number of triangles.

6 Appendix

6.1 projectElectrodes

```
function [ subjstructs, M ] = projectElectrodes( M, subjstructs, normway, interstype, intersval)
PROJECTELECTRODES  Project electrode grids onto a brain model.

   Projects the electrode grids (for every subject they are specified by
   a structure and the field of these structures builds subjstructs) onto
   the model M.
   The altered subjstructs is required by the electrodesContributions( M,
   subjstructs, kernel, param, cutoff) function.

   This procedure also allows to add the intersecting vertices and
   triangles resulting from the projection onto the model M (see also NOTE below).


   CALLING SEQUENCE:
   [ subjstructs ] = projectElectrodes( M, subjstructs, normway, interstype, intersval)

   [ subjstructs ] = projectElectrodes( M, subjstructs, normway )
           is equivalent to
   [ subjstructs ] = projectElectrodes( M, subjstructs, normway, '' )

   INPUT:
       M:              struct('vert', Vx3matrix, 'tri', Tx3matrix) - brain model
       subjstructs:    field of structures, for each subject:
                       struct('electrodes', Nsubjx3matrix)
       normway:        controls the way the normal vector to the surface is computed
           possible values: [x, y, z] - every electrode is projected onto the
                                        surface using the
                                        normal vector line [electrode - [x, y, z]]
                            normdist (double) - only those vertices vert whose
                                                distance(vert, electrode) < normdist
                                                are used to compute the average normal
                                                vector from the normals at these vertices
       interstype:     controls the way near triangles are obtained to compute
                       the intersections
           possible values: '' [default] - all triangles of the model are processed
                            'fixed' - specify a radius intersval (see below)
                            'multipl' - the distance mindist of the closest vertex to
                                        an electrode is multiplied by intersval to
                                        obtain intersdist; all triangles of vertices vert
                                        for which distance(vert, electrode) < intersdist
                                        will then be used for the line x triangle
                                        intersection algorithm
                                        (i.e. electrode projection onto the cortex)
       intersval:      please refer to interstype

       Remark: if interstype ~= '' and if for an electrode no intersections
       can be found, please change interstype or increase intersval.
```

```
OUTPUT:
    subjstructs:    field of structures for each subject:
                    struct('electrodes', Nsubjx3matrix, 'trielectrodes', Nsubjx1matrix)
                    - enhanced subjstructs where 'trielectrodes' are
                    the projected electrode coordinates
    M:              (now redundant, see NOTE below)
                    struct('vert', Vx3matrix, 'tri', Tx3matrix) -
                    altered brain model
                    (electrode intersections added as new vertices
                    and resulting triangles)

NOTE:
    The PROJECTION PART (see code below) has been commented out because
    the addition of new vertices and triangles at the intersection points creates
    shading artifacts (the intersected triangle is more apparent because it
    is now 3x finer than the surrounding ones). When commented out, the
    model M is not altered; however, if one is about to use a coarse M, one
    might want to activate this part of the code and return M because one might
    want to see the exact value of the electrode at that point
    (for fine models M, the values at near vertices are almost identical
    with the one that would be projected).

Example:
    [ subjstructs ] = projectElectrodes( M, subjstructs, 25)
        projection using the surrounding normal vectors; most commonly used for models
        that contain low number of vertices (< 10000) - every brain
        model can be reduced to about this number without affecting the
        accuracy of the projection - and this is normally done (see
        also demo.m)

    [ subjstructs ] = projectElectrodes( M, subjstructs, [0 -10 20])
        projection towards the center point - this is faster than using
        the normals but is less accurate - a brain model is not a
        perfect sphere with an origin

    [ subjstructs ] = projectElectrodes( M, subjstructs, 25, 'fixed', 20)
        projection using the surrounding normal vectors, using fixed distance for
        reducing the number of considered triangles and thus speeding
        up the projection

    [ subjstructs ] = projectElectrodes( M, subjstructs, [0 -10 20], 'multipl', 5)
        projection towards the center point, using flexible distance for
        reducing the number of considered triangles and thus speeding
        up the projection
```

6.2 electrodesContributions

```
function [ vcontribs ] = electrodesContributions( M, subjstructs, kernel, param, cutoff)
ELECTRODESCONTRIBUTIONS    Compute the coefficients of activated vertices.

   Computes contribution values of subject electrode grids at affected brain
   model vertices (those vertices that are near the electrodes as specified
   by the parameters) - see also INPUT: kernel.

   The output vcontribs is required by the
   activateBrain( M, vcontribs, subjstructs, range, cmapstruct, viewstruct ) function


   CALLING SEQUENCE:
       [ vcontribs ] = electrodesContributions( M, subjstructs, kernel, param, cutoff)

   INPUT:
       M:              struct('vert', Vx3matrix) - brain model (eventually the one
                       altered by projectElectrodes.m, see <help projectElectrodes>)
       subjstructs:    field of structures, for each subject:
                       struct('electrodes', Nsubjx3matrix)
       kernel:         a function whose values (at a certain vertex distance from
                       an electrode) is being used for the multiplication of the
                       electrode value (the multiplication is done by activateBrain);
                       the multiplier is understood to be the contribution of
                       an electrode to the vertex
           possible values: 'linear', 'gaussian'
       param:          parameter of the kernel function
                       (for 'linear', this is the point of the linear function at
                       which it is zero; for 'gaussian', this is the standart deviation)
       cut-off:        only those vertices vert whose
                       distance(vert, electrode) < cuf-off are considered to be
                       altered by a near electrode (the more distant ones will not
                       be displayed by activateBrain)

   OUTPUT:
       vcontribs:      struct('vertNo', index, 'contribs', [subj#, el#, mult;...])
                       - structure containing the vertices that are near
                       the electrode grids; this structure contains the multipliers

   REMARK:
       There are several electrodes on subject's electrode grid that can
       contribute with its multipliers to a vertex; the contribution is then a
       scalar product (a weigted sum) of these multipliers and electrode values;
       only the contributions of distinct subjects are averaged by activateBrain.

   Example:
       [ vcontribs ] = electrodesContributions( M, subjstructs, 'linear', 10, 10)
           computes contributions using linear 'fade-out' function,
           supposing that the interelectrode distance is 10 (the parameter
           cutoff might be set to a higher value when using a single
```

```
subject (nicer smearing), but should be set equal to param for
multiple subject usage (so that zero values far from the
subject electrode do not contribute to the averaging at a
vertex close to the electrode of another subject)
```

6.3 activateBrain

```
function [ ] = activateBrain( M, vcontribs, subjstructs, range, cmapstruct, viewstruct )
ACTIVATEBRAIN  Visualize activations on a brain model.

   Data for each subject passed in the subjstructs field (see also INPUT).

   The subjstructs.trielectrodes and vcontribs are output of the
   projectElectrodes and electrodesContributions functions, respectively.

   Each subjectstructs(i).activation matrix NsubjxLsubj may consist of
   different number of activations Lsubj; however, it must consists of
   equal number of electrodes Nsubj as is the length of
   subjectstructs(i).electrodes.

   The activations can be displayed sequentially as specified by range.


   CALLING SEQUENCE:
       activateBrain( M, vcontribs, subjstructs, range, cmapstruct, viewstruct )

   INPUT:
       M:              struct('vert', Vx3matrix, 'tri', Tx3matrix) - brain model
                       (eventually the one altered by projectElectrodes,
                       see <help projectElectrodes>)
       vcontribs:      struct('vertNo', index, 'contribs', [subj#, el#, mult;...])
                       - structure containing the vertices that are near
                       the electrode grids; this structure contains the multipliers
       subjstructs:    field of structures, for each subject:
                       struct('activations', NsubjxLsubjMatrix, ...
                       'trielectrodes', Nsubjx1Matrix)
                        - enhanced subjstructs (output of projectElectrodes),
                        where 'trielectrodes' is a matrix of coordinates
                        of the projected electrodes
       range:          vector specifying which samples of the activation data matrix
       are to be sequentially displayed; its maximum value is {min(Lsubj) for all subj}
       cmapstruct:     controls the mapping of the values onto the colorbar,
                       see cmapstruct below
       viewstruct:     specifies the viewing properties, see below

      structure cmapstruct:
       cmapstruct.cmap - the colormap to use (e.g. colormap('Jet'))
       cmapstruct.basecol - the RGB triples that specifies the colour that the
                            colormap fades into (e.g. [0.7, 0.7, 0.7])
       cmapstruct.fading - boolean value specifying whether or not you want to
                           use the fading capability (if set to false, the first
                           value of colormap will be set to basecol and
                           the rest of the cmap remains untouched)
       cmapstruct.ixg1 and
       cmapstruct.ixg2
           The previous two indices are spanning the range at cmapstruct.cmap
```

```
        that will be faded into basecol; they represent a "fading
        strip of cmap" - if the strip is positioned somewhere centrally on
        the cmap, then the colours
        will fade into the middle value of <ixg2 and ixg1> from both sides; if the
        fading is supposed to happen at the edges of the colorbar, please set:
        for fading at low values, please set:
         ixg2 to the index of cmap whose value starts the fading into basecol
         ixg1 = -ixg2 (because the fading affects always only a half of the strip
         from each side)
         (e.g. ixg2 = 15; ixg1 = -15;)
        for fading at high values, please set:
         ixg2 = length(cmap) + half_of_the_strip_length
         (because the fading affects always only a half of the strip from each side)
         ixg1 = length(cmap)- half_of_the_strip_length
         (because the fading affects always only a half of the strip from each side)
         (e.g. ixg2 = 64 + 15; ixg1 = 64 - 15;)
    cmapstruct.cmin - the value of the signal that is considered to be the
                      minumum of the colormap (cmapstruct.basecol will be
                      preserved as the first index of colormap)
    cmapstruct.cmax - the value of the signal that is considered to be the
                      maximum of the colormap
    cmapstruct.enablecolormap - boolean specifying whether colormap is applied
    cmapstruct.enablecolorbar - when enablecolormap true, this boolean specifies
                                whether colorbar is displayed

   structure viewstruct:
    viewstruct.what2view - a column cell of strings specifying what
                           shall be visualized:
        possible values: 'brain' - shows the grey brain
                         'activations' - shows the activations
                         'electrodes' - shows the original electrode locations
                         'trielectrodes' - shows the projected electrode locations
                              (e.g. {'brain', 'activations'} )
    viewstruct.viewvect - vector used by the view command (e.g. [-90, 0])
    viewstruct.material - string used by the material command (e.g. 'dull')
    viewstruct.enableaxis - boolean specifying whether axes are displayed or not
    viewstruct.enablelight - boolean specifying whether light is used or not -
                             it should be always used so that the surface
                             of the brain is visible with all its sulci
    viewstruct.lightpos - vector specifying the coordinates of the light
                          (respectively to current axes), used by light command
                          (e.g. [-200, 0, 0])
    viewstruct.lightingtype - string specifying the type of lighting technique,
                              used by the lighting command (e.g. 'gouraud')

Example:
    activateBrain( M, vcontribs, subjstructs, 1, cmapstruct, viewstruct )
        see cmapstruct and viewstruct example (usual) values above
```

References

[1] John Burkardt. GEOMETRY Matlab routines, 2005.

[2] RW Cox and JS Hyde. Software for analysis and visualization of functional magnetic resonance neuroimages. *Computers and Biomedical Research*, 10:171–178, 1997.

[3] A.M. Dale, Bruce Fischl, and M.I. Sereno. Cortical Surface-Based Analysis I: Segmentation and Surface Reconstruction. *NeuroImage*, 9(2):179–194, 1999.

[4] F Darvas and D Pantazis. Mapping human brain function with MEG and EEG: methods and validation. *NeuroImage*, 23(supplement 1):S289–S299, 2004.

[5] Online developers. Bioelectromagnetism Matlab Toolbox, 2005.

[6] Eric C. Leuthardt, Gerwin Schalk, and Jonathan R Wolpaw. A braincomputer interface using electrocorticographic signals in humans. *J. Neural Eng. 1*, pages 63–71, 2004.

[7] Steve Marschner. Simple ray-triangle intersection. Cornell University, 2003.