Table of Contents

Table of Contents

1 Chapter 01: Getting Started with NFT

1.1

The Neuroelectromagnetic Forward Head Modeling Toolbox is an open-source software toolbox running under MATLAB (The Mathworks, Inc.) for generating realistic head models from available data (MRI and/or electrode locations) and for solving the forward problem of electro-magnetic source imaging. The toolbox includes tools for segmenting scalp, skull, cerebrospinal fluid (CSF) and brain tissues from T1-weighted magnetic resonance (MR) images. After extracting the segmented tissue volumes, mesh generation can be performed. When MR images are not available, it is possible to warp a template head model to measured electrode locations to obtain a better-fitting head model. The toolbox also includes electrode scalp mesh co-registration and generation of a uniform source space inside the brain volume for to be used in coarse source localization. The Boundary Element Method (BEM) is used for the numerical solution of the forward problem. Toolbox functions can be called from either a graphic user interface or from the command line. Function help messages and a tutorial are included. The toolbox is freely available under the GNU Public License for noncommercial use and open source development.

The toolbox uses the following third party tools and libraries for segmentation, mesh generation and forward problem solution. The source codes for these tools are available.

1. ASC - for triangulation of 3D volumes.

2. Qslim - for mesh coarsening.

3. Matitk - Matlab interface to the ITK image processing toolkit.

4. Metu-bem - Boundary Element Method solver.

The NFT toolbox provides a user interface (UI) for segmentation, mesh generation and for creating the numerical head model. It also has a well defined MATLAB command-line interface.

This manual explains how to use the NFT toolbox. The head modeling UI, the command line API and the structures are described. An overview of the implementation is provided.

The next section describes the installation of the toolbox. The Getting Started section provides an overview of the interface. Head modeling from 3D MR images is described next, followed by head modeling from template warping. This is followed by a section on forward modeling and examples. The final section is a summary of all toolbox functions and commands.

1.2

This section describes installation and configuration of the NFT Toolbox. The following steps are necessary for a proper installation of the toolbox:

1. Extract or copy the toolbox directory to a suitable place on your computer file system.

2. The extracted directory will contain m-files, and C++ executables.

3. Add the toolbox directory to the MATLAB path. You can use the File ? SetPath menu item or the addpath() function. Under linux/unix, you may add the directory to the MATLABPATH.

The toolbox can also make use of the Matlab Parallel Processing toolbox (if installed) to distribute the computation of the transfer and lead-field matrices to multiple processors. To do this, before running NFT, the user must simply enter

```
>> matlabpool(n)  % where n is the number of compute nodes available
```

In parallel mode, wait bars do not appear while computing the transfer and lead-field matrices.

1.3

The toolbox starts by typing Neuroelectromagnetic_Forward_Modeling_Toolbox or NFT on command window. Main window appears as shown in Figure 1. This window is divided into three panels. The first panel is used to select the working folder, and to name the subject and the session. The NFM toolbox requires a subject folder to be specified at startup. All subject specific output is saved into this folder. The filenames are derived from the subject and session names entered into this panel. The second panel is the Head modeling panel. The head model can either be created from MR images, or a template head model can be warped to digitized sensors. The head modeling panel provides the following operations when creating a head model from MR images:

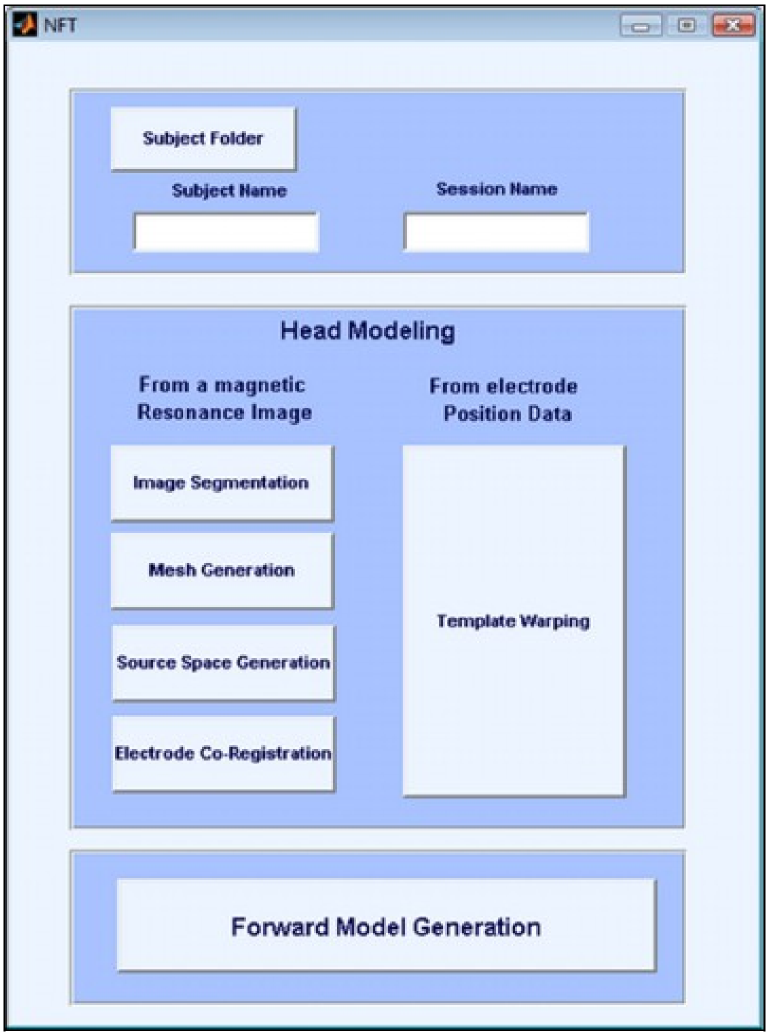


Image Segmentation

Interface for tissue classification from 3D MR Images.

Mesh Generation

Uses the segmentation results to generate realistic BEM meshes.

Source Space Generation

Generates a regular grid sources within the brain mesh.

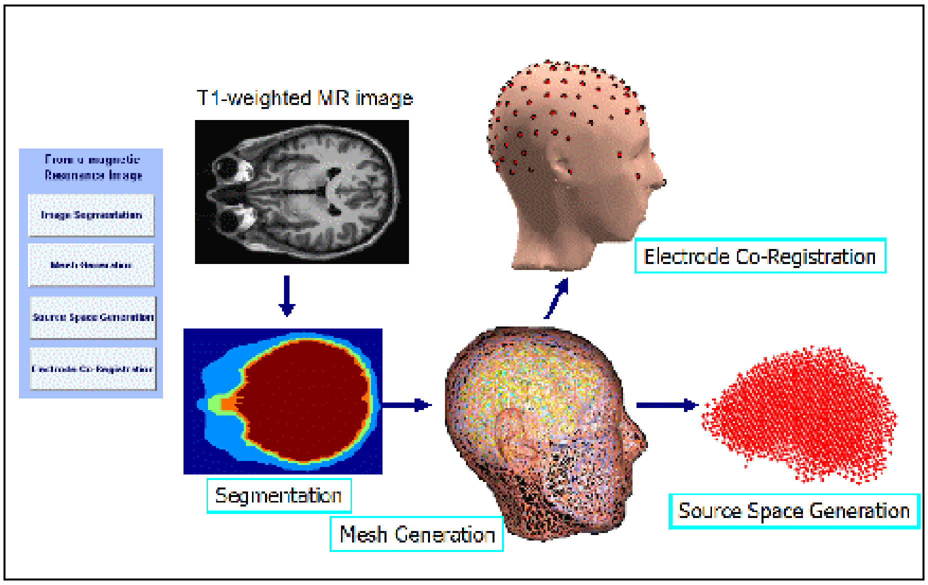
Electrode Co-Registration

Registers digitized electrode locations to the scalp mesh.

When generating a template-based head model from digitized electrode positions, the only option is Template Warping. The final panel in the main menu is for Forward Model Generation. This opens up the Forward Model Generation interface which is used to compute the BEM coefficient matrix, create the transfer matrices for each sensor, and generate lead field matrices for a source distribution.

2 Chapter 02: Head Modeling from MR Images

The steps of head modeling are segmentation, mesh generation, and co-registration of electrode locations with scalp surface. User may also generate a source space to be used in the solution of the inverse problem. Figure 2 shows the steps of head modeling using MR images.



Each step in realistic head modeling is implemented as a separate GUI module reachable from the main menu. These modules are described in the following sub-sections.

2.1

The first step in segmentation is to load the MR image. The input of the segmentation module is a 3-D sagittal T1-weighted MR image. The image format has to be in analyze format and the voxel size need to be 1×1×1 mm. To prepare the image for this toolbox, one may use Freesurfer software (http://surfer.nmr.mgh.harvard.edu/) to perform the following operations:

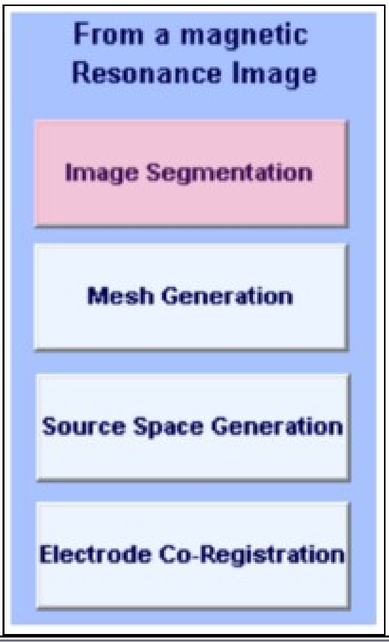
1. Inhomogeneity correction:
   mri_nu_correct.mni --i input_volume --o output_volume --n 2

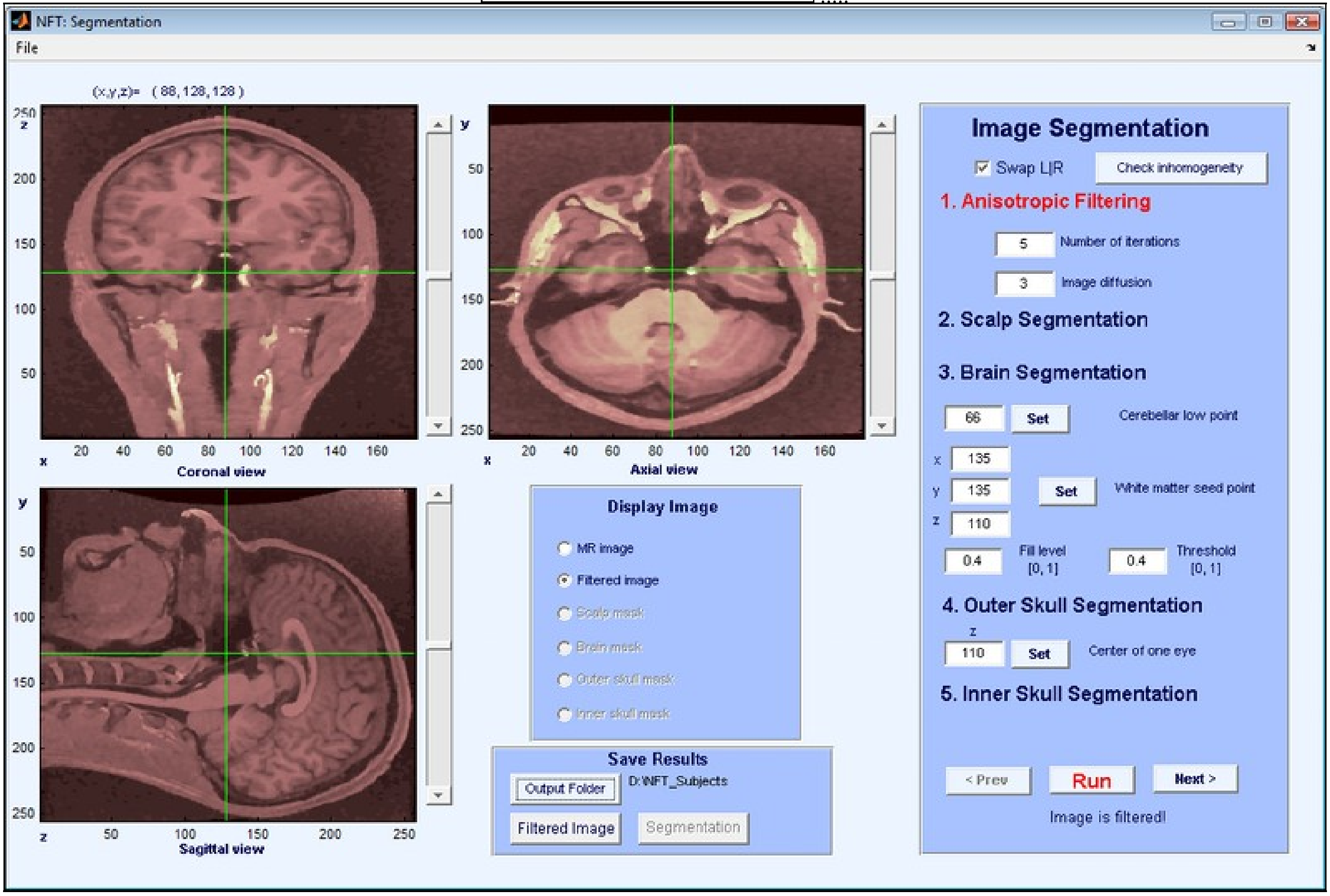
2. Conversion of the input volume to 1 mm volume data:
   mri_convert -i input_volume --conform_size 1 --o output_volume

3. Orient the image:
   mri_convert -i input_volume --out_orientation PSR -ot format
   -o output_volume

4. Save in analyze format:
   mri_convert -it analyze -i output_file_name.img -ot file_type
   -o input_volume

When the image is loaded, slices are shown in sagittal, axial, and coronal orientations and it is possible to select slices easily by using the scroll bars or clicking on the images (Figure 3). The Display image panel allows the user to select which image to display on the image panels. The available choices are the MR volume, the filtered volume or various stages of segmentation.





(x,y,z)= (88, 128, 128)

Image Segmentation

[✓] Swap L|R

1. Anisotropic Filtering

5 Number of iterations

3 Image diffusion

2. Scalp Segmentation

3. Brain Segmentation

66 Cerebellar low point

x 135
y 135 White matter seed point
z 110

0.4 Fill level [0, 1]
0.4 Threshold [0, 1]

4. Outer Skull Segmentation

z 110 Center of one eye

5. Inner Skull Segmentation

Image is filtered!

Output Folder: D:\NFT_Subjects

Display Image: [✓ Filtered image]

The panel on the right of the segmentation GUI shows the segmentation steps that will be performed on the volume in order:

1. Anisotropic filtering.
2. Scalp segmentation.
3. Brain segmentation.
4. Outer skull segmentation.
5. Inner skull segmentation.

The current step is highlighted in red. Pressing the Run button executes the segmentation step. It is possible to repeat a given step, changing parameters and observing the output. Pressing the Next button proceeds to the next step. Below is a discussion of each segmentation step:

2.1.1

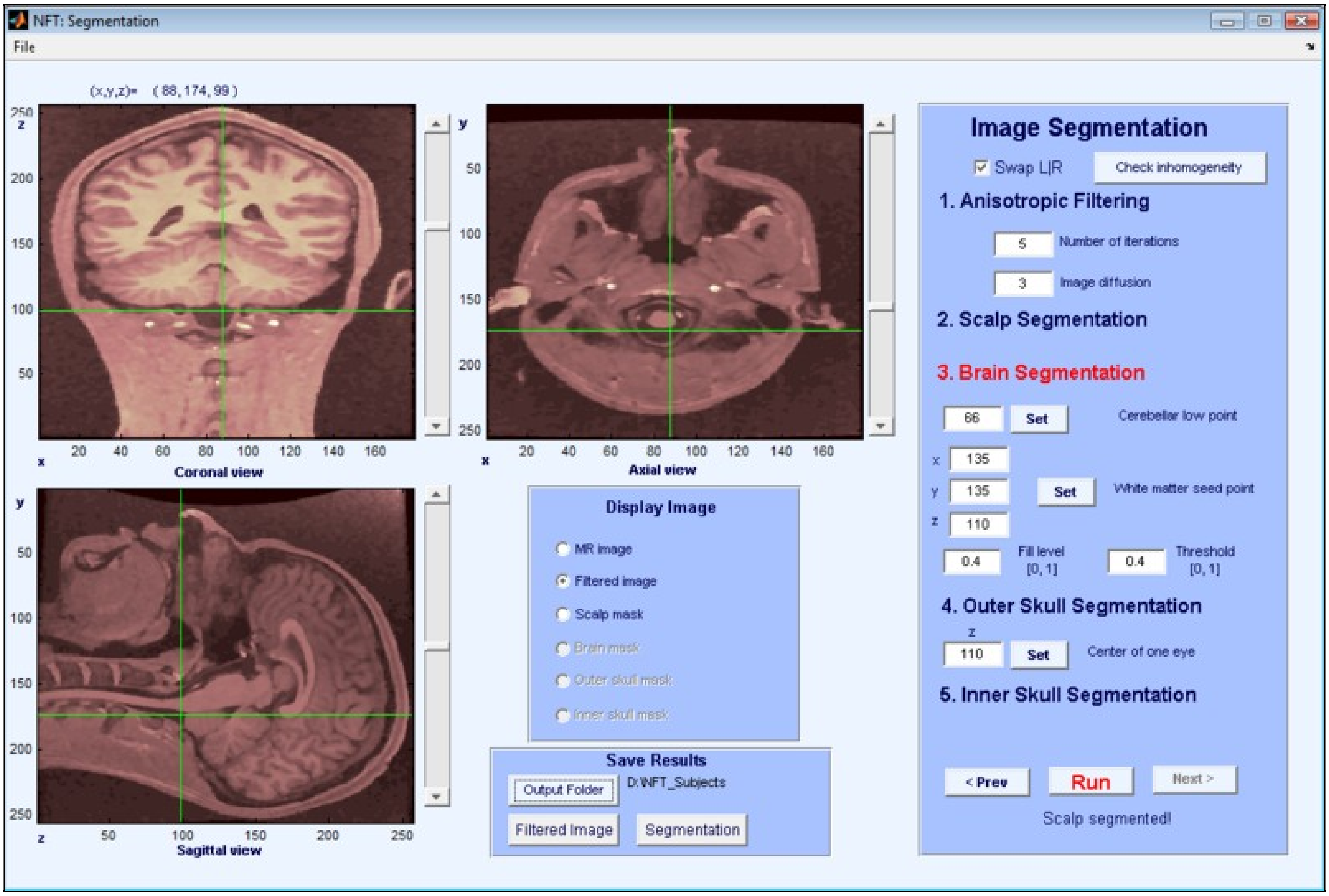
The purpose of anisotropic filtering is to enhance the image quality. This filter increases the SNR of the image while preserving the edges. The inputs to the anisotropic filtering are the number of iterations and image diffusion. The default values of 5 and 3 work well for most MR images. As the values increase, the image starts to get blurred. The output of anisotropic filtering can be observed by selecting ?Filtered Image? from the Display Image panel.

2.1.2

The next step is scalp segmentation, separating the background from the image. There are no user inputs to scalp segmentation. An automatic thresholding algorithm is applied, and the result can be observed by selecting ?Scalp Mask?.

2.1.3

The brain segmentation uses the watershed segmentation algorithm, which selects connected voxels starting from a seed point. To prevent the algorithm from overflowing the brain region, the lowest point of cerebellum has to be selected by the user. This point has to be marked on the image panels using coronal and saggital views of the slices. Once the cursor is at the lowest point of cerebellum, pressing Set + selects the point. Figure 4 shows cursor locations for setting the lowest point for cerebellum. Another input for brain segmentation is a seed point on the white matter, any point can be used as long as it is on the white matter (Figure 5). Pressing the Set + button fetches the cursor coordinates for the seed. The other inputs are the parameters of the watershed segmentation algorithm, and the default values work for most images. The result of Brain segmentation can be seen by selecting ?Brain Mask?.



NFT: Segmentation

File

(x,y,z)= (88, 174, 99)

Image Segmentation

[✓ Swap L|R] Check inhomogeneity

1. Anisotropic Filtering

5 Number of iterations

3 Image diffusion

2. Scalp Segmentation

3. Brain Segmentation

66 Set Cerebellar low point

x 135

y 135 Set White matter seed point

z 110

0.4 Fill level [0, 1]

0.4 Threshold [0, 1]

4. Outer Skull Segmentation

z

110 Set Center of one eye

5. Inner Skull Segmentation

< Prev Run Next >

Scalp segmented!

Display Image

MR image

[✓ Filtered image]

Scalp mask

Brain mask

Outer skull mask

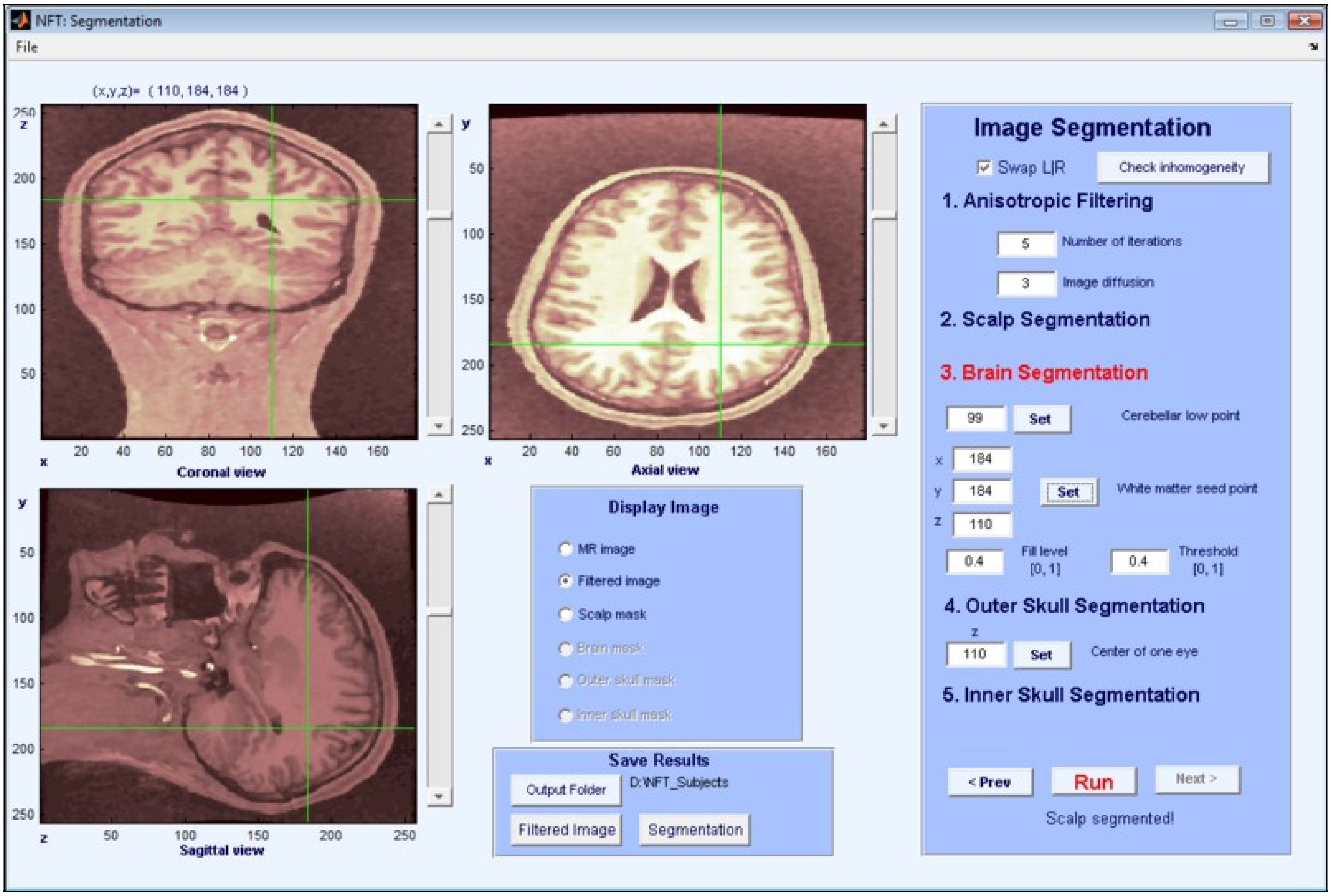
Inner skull mask

Save Results

Output Folder D:\NFT_Subjects

Filtered Image Segmentation

……



(x,y,z)= (110, 184, 184)

[✓ Swap L|R]

5 Number of iterations

3 Image diffusion

99 Cerebellar low point

x 184
y 184
z 110 White matter seed point

Fill level [0, 1] 0.4
Threshold [0, 1] 0.4

z 110 Center of one eye

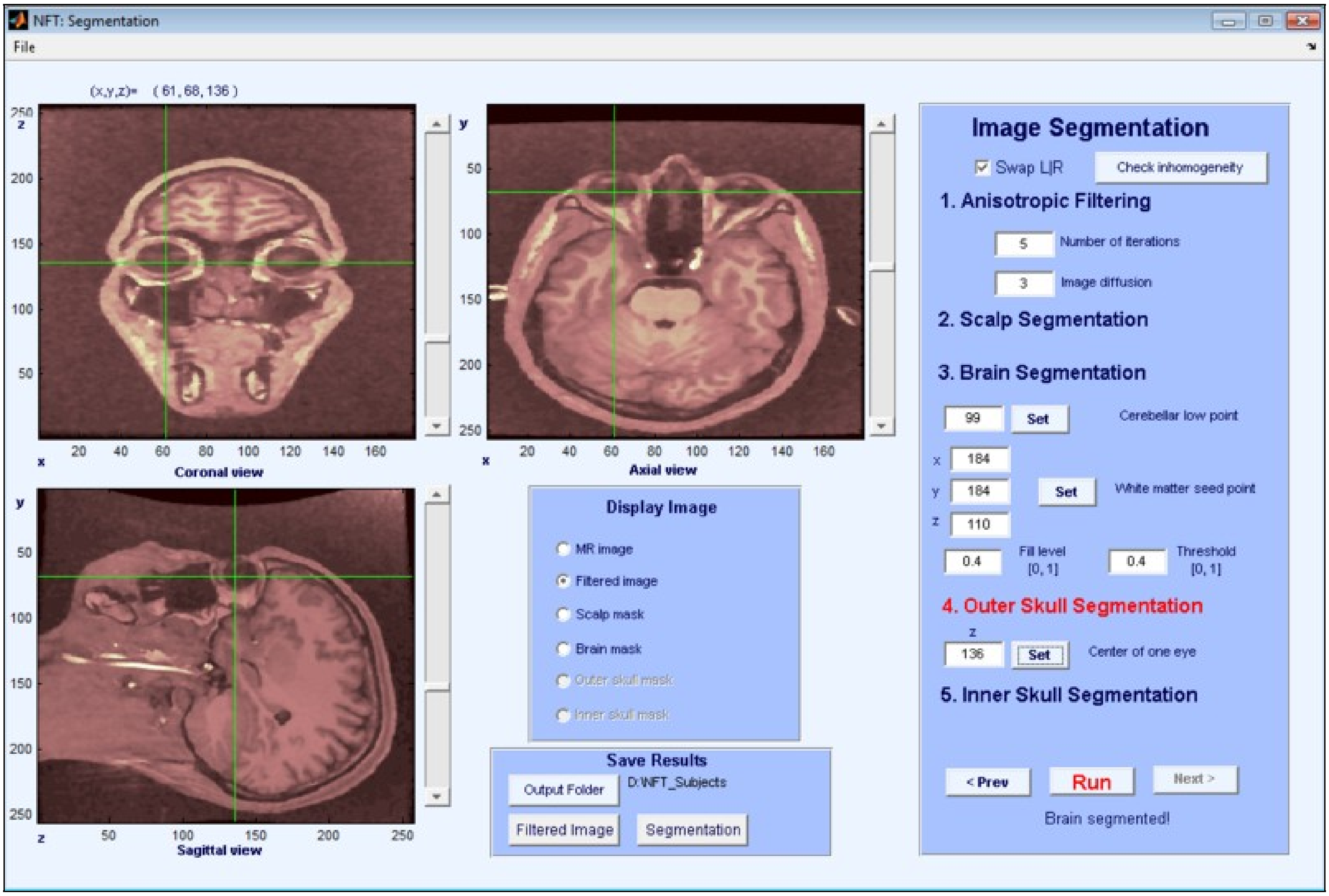
MR image
[✓ Filtered image]
Scalp mask
Brain mask
Outer skull mask
Inner skull mask

Output Folder D:\NFT_Subjects

Scalp segmented!

2.1.4

For outer skull segmentation, seed points for the eye lobes are selected by the user. Once a slice is selected where the eyes are clearly seen on the axial view (Figure 6), Set + is pressed to select that slice. During outer skull segmentation, an image window will pop-up for the user to click on both eye lobes. Figure 7 shows the matlab figure that pops-up to click on eye lobes. Once the eyes are selected the outer skull is segmented and can be seen by ?Outer skull mask?.



(x,y,z)= (61, 68, 136)

Image Segmentation

[✓ Swap L|R]

1. Anisotropic Filtering

5 Number of iterations

3 Image diffusion

2. Scalp Segmentation

3. Brain Segmentation

99 Cerebellar low point

x 184
y 184 White matter seed point
z 110

0.4 Fill level [0, 1]

0.4 Threshold [0, 1]

4. Outer Skull Segmentation

z 136 Center of one eye

5. Inner Skull Segmentation

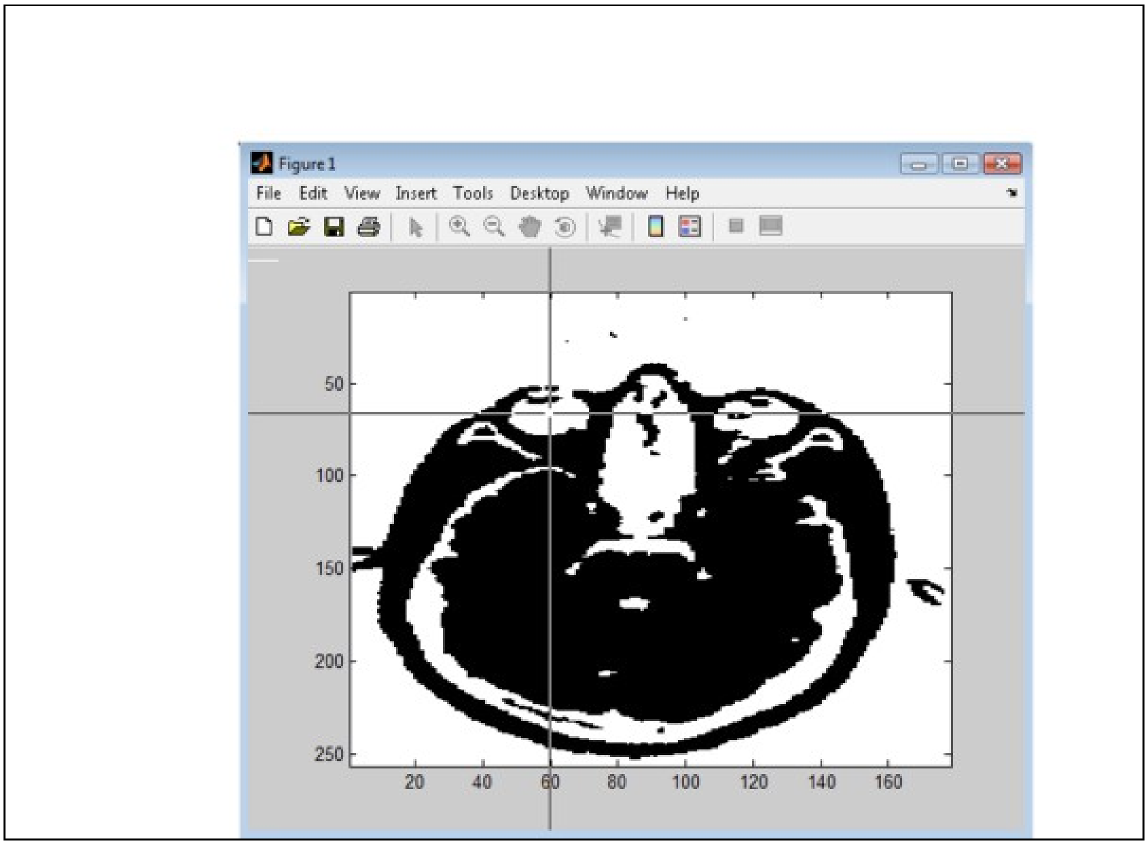
Brain segmented!

Display Image

MR image
[✓ Filtered image]
Scalp mask
Brain mask
Outer skull mask
Inner skull mask

Save Results

Output Folder D:\NFT_Subjects



2.1.5

Inner skull segmentation does not require any user input. After the inner skull is segmented, the outer skull and scalp are checked for intersections or very thin areas. These masks are corrected if there are any intersecting regions or too close regions which won?t be suitalbe for BEM modeling.

The outputs of the segmentation module are filtered MR images and scalp, skull, CSF and brain masks. It is possible to save the results during any stage of segmentation in Matlab data format. The filtered MR images are saved in Matlab format with double presicion, and the name would be Subject_name_filtered_images.mat. The masks are saved in structure format of Matlab, where the name would be Subject_name_segments.mat. When loaded in Matlab, the structure will look like as follows:

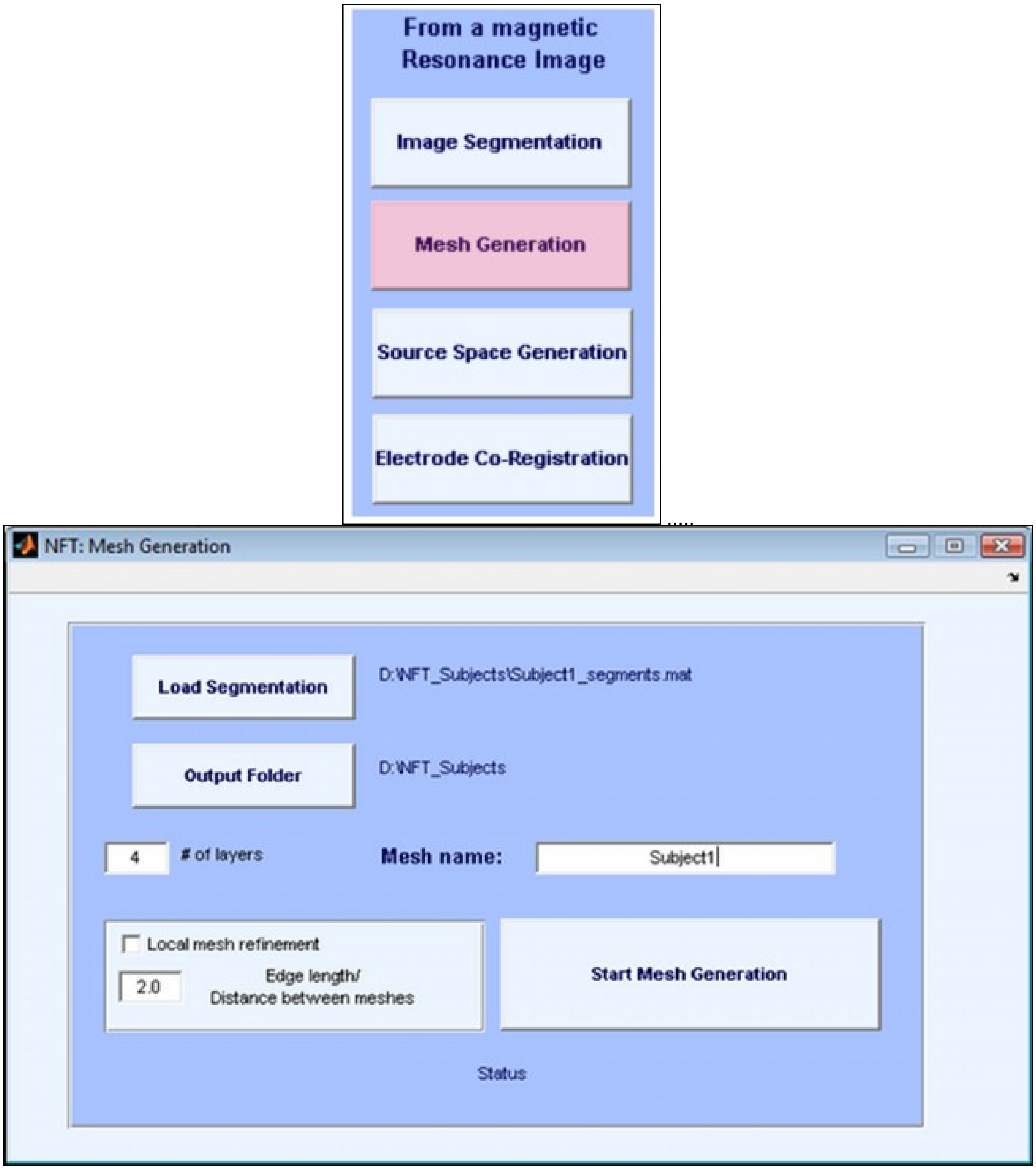
```
Segm =

         scalpmask: [256x258x257 logical]
         brainmask: [256x258x257 logical]
    outerskullmask: [256x258x257 logical]
    innerskullmask: [256x258x257 logical]
```

2.2

The second step in realistic head modeling is mesh generation. The mesh generation module uses the results of the segmentation and outputs the BEM mesh of the head. If the module is invoked from the Main Menu, it will use the Subject Name and Subject Folder selected in the Main Menu for segmentation files. The output folder is set to the Subject folder, and the mesh name is set to the Subject Name. It is possible to change the output folder and load a different segmentation which makes it possible to use the module as a standalone mesh generation tool.

The mesh generation module generates either 3-layer or 4-layer meshes. The number of layers is selected by the user. A three layer mesh has the scalp, skull and the brain regions separated by the scalp, skull and CSF surfaces. The CSF and the brain is considered as a single region. A four layer mesh models scalp, skull, CSF and brain regions, with an additional surface that separates the CSF and the brain.

The interface of Mesh Generation is shown in Figure 8. The generated mesh file is suitable to be used directly by the BEM solver. The format of the mesh file is given in Appendix A. The mesh generation process is described below.



D:\NFT_Subjects\Subject1_segments.mat

D:\NFT_Subjects

4 # of layers

Mesh name: Subject1

☐ Local mesh refinement

2.0 Edge length/ Distance between meshes

Mesh Generation module creates triangular meshes that fits the boundaries of the segmentation. The aim is to approximate the geometry while keeping the number of triangles small enough to prevent running out of resources in the BEM solver. The approach taken by the mesh generation module is to start with a very fine mesh of the surface boundary, and gradually coarsen it, making sure the topology is correct and the quality of the elements is high at each step.

For this purpose, three external programs and various Matlab functions are used. The external programs are Adaptive Skeleton Climbing (ASC) (http://www.cse.cuhk.edu.hk/ttwong/papers/asc/asc.html) for triangulation, Qslim (http://mgarland.org/software/qslim.html) for mesh coarsening, and Showmesh for smoothing and topology correction. Functions written in MATLAB drive this process and also do local mesh refinement. The aim of local mesh refinement is to make sure that the distance between meshes is not too small compared with edge length of the neighboring elements. For this purpose, the elements with long edges are refined if the edge length is larger than the local distance of two neighboring meshes multiplied by the user specified LMR ratio. It is suggested to apply LMR with a ratio of 2.

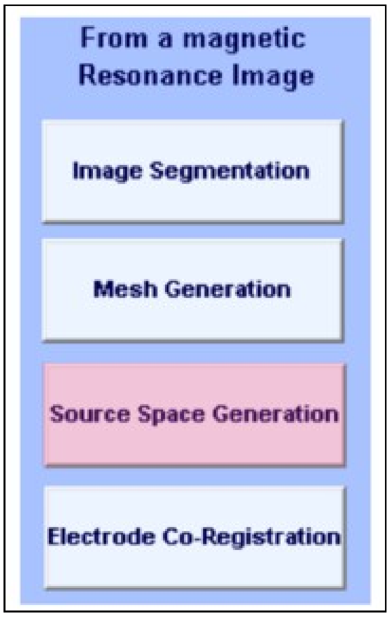
During mesh generation the status of the program is written at the bottom of the window and a progress bar shows the progress of the program.

2.3

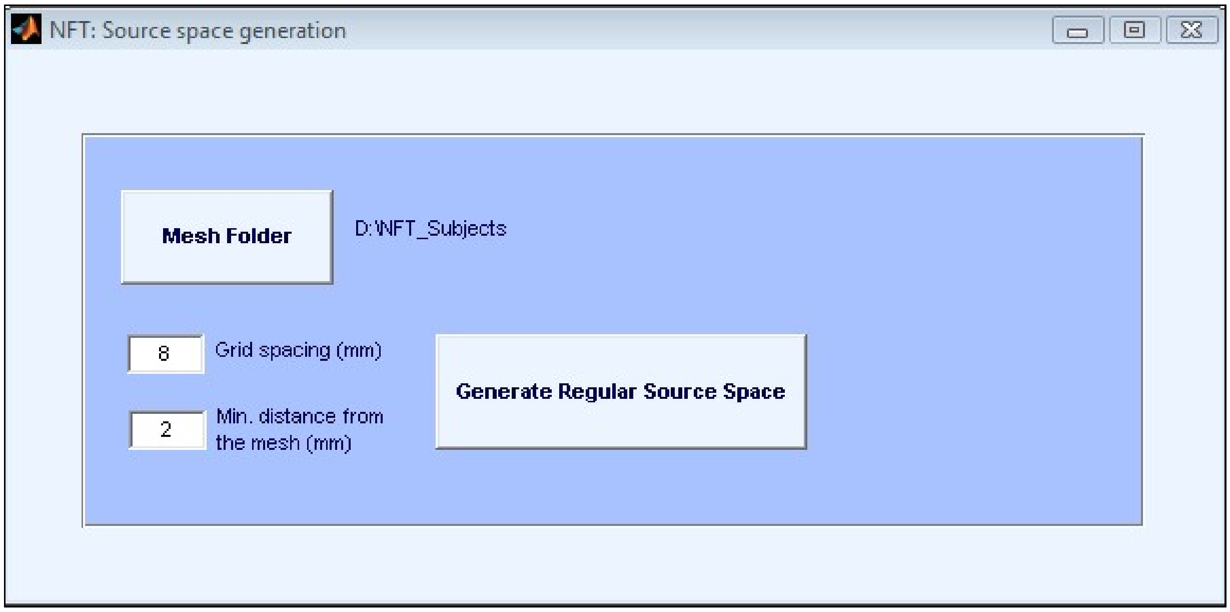
Source space is a set of dipole sources placed within the brain volume. The source spaces are used to generate Lead Field Matrices (LFM) which is a matrix that maps dipole source strengths to electrode potentials.

The Forward Modeling Toolbox contains an option to generate a simple source space consisting of a regular grid. The grid is generated by placing three orthogonal dipoles at each grid location inside the brain volume. The user inputs are the spacing between the dipoles and the minimum distance of a dipole to brain mesh. The spacing determines the minimum distance between two dipoles. The default value of spacing is 8 mm, and the minimum distance of a dipole to brain mesh is 2 mm. These default parameters result in about 6000-7000 dipoles for an average adult human brain. The user interface of this module is shown in Figure 9. The output file is saved in the Output folder set in the Main window as sourcespace.dip in ascii format. It is a matrix of number of dipoles by 6, in each row the x, y, z position and direction of dipoles are given.

A LFM using a regular grid source space can be used in single dipole parametric inverse problem solution to find a coarse estimate of the dipole position.



.....



D:\NFT_Subjects

8 Grid spacing (mm)

2 Min. distance from the mesh (mm)

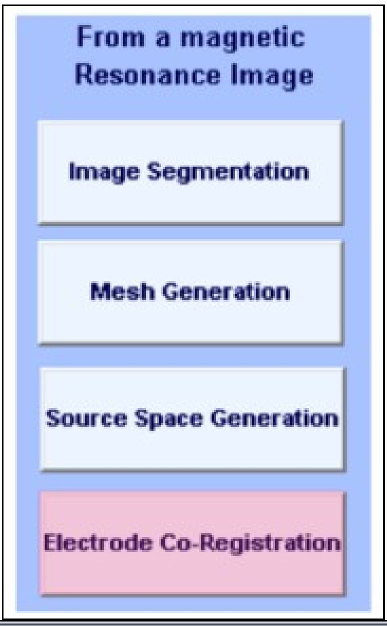
2.4

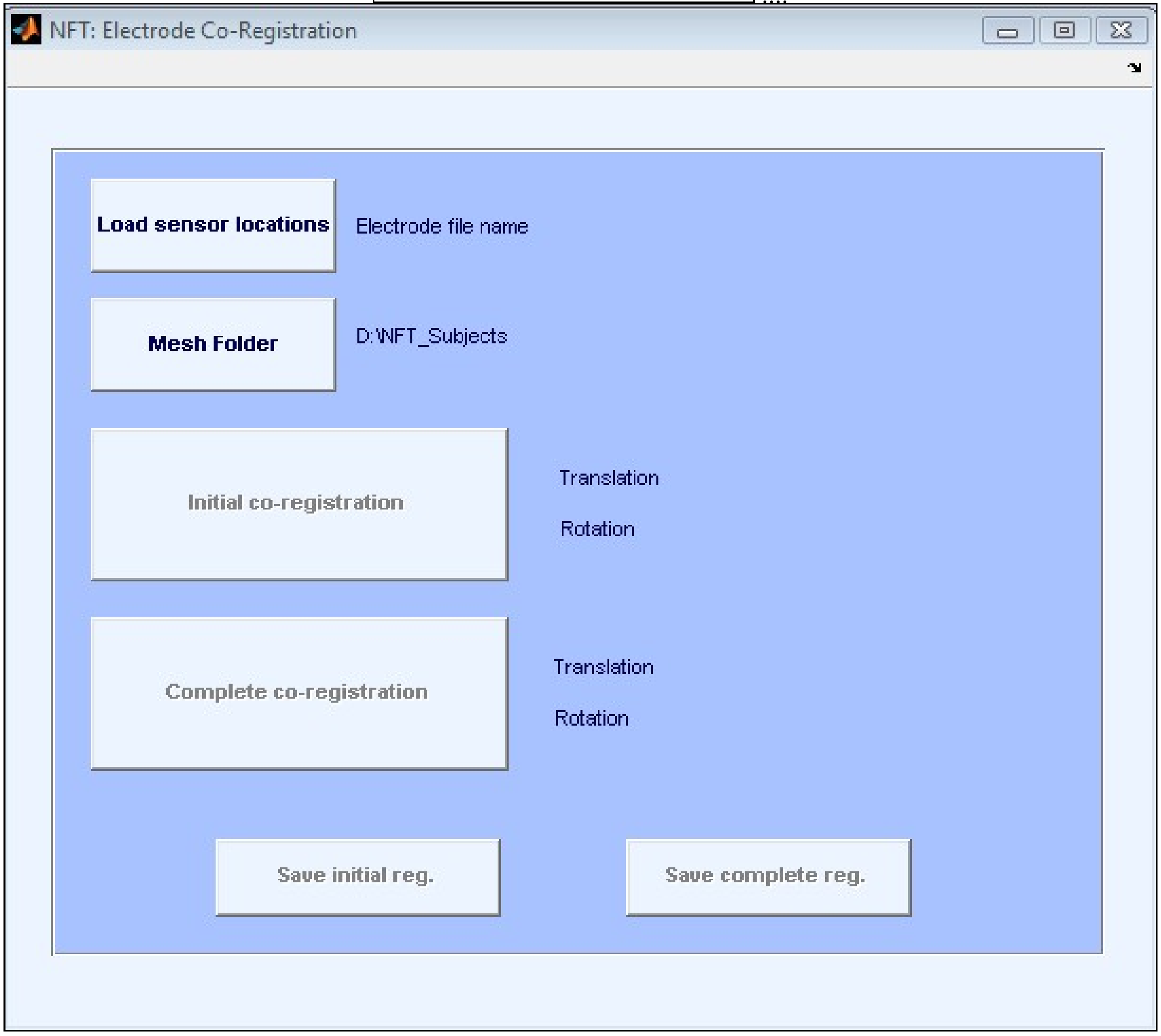
The BEM mesh is generated from the 3D MR volume, and uses the same coordinate system as the volume. When working with EEG recordings, the electrode coordinates, measured by a digitizer, must be mapped to the mesh coordinates. This step is called the co-registration of electrode locations.

The input to the Electrode co-registration module is the electrode locations. The scalp mesh of the subject is loaded automatically, and the electrodes are co-registered to the scalp mesh. The co-registration is done in two steps. First the user manually co-registers the sensors pressing the Initial co-registration button. This starts EEGLAB?s co-registration function and a coarse registration is done to bring the sensors to the mesh coordinate system. The second step is the Complete co-registration. This step starts from the initial co-registration and automatically finds the best translation and rotation parameters by minimizing the total distance between the sensors and scalp surface.

The interface of co-registration is shown in Figure 10. At the end of each registration step, a figure pops up to show the registered electrodes on the scalp surface. It is possible to save either the initial or the complete registration. The outputs of the program are registered electrodes and index of the electrodes in the scalp mesh region.

Note that the outputs of segmentation, mesh generation, and source space are subject specific. The Subject Name is used in output files for these stages. On the other hand, the electrodes must be registered each time the electrode positions change. Therefore, the co-registration output is specific to a session. The result of electrode co-registration is saved as Session_Name_Subject_Name_headsensors.sens in ASCII format.


From a magnetic Resonance Image
Image Segmentation
Mesh Generation
Source Space Generation
Electrode Co-Registration


NFT: Electrode Co-Registration
Load sensor locations
Electrode file name
Mesh Folder
D:\NFT_Subjects
Initial co-registration
Translation
Rotation
Complete co-registration
Translation
Rotation
Save initial reg.
Save complete reg.

2.5

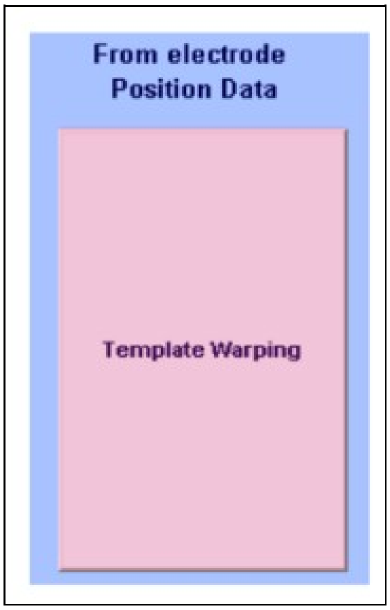
When the MR images of the subject is not available, a frequently used approach is to use a template head mesh, and map the electrodes to this template for source localization. The MNI brain, which is created by the Montreal Neurological Institute (MNI) by averaging the head MRIs of 305 normal subjects, is frequently used for this purpose.

An alternative approach suggested by Darvas et al [1] is to warp a template mesh to fit the sensor locations. The toolbox implements this functionality to generate subject specific head models when no MR images are available. This results in more realistic head models compared with using a template mesh, and mapping electrodes to it. The template model that is used in this toolbox is a 3-layer BEM mesh extracted from the MNI brain.

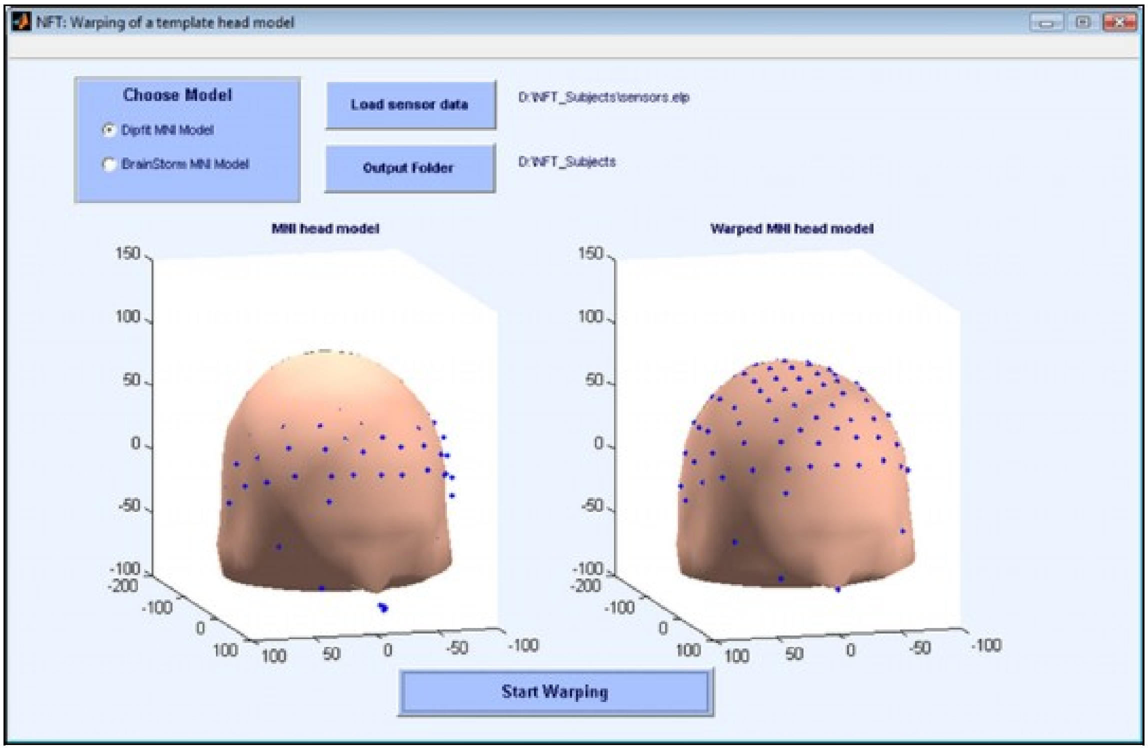
The warping is computed based on fiducials: the nasion and left and right preauricular points. Using these 3 points, another point is calculated on the top of the head on both the template model and subject?s electrode locations. Using these 4 points, the sensor locations and head model are brought into same coordinate system. After this initial co-registration, 19 landmarks on both the head model and sensors are located. These landmarks are used to find the warping parameters. The warping method is a non-rigid thin plate spline method. After finding the warping for scalp, all the surfaces and the source space are warped using the same warping parameters.

The inputs of the warping module are the fiducials and the electrode locations (obtained from a digitizer). The outputs are the warped mesh, warped source space, indices of the electrodes on the mesh, fitted electrode locations and the warping parameters in case a user wants to warp back the localized sources to the template model. Note that the number of warped electrodes may be lower, since the MNI head is not a whole head model, and some electrodes may fall out of the template mesh.

In Figure 11 the interface for warping module is shown.



.....



References

[1] F. Darvas, J.J. Ermer, J.C. Mosher, R.M. Leahy, Generic head models for atlas-based EEG source analysis, Human Brain Mapping, vol. 27(2), 2005, pp 129-143.

Getting Started with NFT NFT Wiki Home Forward Model Generation

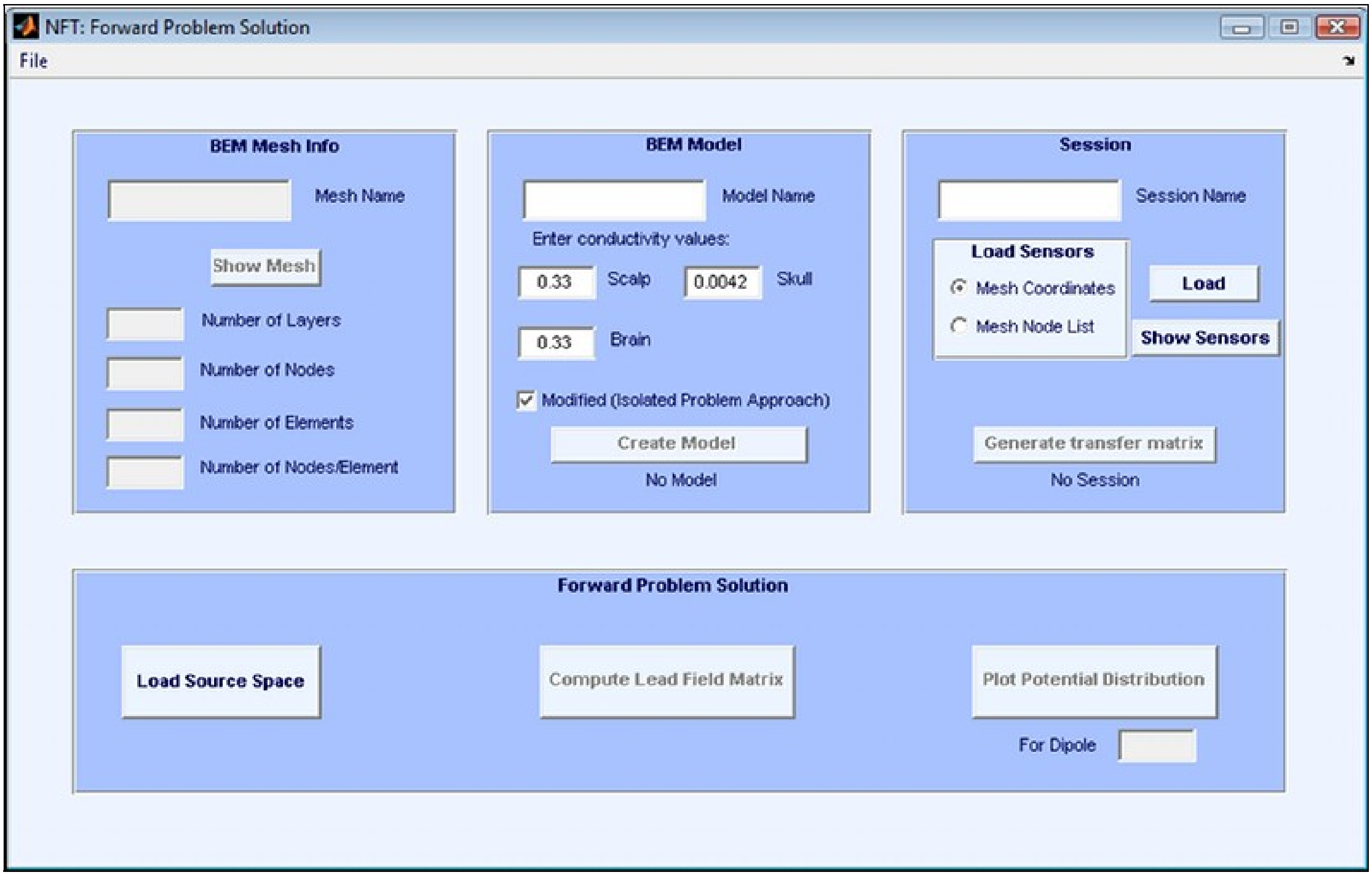
3 Chapter 03: Forward Model Generation

3.1

The boundary element method (BEM) is a numerical computational technique for solving partial differential equations. In electro-magnetic source imaging (EMSI) of brain activity, it is used to solve the forward problem using realistic head models. When using BEM in head modeling, the head is assumed to be composed of uniform conductivity regions (i.e., scalp, skull, brain etc.) and the tissue boundaries are represented by triangular surface elements. The details of the BEM implementation used in this toolbox may be found in [2] and [3]. This section describes the BEM Toolbox.

- Press ?Forward Model Generation? on main menu.
- Load a mesh from file. It is also possible to load a model or a session if you have created them previously.
- Enter model name and conductivity values and press on ?Create Model? button.
- When the model is created, enter session name and load sensors. Sensors may be loaded from a list of nodes (.dat file) or from mesh coordinates (.sens file).
- Press on ?Generate Transfer Matrix?.
- Load source space from .dip file. It should be a number of dipoles by 6 matrix, giving the location and orientation of the dipoles.
- Press on ?Compute Lead Field Matrix?.
- You may plot potential distribution for the nth dipole.

The potentials are save under ?session_name?_LFM as a Matlab .mat file. The user interface is shown in Figure 12.



A forward problem solution using BEM consists of the following steps:

1. Compute the BEM matrices for a given head model.
2. Compute the transfer matrix for a given set of electrodes.
3. Compute the source vector for a given set of dipoles.
4. Obtain the electrode potentials due to the dipole activity.

The BEM solver that is interfaced from MATLAB to compute the BEM matrices is described below. Then, the data structures and functions used by the BEM toolbox are described.

3.2

The BEM solver is written in C++ and is an executable program that is started from MATLAB with correct parameters to compute and save BEM matrices. The solver is called transparently from the MATLAB and need not be called explicitly by the user of the toolbox. However a familiarity with basic options and the program outputs would be useful.

```
$ bem_matrix

        Generate BEM coefficient matrices, including inner matrices for IPA
        Usage: bem_matrix [-f matname] [-m magsens] [-o mod] meshname [s=sig ...]
                -f: save matrices to matname.[cdi]mat (default meshname)
                -m: magnetic sensor file name
                -o: interface to use with modified equations
                (1: outer, 0: to disable)
        s=sig: s:region (1: outer), sig: conductivity
                (default conductivity: 0.2 for all regions)
```

As can be seen from the usage text, the bem_matrix application can generate and save BEM matrices for computing potential and magnetic fields. It can use the isolated problem approach (IPA) to compansate for the loss of accuracy due to the low conductivity of the skull.

3.3

When IPA is not used, only the BEM Coefficient matrix is generated for EEG. It?s saved with the extension .cmt. If IPA is applied, two more matrices are generated and saved: .dmt and .imt.

In addition to the matrices created by the BEM solver, an additional matrix is the inverse of the Inner Coefficient Matrix. This matrix is the inverse of the imt matrix. It is inverted using the builtin MATLAB function inv() and saved on disk using the extension .iinv.

3.4

The transfer matrix makes it possible to have very fast forward problem solutions. It is computed by inverting the selected columns of the coefficient matrix. In the toolbox, the inversion is done in MATLAB using the GMRES iterative solver. The resulting matrix is saved to the disk using the filename extension .tmte.

Since the matrix generation can take a long time for large meshes, an on disk copy is always present so that future computations can be made without generating the matrices from scratch. This is true for all the BEM and transfer matrices generated by the solver and by MATLAB. The matrices are loaded into MATLAB as needed, and can be cleared by the user as necessary to save memory. An additional advantage of this approach is that the matrices can be created externally (manually, at an other computer, etc.) and used by the toolbox.

3.5

The BEM toolbox functions are implemented on three main data structures which represent information available at different stages of the forward problem solution procedure. The structures correspond to the BEM Mesh, the Head Model which is a combination of the mesh, conductivities and solver parameters, and a Session which specifies the sensor (electrode) positions used to acquire the data. The structures are described below:

3.5.1

Mesh structure contains coordinates and connectivity and boundary information from the mesh file.

3.5.2

The model structure includes the mesh structure and conductivity values for the mesh tissue classes and the index of the modified boundary, for use with IPA. If the index is set for smaller than 1, then IPA is not applied. If, for example, it?s set for 3 and there are 4 tissue classes, then 3rd and the 4th layers are the inner layers and the RHS vector is modified.

Furthermore, as the model matrices (.cmt, .dmt, .imt and .iinv) are loaded, they are stored inside the model structure as fields of the same name. The name of the model is used as the base filename when loading the model matrices.

3.5.3

The session structure represents a ?recording session?. It includes the model structure and a matrix relating the positions of the sensors on the scalp to the nodes of the mesh (Smatrix). Each electrode is a weighted sum of the nodes of the element. The weights are determined by the element shape functions. The format of the Smatrix is as follows: [electrode_index node_index weight]. The rows of the matrix must be sorted by electrode_index and there can be more than one row with a given electrode index.

When the transfer matrix (.tmte) is loaded, it is stored inside the session structure. The name of the session is used as the base filename when saving and loading the transfer matrix.

References:

[2] Z. Akalin Acar, N.G. Gencer, An advanced BEM implementation for the forward problem of Electro-magnetic source imaging, Physics in Med. and Biol., vol. 49(5), 2004, pp 5011-28.

[3] N.G. Gencer, Z. Akalin Acar, Use of the isolated problem approach for multi-compartment BEM models of electro-magnetic source imaging, Physics in Med. and Biol., vol. 50, 2005, pp 3007-22.

---

4 Chapter 04: NFT Examples

Forward Model Generation NFT Wiki Home NFT Commands and Functions

In this section, two head modeling examples are presented. Both of these examples use the same subject. The first example generates a realistic head model using the MR image of the subject. The second example warps the template head model to 141 digitized electrode locations. Mesh generation and electrode registration results are given for both of these examples. The computational cost of each modeling stage and sizes of the resulting output files are also given.

4.1

For the first example a four layer mesh is generated for the subject through segmentation and mesh generation steps. The mesh consists of scalp, skull, csf and brain layers for a total of 16016 nodes and 32024 elements. The individual layers can be seen in Figure 13.

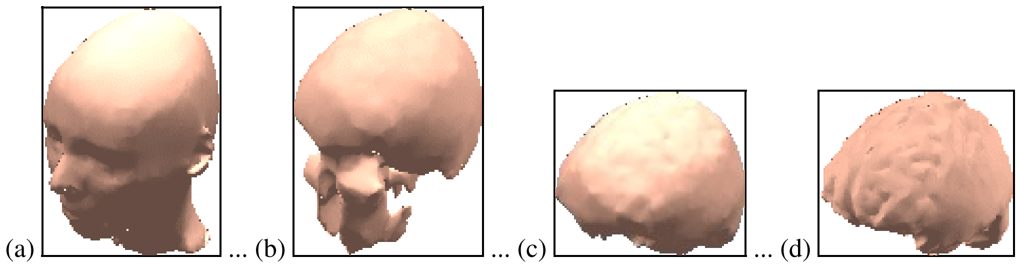
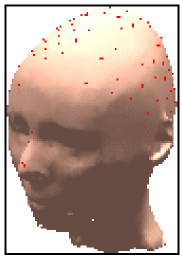


Figure 13: BEM model of the scalp, skull, csf and the brain obtained from an MR image. (a) scalp mesh, (b) skull mesh, (c) CSF mesh, (d) brain mesh.

After mesh generation, the electrodes and the realistic mesh is co-registered. The result of co-registration can be seed in Figure 14.



The second example assumes that the only available subject data is the 141 digitized electrode locations. For warping the template MNI mesh is used, which has three layers and 3000 nodes and 5988 elements. This is the standard mesh that is also used by other BEM solvers in the literature. The results of warping can be seen in Figure 15.

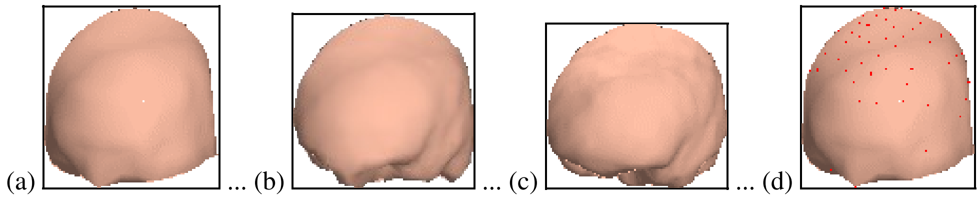


Figure 15: BEM model of the scalp, skull, the brain obtained by warping a template head model to electrode locations. (a) scalp mesh, (b) skull mesh, (c) brain mesh, (d) electrode locations.

Note that the realistic model, and the warped model are two different models for the same experiment. Since the MNI head only contains the half of the head above the mouth, some electrodes had to be discarded. While the realistic model represents the real geometry of the head much better than the warped model, the warped model itself is an improvement over the template MNI head itself.

4.2

The computational cost of using a realistic head model is related to the size of the BEM matrices, which depend on the mesh. The aim of this section is to give an idea about how long different stages of the head modeling and forward problem solution takes.

The realistic model generated using MR image consists of 4 layers and has 16016 number of nodes, and 32024 number of faces in total. Local mesh refinement is done using LMR ratio of 2. The number of faces for each surface is as follows: Scalp:6944, Skull:7084, Csf: 9298, Brain:8698 elements.

Table 1 shows the computation times for realistic head modeling and forward model generation when head model is obtained using MR images. Table 2 shows the computation times for forward model generation when the head model is obtained by warping a template head model. Warping of a template head model and source space generation takes only seconds, therefore, these are not given in the tables. The computations are done on a 64-bit Opteron processor.

Table 1: Computational complexity for the realistic model

Process	Time
Segmentation	25 minutes
Mesh Generation	38 minutes
Co-registration	25 minutes
Generation of BEM matrices (16016 nodes)	2 hours
Calculation of transfer matrix (141 sensors)	3.2 hours
Calculation of Lead Field Matrix (6075 dipoles)	1 hour

Table 2: Computational complexity for the warped model

Process	Time
Generation of BEM matrices (6006 nodes)	19 minutes
Calculation of transfer matrix (135 sensors)	15 minutes
Calculation of Lead Field Matrix (10131 dipoles)	30 minutes

The transfer matrix computation and lead-field generation steps may be executed on multiple processors if the MATLAB Parallel Processing Toolbox is available. We have measured a 2.6x speed-up by generating the transfer matrix on a quad-core instead of a single core processor.

4.3

The toolbox uses the Subject folder to save the generated meshes and matrices. The names of the output files are derived from the subject and session names. This section lists the contents of output folders and size of the files for the two examples discussed above.

Table 3 shows the contents of the output folder when Subject Name is SubjectA and session name is Session1 for the example given in Table 1. Table 4 shows for the case given in Table 2, when the subject name is entered as SubjectB and session name as Session1.

Table 3: Output folder contents and file sizes for the model generated from an MR image.

File	Size
SubjectA_segments.mat	0.4 MB
SubjectA_filtered.mat	84 MB
SubjectA.bei	67 bytes
SubjectA.bec	1.2 MB
SubjectA.bee	0.7 MB
SubjectA.model	473 bytes
SubjectA.cmt	2.9 GB

SubjectA.dmt	844 MB
SubjectA.imt	939 MB
sourcespace.dip	581 KB
Session1_SubjectA_headsensors.sens	6.9 KB
Session1_SubjectA_sensorindex.mat	2.2 KB
Session1.session	4.8 KB
Session1.tmte	53.9 MB
Session1_LFM.mat	6.3 MB

Table 4: Output folder contents and file sizes for the model generated warping a template head model to electrode locations.

File	Size
SubjectB.bei	52 bytes
SubjectB.bec	381 KB
SubjectB.bee	240 KB
SubjectB_warping	2.3 KB
SubjectB.model	381 bytes
SubjectB.cmt	432.1 MB
SubjectB.dmt	136.6 MB
SubjectB.imt	46.3 MB
sourcespace.dip	959 KB
Session1_SubjectB_headsensors.sens	6.4 KB
Session1_SubjectB_sensorindex.mat	2.1 KB
Session1.session	7.6 KB
Session1.tmte	32.7 MB
Session1_LFM.mat	16.9 MB

Forward Model Generation   NFT Wiki Home   NFT Toolbox Commands and Functions

5 Chapter 05: NFT Commands and Functions

NFT Examples NFT Wiki Home BEM Mesh Format

This section summarizes the MATLAB commands and data structures used for each stage of head modeling using the NFT toolbox. The function reference can be found in Appendix B

5.1

The toolbox functions have names all lowercase, words separated by underscore. The GUI function names start with capital letters. The user interface functions all begin with the name of the module, such as bem_, mesh_, segm_, warping_ while utility functions are prefixed by util, for instance BEM utility functions start with utilbem_. All functions have help sections describing the usage, inputs and outputs, compatible with the help2html() conventions. They also include a license block.

In user interface functions (bem_, etc.) the input arguments are validated before use. The utility functions, which are mostly used internally do minimal validation.

5.2

The main user interface for the segmentation is initiated using Segmentation() command which opens up the GUI for segmentation.

While most of the BEM matrix generation functionality can be initiated from the GUI interface, each operation can also be performed through matlab functions. After loading the MR image the following functions are called respectively: segm_aniso_filtering() , segm_scalp() , segm_brain() , segm_outer_skull() , segm_inner_skull() , segm_final_skull() .

5.3

The main user interface for mesh generation is initiated using Mesh_generation() command which opens up the GUI for mesh generation. The function loads the segmentation and generates meshes for scalp, skull, CSF and brain. If the user wants to refine the meshes locally, mesh_local_refinement() function is called. The topology of the generated meshes are checked by mesh_final_correction() , and finally, the total head mesh is written in the format described in A using the function mesh_read_write() .

5.4

The main user interface for the co-registration of electrode locations with MR images is initiated using Coregistration() command which opens up the GUI for co-registration.

5.5

The main user interface for the warping of a template head model is initiated using Warping_mesh() command which opens up the GUI for warping functions. After loading the electrode locations the MNI mesh is loaded. Using warping_main_function() function the warping parameters and the warped mesh are calculated.

5.6

The main user interface for the forward model generation is initiated using Forward_Problem_Solution() command. While most of the BEM matrix generation functionality can be initiated from the GUI interface, each operation also be performed through matlab functions.

The functions used by the BEM module are used for creating the BEM structures, running the solver to generate the model matrices, and solving for single or multiple dipoles.

The state of the forward solution is stored in the structures, and no global variables are used by the toolbox. Since the contents of the structure arguments may change during a function call, most interface functions return the structure so that the changes can be preserved (MATLAB has no OUT arguments).

5.6.1

A set of mesh files can be loaded with the bem_load_mesh() function which returns a mesh structure.

5.6.2

The model structure which combines the mesh, conductivities and solver IPA parameters is obtained using the bem_create_model() function. Once the model structure is obtained, it is possible to invoke the solver using the bem_generate_eeg_matrices() function to generate the BEM matrices. Individual matrices can be loaded into the model structure using the bem_load_model_matrix() function.

5.6.3

The session structure is used for solving the forward problem at a given set of sensors. The structure is created using bem_create_session() from a model and a list of sensors. The list of sensors can be generated from a list of nodes using the bem_smatrix_from_nodes() function.

The transfer matrix for the sensors specified in the session can be generated using the bem_generate_eeg_transfer_matrix() function which computes and saves the transfer matrix. The computed transfer matrix can be loaded into the session structure using the bem_load_transfer_matrix() function.

There are two functions for obtaining forward solutions. The bem_solve_dipoles_eeg() function computes the sensor potentials due to the activation of a number of dipoles. The bem_solve_lfm_eeg() function is suitable for generating a Lead Field Matrix since it returns a matrix of single dipole solutions.

5.6.4

The utility functions are used internally by the functions described above.

The external user configuration can be returned using the NFT_get_config() function. This m-file can be edited manually to specify run-time options for the toolbox. User interface functions call this function to get configuration variables as needed.

The utilbem_compute_cond() and utilbem_compute_indices() functions compute conductivity and index information from the mesh. These functions are called by bem_create_model() and the results are stored in the model structure.

There are two utility functions for computing source (right-hand-side) vectors. utilbem_multilayer_rhs() is used for IPA and utilbem_pot_unbound() is used without IPA.

NFT Examples | NFT Wiki Home | BEM Mesh Format

6 NFT Appendix A: BEM Mesh Format

6.1

The generated BEM mesh is stored on disk as a set of three files: an element file, a coordinate file, and an information file. All three mesh files have the same base name, with the file extension specifying the file type. The file extensions are .bee for the element file, .bec for the coordinate file and .bei for the information file. All the files are ASCII text files for easier processing and portability.

The information file (.bei) defines the high level properties of the mesh. Each mesh consist of one or more boundaries. Since the boundaries separate tissues, each boundary has an inside and outside tissue type. The first row of the information file contains information about the mesh structure. The entries of the first row are the number of boundaries, the number of nodes, the number of elements, and the number of nodes per element respectively. For linear meshes there are 3 nodes per element and for quadratic meshes there are 6 nodes per element. The following rows of the information file define the boundary information. Since an element can be a part of only one boundary, the elements of the mesh are grouped according to the boundary, and from outside to inside. Therefore, each boundary is a consecutive group of elements. For the boundary rows, the first column is the boundary index. Second column gives the number of elements in the boundary. The third and fourth columns represent the inner and outer tissue class of the boundary.

The tissue class is an integer representing a tissue. This number is defined per mesh, there is no global assignment of tissue classes at the moment. The purpose of tissue class is to uniquely define the various tissues that are represented by the mesh. Since different tissues may have same or similar conductivities, using a tissue class identifier provides a better distinction. Furthermore, this scheme makes it possible to solve the same mesh geometry using different tissue conductivity values.

The coordinate file (.bec) defines the physical coordinates of the nodes in the BEM mesh. There is one node per row. The first column is the node index, and runs from one to the number of nodes in the mesh. The next three columns represent the x, y, and z coordinates of the node.

The element file (.bee) defines the connectivity of the nodes for each element. The element file defines one element per row. The first column is the element index, and runs from one to the number of elements in the mesh. The next three (linear mesh) or six (quadratic mesh) columns define the node indexes for the element. Note that the order of nodes are important, and define the orientation of the element.

7 NFT Appendix B: Function Reference

BEM Mesh Format NFT Wiki Home Effect of brain-to-skull conductivity ratio estimate

7.1

```
bem_create_model() - Creates a model structure combining a mesh,
                     conductivity information and BEM parameters.

Usage:
  >> model = bem_create_model(name, mesh, cond, mod);

Inputs:
  name - model name, used as a base filename for matrices
  mesh - mesh structure obtained from BEM_LOAD_MESH
  cond - conductivity values for mesh tissue classes
         vector, starting from first tissue,
         0 is reserved for air
  mod  - index of the modified boundary, for use with IPA.
         If mod <= 0, IPA is not used.

Outputs:
  model - model structure with the following fields.

Model Structure:
  name      - name of the model
  mesh      - mesh structure
  cond      - conductivity vector
  mod       - modified boundary information
  node_cond - Average conductivity around a node

Optional Fields:
  ind_mod       - node indices of the modified boundary
  ind_imesh     - node indices of the inner mesh
  ind_imesh_mod - node indices of the modified boundary
                  relative to the inner mesh
```

7.2

```
bem_create_session() - Creates a session structure combining a model,
      and sensor data. The session structure contains a model for a
      complete head and electrode ?recording session?. Data recorded
      using the same set of sensor locations is considered
      to be in the same session.

Usage:
      >> session = bem_create_session(name, model, Smatrix);

Inputs:
  name    - session name, used as a base filename for matrices
  model   - model structure obtained from bem_create_model().
  Smatrix - matrix that defines EEG electrodes in terms of
    the BEM mesh. Each electrode is a weighted sum of the nodes
    of the element. The weights are determined by the element
    shape functions. The format of the Smatrix is as follows:
         [electrode_index node_index weight]
    The rows of the matrix must be sorted by electrode_index
    and there can be more than one row with a given electrode
    index. The Smatrix can be constructed using one of these:
    bem_smatrix_from_nodes() and bem_smatrix_from_points().

Outputs:
  session - session structure with the fields defined in the next section

Session Structure:
  name    - name of the session
  model   - model structure
  Smatrix - EEG sensor information matrix.
  num_electrodes - number of EEG sensors
```

7.3

```
bem_generate_eeg_matrices() - Generates BEM matrices for a BEM model.

Usage:
      >> bem_generate_eeg_matrices(model);

Inputs:
  model - model structure generated by BEM_CREATE_MODEL

Notes: The matrices are created using the name given in the model structure
      The following matrices are created on disk, and can be read
      using the bem_load_model_matrix() function
      <model.name>.cmt - BEM coefficient matrix
      <model.name>.dmt - band of Coefficient matrix used by IPA
      <model.name>.imt - inner Coefficient Matrix used by IPA
```

7.4

```
bem_generate_eeg_transfer_matrix() - Generates EEG transfer matrix.

Usage:
  >> session = bem_generate_eeg_transfer_matrix(session);

Inputs:
  session - session structure generated by BEM_CREATE_SESSION

Outputs:
  session - session object; since it may be modified when
            loading model matrices.

Notes: The matrix is created using the name given in the session structure
      with "tmte" extension for the EEG transfer matrix. It can be loaded
      using the bem_load_transfer_matrix().
```

7.5

```
nft_get_config() - Returns the toolbox configuration information
      program path names, defaults, etc. Other toolbox functions call this m-file
      to read their configuration.

Outputs:
  conf - config structure.

Config Structures:
  bem_matrix_program - name of the program that creates EEG matrices.
  asc - name of adaptive skeleton climbing program.
  qslim - name of the coarsening program.
  showmesh - name of the correction and smoothing program.
```

7.6

```
bem_load_mesh() - Loads a BEM mesh.

Usage:
  >> mesh = bem_load_mesh(name);

Inputs:
  name - mesh name excluding the extension.

Outputs:
  mesh - mesh structure.

Mesh Structure:
  name  - mesh name
  coord - node coordinates
  elem  - elements (connectivity information)
  num_nodes - number of nodes
  num_elements - number of elements
  num_boundaries - number of boundaries
  num_node_elem - number of nodes per element
  num_class - number of tissue classes
  bnd - boundary information array
       [num_elements inner_tissue_class outer_tissue_class]
```

7.7

```
bem_load_model_matrix() - Loads the BEM matrix with extension ?ext?.
The matrix file name is defined by <model.name>.<ext>
Matrix is placed into the model with a field name ?ext?

Usage:
  >> model = bem_load_model_matrix(model, ext);

Inputs:
  model - model structure generated by bem_create_model().
  ext - describes the matrix type. The following types are defined:
    ?cmt? - BEM Coefficient matrix
    ?dmt? - band of Coefficient matrix used by IPA
    ?imt? - inner Coefficient Matrix used by IPA
    ?iinv? - inverse of the inner coefficient matrix. This matrix is
      computed from ?imt? if the file does not exist.

Outputs:
  model - model structure.
```

7.8

```
bem_load_transfer_matrix() Loads the BEM transfer matrix with extension ?ext?.
      The matrix file name is defined by <session.name>.<ext>.
      Matrix is placed into the session with a field name ?ext?.

Usage:
  >> session = bem_load_transfer_matrix(session, ext);

Inputs:
  session - session structure generated by bem_create_session().
  ext - describes the matrix type. The following type is defined
    ?tmte? - BEM Transfer matrix.

Outputs:
  session - updated session structure.
```

7.9

```
bem_smatrix_from_nodes() - Generates Smatrix from nodes of a mesh.
      Smatrix is the sensor information matrix used in
      bem_create_session(). See bem_create_session() for more information.

Usage:
  >> Smatrix = bem_smatrix_from_nodes(mesh, nodes);

Inputs:
  mesh - mesh structure
  nodes - vector of node indices.

Outputs:
  Smatrix - defines one electrode per node.
```

7.10

```
bem_solve_dipoles_eeg() - Computes the potential arising from the given dipoles
at the sensor locations defined by the session.

Usage:
  >> [pot, session] = bem_solve_dipoles_eeg(session, dipoles);

Inputs:
  session - session structure defining the model and the sensors.
  dipoles - dipole matrix. Each row defines a dipole as follows:
          [x y z px py pz]

Outputs:
   pot - potentials at the sensors due to simultaneous activation of
         the dipoles.
   session - the updated session structure, in case any matrices are
             loaded, modifying the underlying model.
```

7.11

```
bem_solve_lfm_eeg() - Computes the LFM arising from given dipoles
at the sensor locations defined by the session.

Usage:
  >> [pot, session] = bem_solve_lfm_eeg(session, dipoles);

Inputs:
  session - session structure defining the model and the sensors
  dipoles - dipole matrix. Each row defines a dipole as follows:
           [x y z px py pz]

Outputs:
  pot - the LFM at the sensors produced by activation of each dipole,
  session - updated session structure in case any matrices are
            loaded, modifying the underlying model.
```

7.12

```
Coregistration() - Produces the GUI for co-registration of electrode locations
 to the scalp mesh.

Usage:
  >> Coregistration(varargin);

Optional Arguments:
  ?subjectdir? - (string) Output folder. The results will be saved
   in this folder.
  ?subject? - (string) Subject name. The subject-specific output files will start
   with this name.
  ?session? - (string) Session name. The file name for registered electrodes will
   start with this name.
```

7.13

```
Forward_Problem_Solution() - Produces the forward model generation GUI.

Usage:
  >> Forward_Problem_Solution(varargin);

Optional Arguments:
  ?subjectdir? - (string) Output folder. The results will be saved in this folder.
  ?subject? - (string) Subject name. Subject-specific output files will be saved
   starting with this name.
  ?session? - (string) Session name. Session-specific output files will be saved
   starting with this name.
```

7.14

```
Mesh_generation() - Produces the mesh generation GUI.

Usage:
  >> Mesh_generation(varargin);

Optional Arguments:
  ?subjectdir? - (string) Output folder. The results will be saved
   in this folder.
  ?subject? - (string) Subject name. Output files will start with this name.
```

7.15

```
mesh_local_refinement() - Refines the meshes in a given folder. Loads
Scalp.smf, Skull.smf, Csf.smf, Brain.smf meshes in .smf format and saves
them with the same names.

Usage:
  >> mesh_local_refinement(of, nl, ratio_lmr);

Inputs:
  of - mesh folder
  nl - number of layers (3 or 4)
  ratio_lmr - ratio of local edge length to local distance between meshes
```

7.16

```
mesh_final_correction() - Performs mesh correction for Scalp.smf,
Skull.smf, Csf.smf, Brain.smf in a given folder. Saves the meshes with
the same names in .smf format.

Usage:
  >> mesh_final_correction(of, nl);

Inputs:
  of - mesh folder
  nl - number of layers (3 or 4)
```

7.17

```
mesh_read_write() - Reads Scalp, Skull, Csf and Brain meshes in .smf
format in a given folder and writes the total head mesh  bec, bee, bei
format.

Usage:
  >> mesh_read_write(of, mesh_name, nl);

Inputs:
  of - mesh folder
  mesh_name - name of the mesh that will be saved in bec, bee, bei format
  nl - number of layers (3 or 4)
```

7.18

```
Segmentation() - Produces the GUI for running segmentation functions.

Usage:
  >> Segmentation(varargin);

Optional Arguments:
  ?subjectdir? - (string) Output folder. The results will be saved
   in this folder.
  ?subject? - (string) Subject name. The output files will start with this name.
```

7.19

```
segm_aniso_filtering() - Performs anisotropic filtering.

Usage:
  >> [b] = segm_aniso_filtering(out, iter, ts, cond);

Inputs:
  out - input image
  iter - number of iterations
  ts - sampling rate
  cond - diffusion parameter

Outputs:
  b - output image
```

7.20

```
segm_scalp() - Performs scalp segmentation

Usage:
  >> [Sca] = segm_scalp(b);

Inputs:
  b - input image (filtered MR image)

Outputs:
  Sca - scalp mask
```

7.21

```
segm_brain() - Performs brain segmentation

Usage:
  >> [Bra] = segm_brain(b,Sca, sli, WMp,sl, st);

Inputs:
  b - input image (filtered MR image)
  sli - lowest point for cerebellum
  WMp - white matter point
  sl,st - fill level and threshold for watershed segmentation

Outputs:
  Bra - brain mask
```

7.22

```
segm_brain() - Performs outer skull segmentation

Usage:
  >> [Sk_out, X_dark] = segm_outer_skull(b, Sca, Bra, sli_eyes);

Inputs:
  b - input image (filtered MR image)
  Sca - scalp mask
  Bra - brain mask
  sli_eyes - slice of the eyes

Outputs:
  Sk_out - outer skull mask
  X_dark - dark regions of b
```

7.23

```
segm_inner_skull() - Performs inner skull segmentation

Usage:
  >> Sk_in = segm_inner_skull(b, Sk_out, X_dark, Bra);

Inputs:
  b - imput image (filtered MR image)
  Sk_out - outer skull mask
  X_dark - dark regions of b
  Bra - Brain mask

Outputs:
  Sk_in - inner skull mask
```

7.24

```
segm_final_skull() - Corrects scalp and outer skull masks wrt inner
                  skull mask

Usage:
  >> [Sca, Sk_out] = segm_final_skull(Sca, Sk_out, Sk_in, WMp);

Inputs:
  Sca    - scalp mask
  Sk_out - outer skull mask
  Sk_in  - inner skull mask
  WMp    - white matter point

Outputs:
  Sca    - scalp mask
  Sk_out - outer skull mask
```

7.25

```
utilbem_compute_cond() - Computes the average conductivity
    around a node to be used in BEM computations.

Usage:
  >> cond = utilbem_compute_cond(Coord, Elem, layers, sigma)

Inputs:
   Coord  - node coordinate matrix
   Elem   - element connectivity matrix
   layers - mesh boundary information
   sigma  - vector of element conductivities
```

```
Outputs:
   cond - average conductivity for each node (vector)

Note: The average conductivity is normally the average of the inner and outer
   conductivities of the layer that a node belongs to. However, for
   meshes with intersecting boundaries there may be three or more tissues
   around a node. This function also handles this general case.
```

7.26

```
utilbem_compute_indices() - When the Isolated Problem Approach (IPA) is used,
    the BEM equations are modified to reduce the numerical errors due to
    the low conductivity skull layer.  For this purpose an "inner mesh"
    is defined consisting of skull and the inner layers. This function
    computes the node indices corresponding to these layers to be used
    in BEM computations.

Usage:
  >> [indMsh, indMod, indMshMod] = utilbem_compute_indices(Coord, Elem, layers, mod);

Inputs:
   Coord  - node coordinate matrix
   Elem   -  element connectivity matrix
   layers - mesh boundary information
   mod    - index of the boundary modified for IPA.

Outputs:
   indMsh - coordinate indices of inner mesh nodes (vector).
   indMod - coordinate indices of modified boundary nodes (vector).
   indMshMod - Coordinate indices of modified boundary relative to inner
               mesh.
```

7.27

```
utilbem_multilayer_rhs() - Computes the Right-Hand-Side of the
    BEM matrix equation when IPA (Isolated Problem Approach) is
    used. The RHS vector corrected by IPA reduces numerical errors
    in computation due to the low conductivity skull layer.

Usage:
  >> RHS = utilbem_multilayer_rhs(Coord, cond, indMod, indMsh, indMshMod,
   iinv, dmt, s2, s3, dip);

Inputs:
   Coord     - node coordinate matrix
   cond      - average conductivity for each node (vector)
               as computed by utilbem_compute_cond()
   indMod    - coordinate indices of modified boundary nodes (vector).
   indMsh    - coordinate indices of inner mesh nodes (vector).
   indMshMod - coordinate indices of modified boundary relative to inner mesh.
               The indices are computed by utilbem_compute_indices()
   iinv      - inverse inner matrix.
   dmt       - sub coefficient matrix (# of nodes) x (# of mod. boundary nodes).
   s2        - outer conductivity of the modified layer.
   s3        - inner conductivity of the modified layer
   dip       - dipole parameters [x y z px py pz]

Outputs:
   RHS - The right-hand-side of the BEM matrix equation for a given
   dipole.
```

7.28

```
utilbem_pot_unbound() - Computes the unbounded potential at the
    given coordinates due to a single dipole. The result is
    weighted by the average conductivity around the node.

Usage:
  >> pot = utilbem_pot_unbound(Coord, cond, dip)

Inputs:
   Coord - node coordinate matrix
   cond  - average conductivity for each node (vector)
           as computed by utilbem_compute_cond()
   dip   - dipole [x y z px py pz]
```

```
Outputs:
   pot - node potentials for the homogeneous model

Notes: When IPA is not used the weighted potential
   is the  Right-Hand-Side of the BEM matrix equation.
```

7.29

```
Warping_mesh() - Produces the GUI for warping of a template head model
        to measured electrode locations.

Usage:
  >> Warping_mesh(varargin);

Optional Arguments:
  ?subjectdir? - (string) Output folder. The results will be saved
   in this folder.
  ?subject? - (string) Subject name. Subject-specific output files will be saved
   starting with this name.
  ?session? - (string) Session name. Session-specific output files will be saved
   starting with this name.
```

7.30

```
warping_main_function() - Main warping function.

Usage:
  >> [Ptm, ind, Cscalp_w,Cskull_w,CCSF_w,W,A,e,LMm2] =
  warping_main_function(Cscalp, Escalp, Cskull, CCSF,LMm, Fm, Fd, pos);

Inputs:
  Cscalp, Escalp - scalp mesh
  Cskull         - coordinates of skull mesh
  CCSF           - coordinates of CSF mesh
  LMm            - landmarks on the template mesh
  Fm             - fiducials of the template mesh
  Fd             - fiducials from the digitizer data
  pos            - electrode locations

Outputs:
  Ptm      - electrode locations on the mesh
  ind      - index of the electrodes on the mesh
  Cscalp_w - warped scalp mesh coordinates
  Cskull_w - warped skull mesh coordinates
  CCSF_w   - warped CSF mesh coordinates
  W, A, e  - warping transform parameters
  LMm2     - warped landmarks
```

BEM Mesh Format   NFT Wiki Home   Effect of brain-to-skull conductivity ratio estimate

8 NFT Appendix C: Effect of brain-to-skull conductivity ratio estimate

8.1

An important consideration for obtaining accurate forward problem solutions is to correctly model the distribution of conductivity within the head. In the literature, consistent conductivity values have been reported for scalp, brain, and CSF tissues but there has been a huge variation in reported skull conductivity values. This is partly caused by variations in skull conductivity from person to person and throughout the life cycle (Hoekema et al 2003), and partly from use of different measurement/estimation methods (Oostendorp et al 2000). In the 1970's and 80's the brain-to-skull conductivity ratio was reported to be 80 (Rush 1968, Cohen 1983), still a very commonly used ratio in EEG source localization. More recent studies in the last decade have reported this ratio to be as low as 15 (Oostendorp 2000). In a more recent study on epilepsy patients undergoing presurgical evaluation using simultaneous intra-cranial and scalp EEG recordings, the average brain-to-skull conductivity ratio was estimated to be 25 (Lai 2005).

Below, we present some simulation results showing the effects of using incorrect skull conductivity values on equivalent dipole source localization. For this purpose, we solved the forward electrical head model problem using a realistic, subject-specific four-layer BEM model built from a subject?s MR head image using the NFT toolbox (Akalin Acar & Makeig, 2010). We set the forward model (?ground truth?) brain-to-skull conductivity ratio to 25 and then solved the inverse problem using the same realistic head model using the commonly used ratio of 80. This produced equivalent dipole localization errors of up to 2.5 cm (figure below, top row). The positions of the grid of model dipoles were moved towards the scalp surface. On the other hand, (figure below, bottom row) if the brain-to-skull conductivity ratio was mis-estimated to be 15 when solving the inverse problem, the dipoles were localized more towards the center of the brain with localization errors up to 1 cm (Akalin Acar & Makeig, 2012). Therefore, correct modeling of skull conductivity is an important factor for EEG source localization.

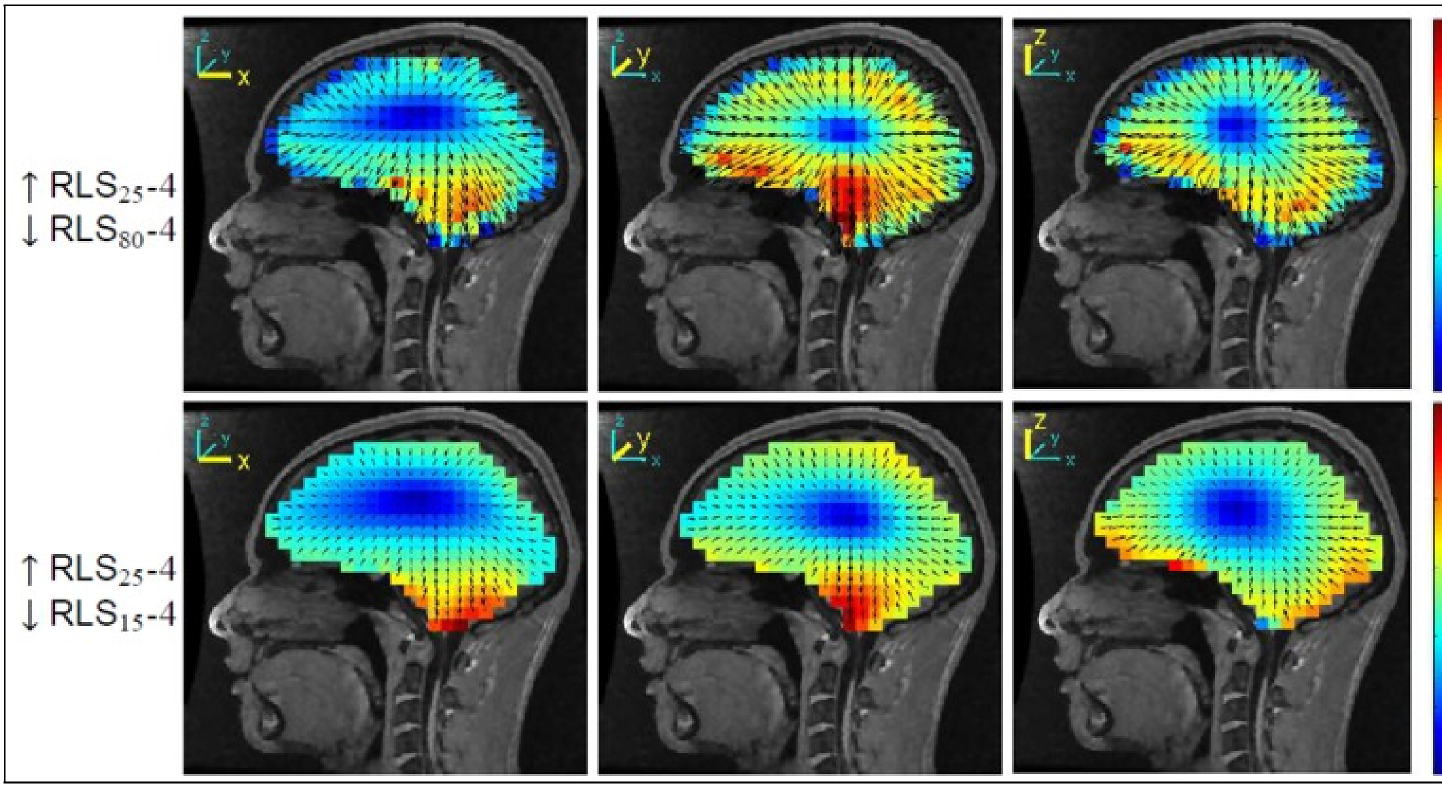


Figure 1. Equivalent dipole source localization error directions (arrows) and magnitudes (colors) for a 4-layer realistic BEM head model when the brain-to-skull conductivity ratio was estimated to be 80 as opposed to the actual simulated

forward model value of 25 (top row) and as 15 (as opposed to 25) (bottom row). The source space was a regular Cartesian grid of single equivalent dipole sources with 8-mm spacing filling the brain volume. The three columns show the errors when the equivalent dipole sources are oriented in x-, y-, and z-directions, respectively.

BEM Mesh Format | NFT Wiki Home | SCCN Wiki Home