Exercise Sheet 3: Head Modeling

In [1]:
```
import numpy as np
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from scipy import constants
import bci_minitoolbox as bci
from scipy import ndimage

from matplotlib import cm
```

Task 1: T1-MRI scan (1 point)

In Magnetic Resonance Imaging (MRI), a T1-scan is often used as a structural scan of the anatomy. This can be used to extract the individual geometry of the head of a subject. T1 is a certain longitudinal relaxation time of the nuclear spin of atoms and is dependent on the individual binding of the atoms and the aggregation state of the matter. Measuring the remaining resonance after a certain time can lead to insights on the composition of the matter under study. The result is a spatially resololved intensity value that can be treated like any other 3-dimesnional gray-scale image.

The file T1.npy contains the T1-scan of a human subject's head. In this image, the first dimension represents the direction from left to right pre-auricular point (a certain point above the ear channel on the ear conch). This direction is also called lateral. The second dimension is defined by the direction from the center between the two pre-auricular points (called the intra-auricular point) to the nasal point. This dimension is also called frontal. The third and last dimension is perpendicular to the other two dimensions oriented towards the top of the head. It's called the superior direction.

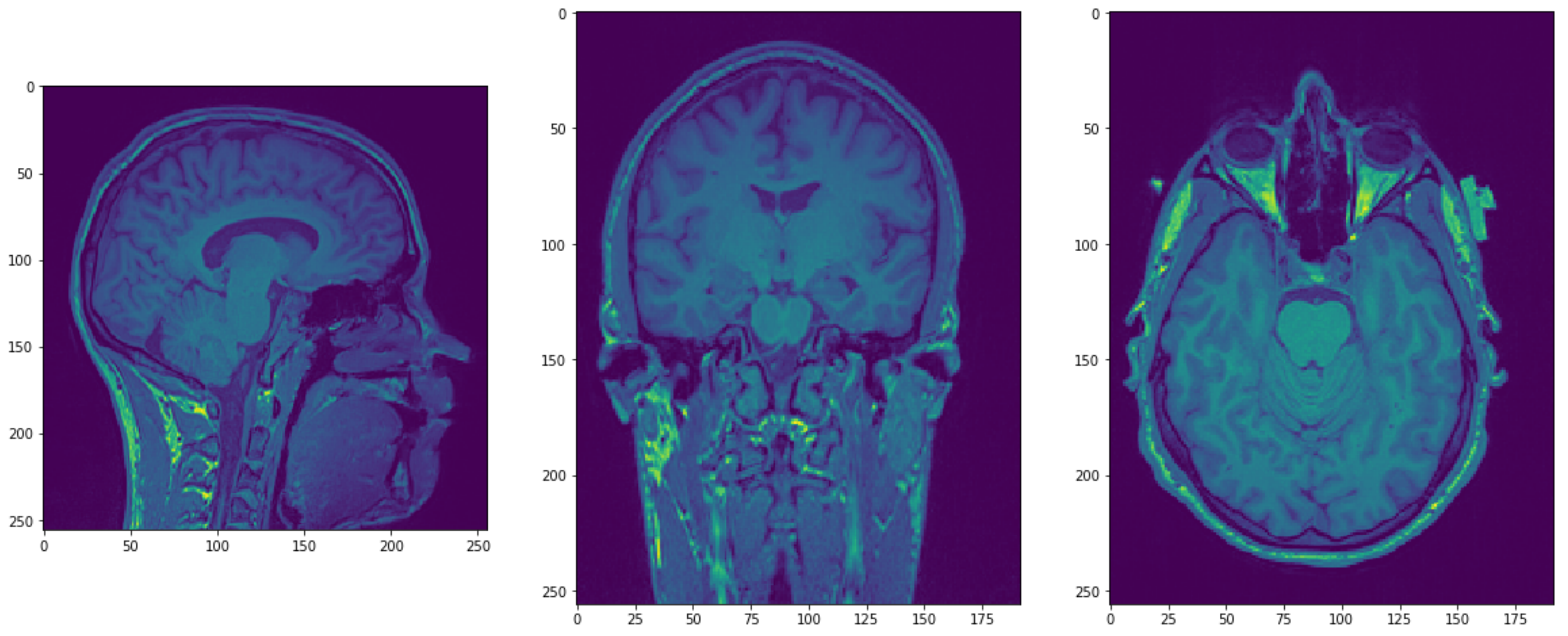
Tasks:

a) Use the command *imshow* from the Matplotlib to plot 3 images in each mid-plane slice of the MRI.

b) Plot a histogram over the intensity values (function *hist*) in a linear and a logarithmic scale for the frequency. Also make a plot zooming in from 0 to 200 in the logarithmic scale (making 3 histograms in total).

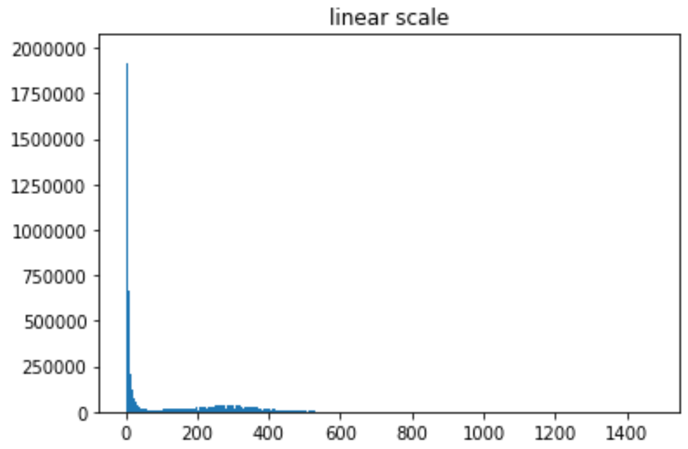
What can you say about the intensity values of different tissue types? Are they well separated?

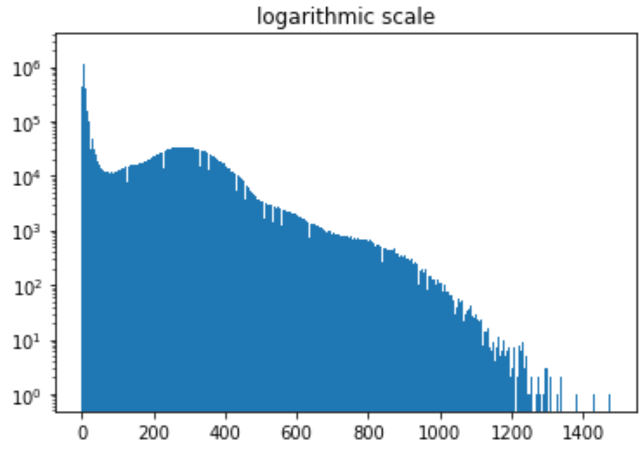
```
In [2]: mri=np.load('files/T1.npy');
        lateral = ndimage.rotate(mri[96,:,:], 90)
        frontal = ndimage.rotate(mri[:,128,:], 90)
        superior = ndimage.rotate(mri[:,:,128], 90)

        fig, ax =plt.subplots(1,3,figsize=(20,20))
        ax[0].imshow(lateral)
        ax[1].imshow(frontal)
        ax[2].imshow(superior)

Out[2]: <matplotlib.image.AxesImage at 0x7fd3c58c9630>
```

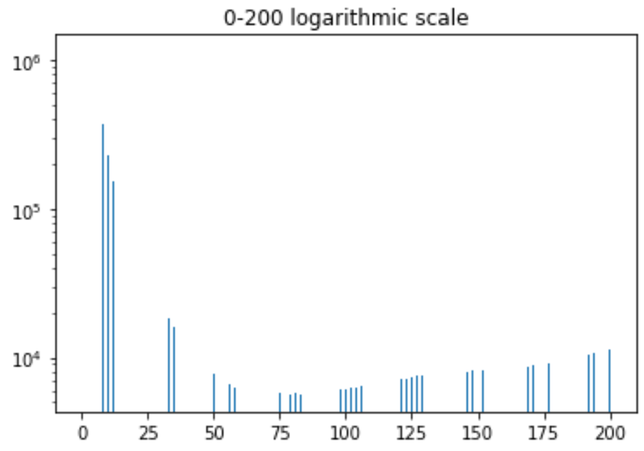


In [3]:
```
plt.hist(mri.ravel(),bins='auto')
plt.title('linear scale')
plt.figure()
plt.hist(mri.ravel(),bins='auto',log=True)
plt.title('logarithmic scale')
plt.figure()
plt.hist(mri.ravel(),range=(0,200),bins='auto',log=True)
plt.title('0-200 logarithmic scale')
```

```
Out[3]: Text(0.5,1,'0-200 logarithmic scale')
```







What can you say about the intensity values of different tissue types? Are they well separated?

- No, they are not so very well seepearted.

Task 2: Segmentation by thresholds (1 point)

If we want to segment the MRI into different tissue types, we need to find out, which intensity corresponds to which tissue type. You can look at the spatial distribution of the intensity value ranges by producing clipped versions of the MRI. Therefore, you have to set a certain range of intensity values to 0 and the rest to 1.

Tasks:

Pick out a peak of your choice (can also be minor peaks) in the logarithmic histogram of the T1-intensities from task 1, define a lower and upper margin flanking that peak and display clipped images of

a) values below the peak

b) all values within this peak

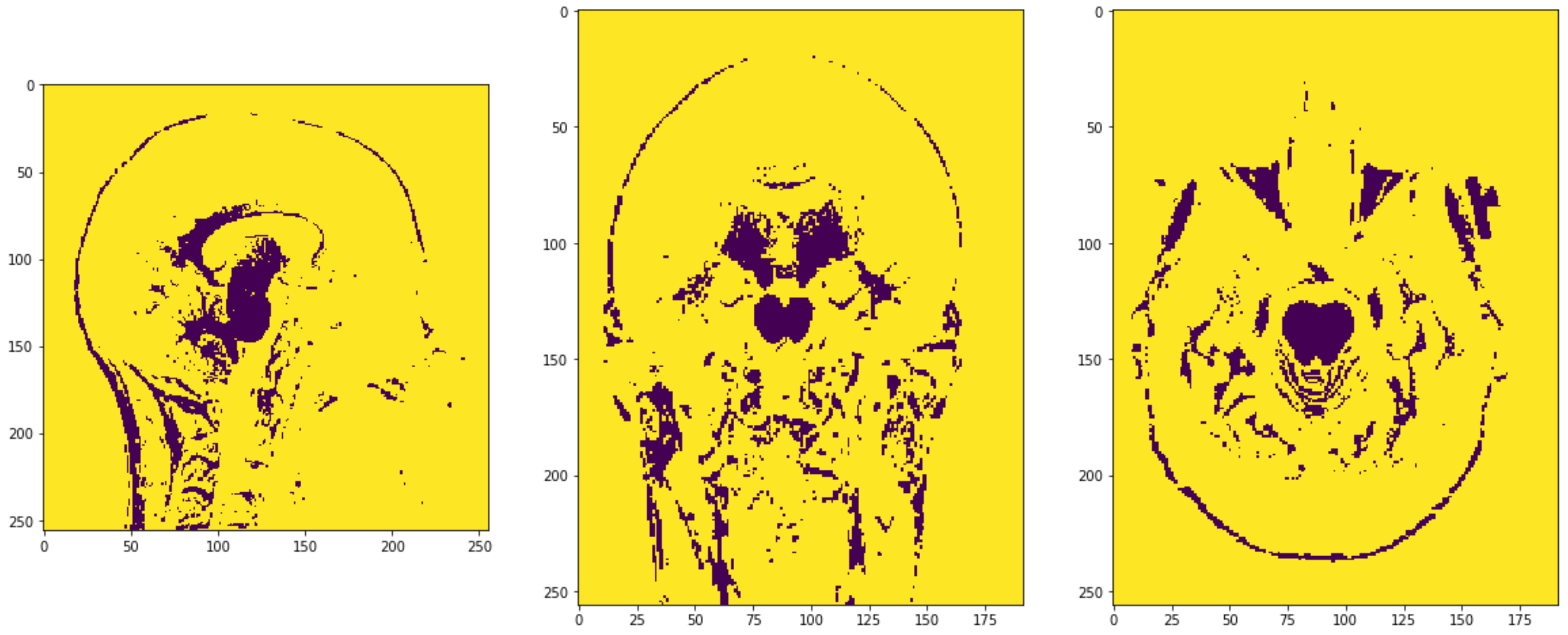
c) all values above this peak.

use *imshow* again for display.

Can you clearly get a certain structure or are they mixed up?

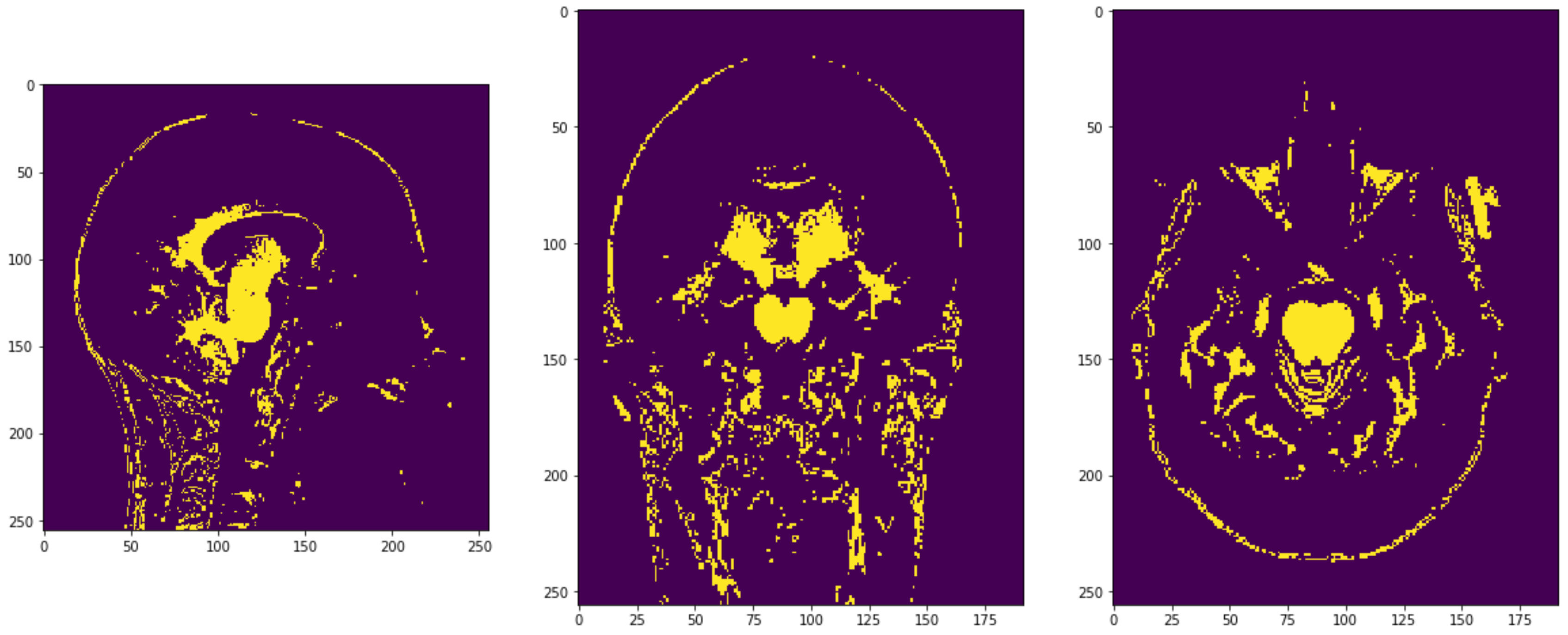
In [4]:
```
lateral = ndimage.rotate(mri[96,:,:], 90)
frontal = ndimage.rotate(mri[:,128,:], 90)
superior = ndimage.rotate(mri[:,:,128], 90)

for i in range(len(lateral)):
    for j in range(len(lateral[i])):
        if(lateral[i][j] < 400):
            lateral[i][j] = 1
        else:
            lateral[i][j] = 0

for i in range(len(frontal)):
    for j in range(len(frontal[i])):
        if(frontal[i][j] < 400):
            frontal[i][j] = 1
        else:
            frontal[i][j] = 0

for i in range(len(superior)):
    for j in range(len(superior[i])):
        if(superior[i][j] < 400):
            superior[i][j] = 1
        else:
            superior[i][j] = 0


fig, ax =plt.subplots(1,3,figsize=(20,20))
ax[0].imshow(lateral)
ax[1].imshow(frontal)
ax[2].imshow(superior)
```

```
Out[4]: <matplotlib.image.AxesImage at 0x7fd3bd787748>
```



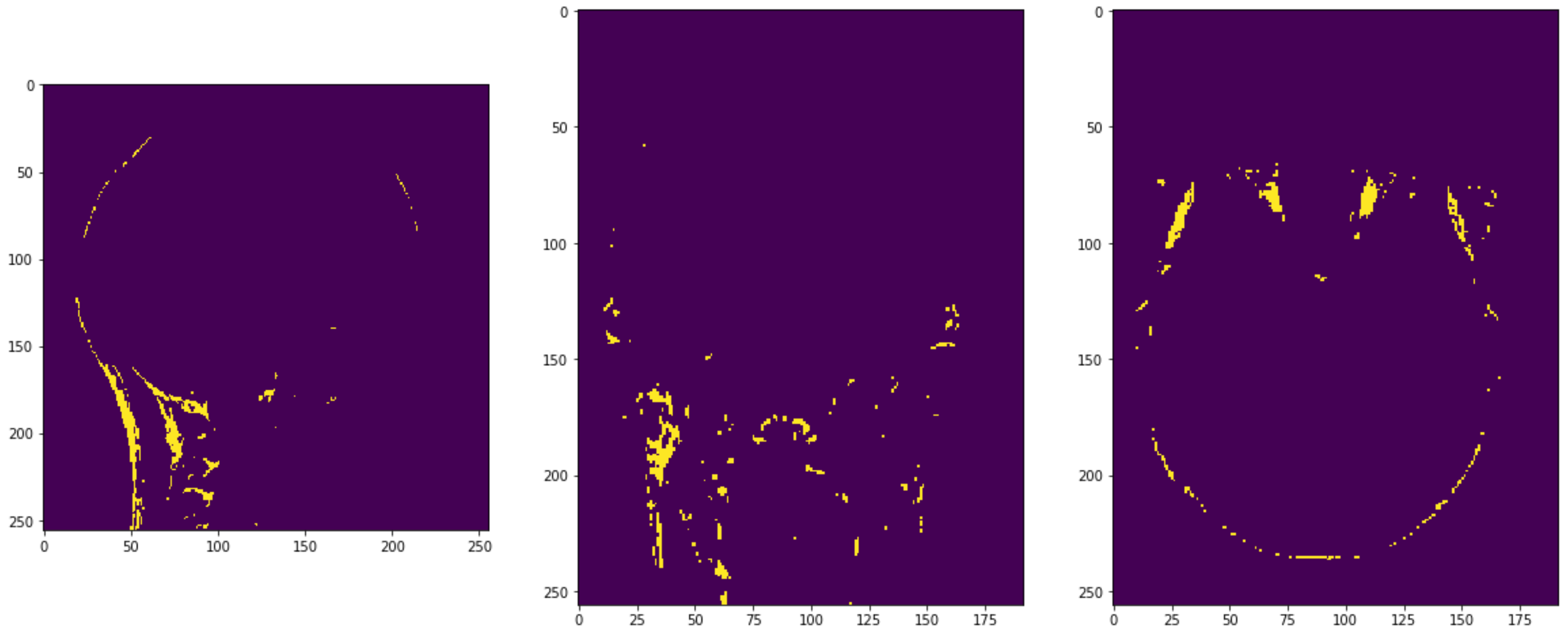
In [5]:
```
lateral = ndimage.rotate(mri[96,:,:], 90)
frontal = ndimage.rotate(mri[:,128,:], 90)
superior = ndimage.rotate(mri[:,:,128], 90)

#we choose between 400 - 600
for i in range(len(lateral)):
    for j in range(len(lateral[i])):
        if(lateral[i][j] >= 400 and lateral[i][j] <= 600):
            lateral[i][j] = 1
        else:
            lateral[i][j] = 0

for i in range(len(frontal)):
    for j in range(len(frontal[i])):
        if(frontal[i][j] >= 400 and frontal[i][j] <= 600):
            frontal[i][j] = 1
        else:
            frontal[i][j] = 0

for i in range(len(superior)):
    for j in range(len(superior[i])):
        if(superior[i][j] >= 400 and superior[i][j] <= 600):
            superior[i][j] = 1
        else:
            superior[i][j] = 0


fig, ax =plt.subplots(1,3,figsize=(20,20))
ax[0].imshow(lateral)
ax[1].imshow(frontal)
ax[2].imshow(superior)
```

```
Out[5]: <matplotlib.image.AxesImage at 0x7fd3c587e0b8>
```



In [6]:
```
lateral = ndimage.rotate(mri[96,:,:], 90)
frontal = ndimage.rotate(mri[:,128,:], 90)
superior = ndimage.rotate(mri[:,:,128], 90)

for i in range(len(lateral)):
    for j in range(len(lateral[i])):
        if(lateral[i][j] > 600):
            lateral[i][j] = 1
        else:
            lateral[i][j] = 0

for i in range(len(frontal)):
    for j in range(len(frontal[i])):
        if(frontal[i][j] > 600):
            frontal[i][j] = 1
        else:
            frontal[i][j] = 0

for i in range(len(superior)):
    for j in range(len(superior[i])):
        if(superior[i][j] > 600):
            superior[i][j] = 1
        else:
            superior[i][j] = 0


fig, ax =plt.subplots(1,3,figsize=(20,20))
ax[0].imshow(lateral)
ax[1].imshow(frontal)
ax[2].imshow(superior)
```

```
Out[6]: <matplotlib.image.AxesImage at 0x7fd3beff0048>
```



our choosen intensity range seems to show the celebellum and most of the skull.

Task 3: Tissue masks (1 point)

Correct segmentation is a non-trivial task, which a lot of scientific literature can be found on. There are automatic and semi-automatic routines, but many times aditional manual error-correction is done if precise segmentations are needed. Here we want to look at the results of on an automatic routine developed in [Huang, Y., Dmochowski, J.P., Su, Y. and Datta, A. (2013), 'Automated mri segmentation for individualized modeling of current flow in the human head', Journal of Neural Engineering, 10(6):066004].

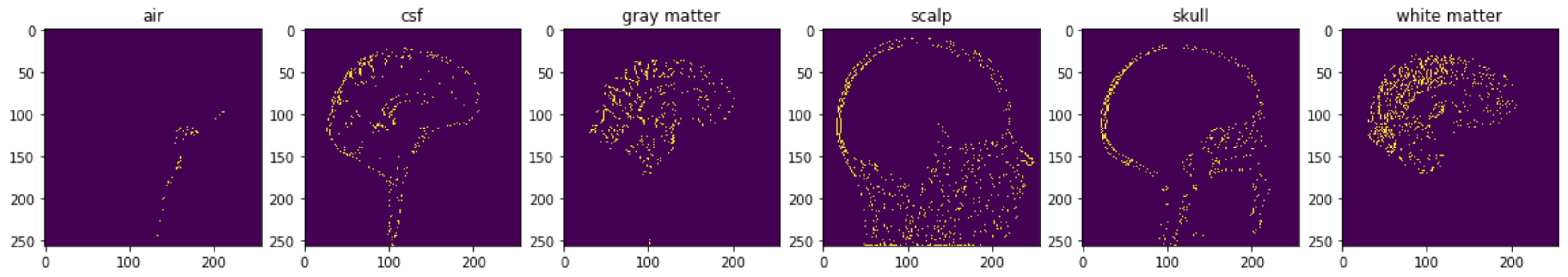
The tissue masks extracted by this algorithm have been based upon the SPM-toolbox (http://www.fil.ion.ucl.ac.uk/spm/ (http://www.fil.ion.ucl.ac.uk/spm/)) and the new_segement algorithm therein. The algorithm uses a tissue probability map (TPM) extended to the neck, where local probabilities of tissue occurrence are additionally used as priors for the extraction. The MRI is first matched to the TPM by a non-linear deformation (=warp) and then the local intensities are evaluated based on their intensity value and the additional information of local tissue probability.

Tasks:

Load the files for the 6 different tissue types (air,CSF,gray matter,white matter, scalp, skull) and display them all in a medial sagittal cut (the medium vertical plane from the side of the head) with *imshow*.

In [7]:
```
air=np.load('files/air.npy')
csf=np.load('files/csf.npy')
gray=np.load('files/gray.npy')
scalp=np.load('files/scalp.npy')
skull=np.load('files/skull.npy')
white=np.load('files/white.npy')
# -> the different tissue types

lateral_air = ndimage.rotate(air[96,:,:], 90)
lateral_csf = ndimage.rotate(csf[96,:,:], 90)
lateral_gray = ndimage.rotate(gray[96,:,:], 90)
lateral_scalp = ndimage.rotate(scalp[96,:,:], 90)
lateral_skull = ndimage.rotate(skull[96,:,:], 90)
lateral_white = ndimage.rotate(white[96,:,:], 90)

fig, ax =plt.subplots(1,6,figsize=(20,20))
ax[0].set_title("air")
ax[0].imshow(lateral_air)
ax[1].set_title("csf")
ax[1].imshow(lateral_csf)
ax[2].set_title("gray matter")
ax[2].imshow(lateral_gray)
ax[3].set_title("scalp")
ax[3].imshow(lateral_scalp)
ax[4].set_title("skull")
ax[4].imshow(lateral_skull)
ax[5].set_title("white matter")
ax[5].imshow(lateral_white)
```

Out[7]:
```
<matplotlib.image.AxesImage at 0x7fd3bdb6c550>
```



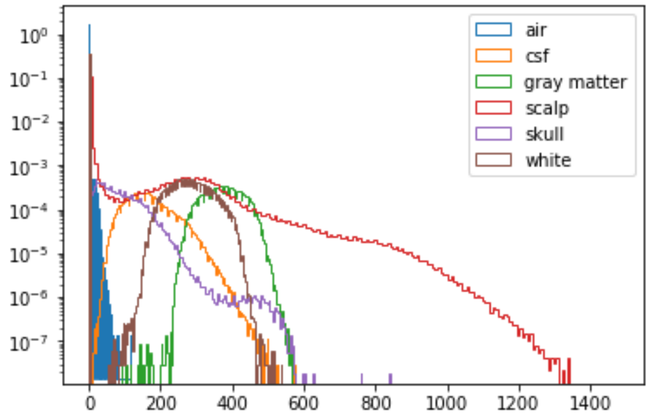
Task 4: T1 intensities by tissue class (2 points)

As we now have the spatial distribution of differen tissue types, we can check how the different T1-intensity values are distributed within each type.

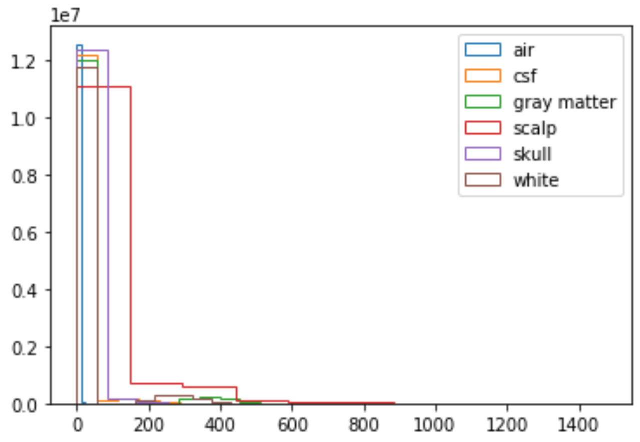
Tasks:

Therefore, plot histograms (*np.histogram*) that only consist of those parts of the MRI that belong to a certain tissue types by using the masks from exercise 3. Also, plot a histogram of all 6 tissue types at once (this is different than the histograms from task 1, because the air outside the head has been removed from the segmentations). Plot all histograms into one plot.

In [8]:
```
plt.hist(np.ravel(air * mri), histtype="step", label="air", bins = 200,log=True,normed=True)
plt.hist(np.ravel(csf * mri), histtype="step", label="csf", bins = 200,normed=True)
plt.hist(np.ravel(gray * mri), histtype="step", label="gray matter", bins = 200,normed=True)
plt.hist(np.ravel(scalp * mri), histtype="step", label="scalp", bins = 200,normed=True)
plt.hist(np.ravel(skull * mri), histtype="step", label="skull", bins = 200,normed=True)
plt.hist(np.ravel(white * mri), histtype="step", label="white", bins = 200,normed=True)
plt.legend()
plt.show()
```



In [9]:
```
#without log
plt.hist(np.ravel(air * mri), histtype="step", label="air")
plt.hist(np.ravel(csf * mri), histtype="step", label="csf")
plt.hist(np.ravel(gray * mri), histtype="step", label="gray matter")
plt.hist(np.ravel(scalp * mri), histtype="step", label="scalp")
plt.hist(np.ravel(skull * mri), histtype="step", label="skull")
plt.hist(np.ravel(white * mri), histtype="step", label="white")
plt.legend()
plt.show()
```



Task 5: 4-shell 3D-mesh of a human head (1 point)

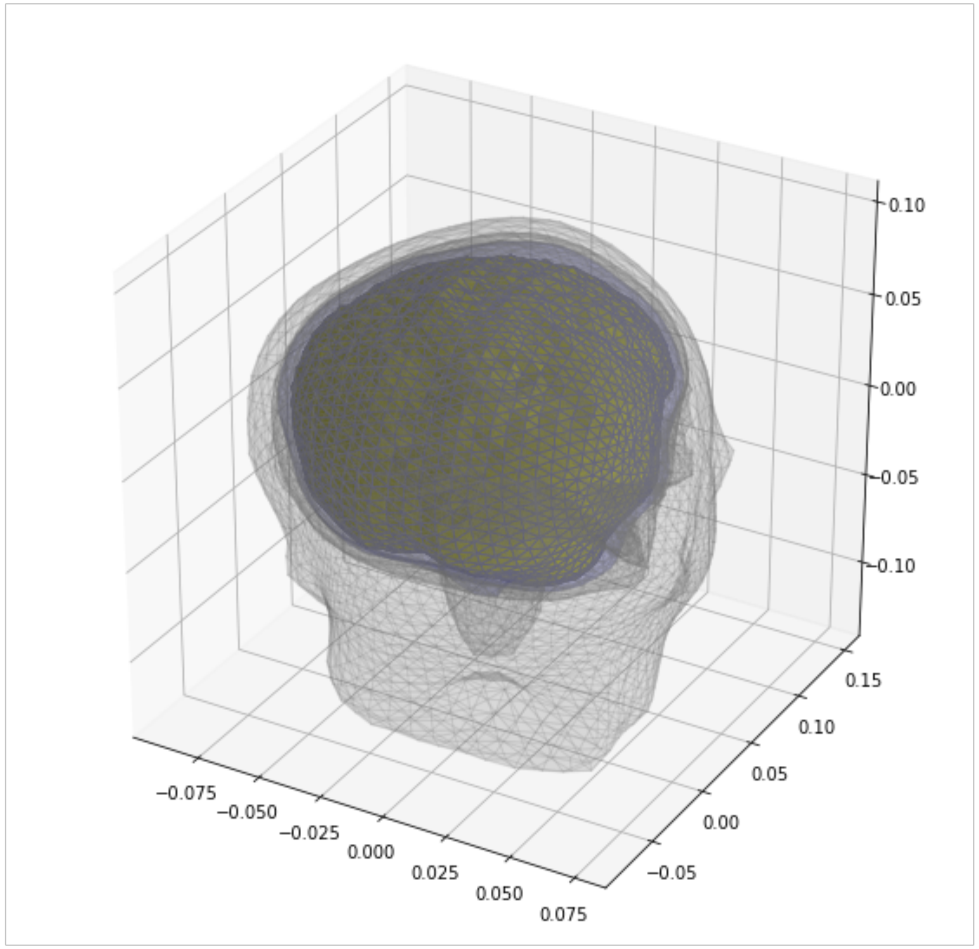
From these segmented MRIs, different geometrical representations can be extracted for further numerical head modeling. For Finite Element Method (FEM) head modeling, using the voxels of the MRI as the single discrete finite elements would be the next step. We will in contrast focus on the Boundary Element Method (BEM) for head modeling. For BEM head models a representation of the boundaries between different subdomains is needed. This is mostly done in a triangular fashion, where every surface is represented by a set of vertexes (points) and the triangles connecting them.

The extraction of the meshes was in our case done by the *project_mesh* algorithm of the fieldtrip tolbox (http://fieldtriptoolbox.org/ (http://fieldtriptoolbox.org/)), which starts with a uniform polygon for each tissue type and projects every vertex of it to the outermost position still within the specific tissue type. The results for 4-shells with tissue types brain, CSF, skull and scalp are saved in positions of vertices in 3 dimensions (pos1-pos4) and the indices of the individual vertices connected by triangles (tri1-tri4). The variable names are sorted from inside of the brain (pos1/tri1) to the scalp (pos4/tri4) and the origin is located at the intra-auricular point within the brain (center between both ears).

Tasks:

Use the *ax.plot_trisurf* function to plot the 4 shells into one 3-dimensional figure (*ax = fig.add_subplot(111, projection='3d'*) in different colors. Set the alpha level (the transparency) of each shell accordingly to be able to see the inner shells from outside.

In [10]:
```
pos1=np.load('files/bnd4_pos1.npy')
tri1=np.load('files/bnd4_tri1.npy')
pos2=np.load('files/bnd4_pos2.npy')
tri2=np.load('files/bnd4_tri2.npy')
pos3=np.load('files/bnd4_pos3.npy')
tri3=np.load('files/bnd4_tri3.npy')
pos4=np.load('files/bnd4_pos4.npy')
tri4=np.load('files/bnd4_tri4.npy')

fig = plt.figure(figsize=(10, 10))
ax=fig.add_subplot(111, projection='3d')

x_axes = pos1[:,0]
y_axes = pos1[:,1]
z_axes = pos1[:,2]

ax.plot_trisurf(x_axes, y_axes, z_axes, triangles=tri1, color='yellow', alpha=1, edgecolor='gray')


x_axes2 = pos2[:,0]
y_axes2 = pos2[:,1]
z_axes2 = pos2[:,2]

ax.plot_trisurf(x_axes2, y_axes2, z_axes2, triangles=tri2, color='blue', alpha=0.1, edgecolor='gray')


x_axes3 = pos3[:,0]
y_axes3 = pos3[:,1]
z_axes3 = pos3[:,2]

ax.plot_trisurf(x_axes3, y_axes3, z_axes3, triangles=tri3, color='grey', alpha=0.1, edgecolor='gray')

x_axes4 = pos4[:,0]
y_axes4 = pos4[:,1]
z_axes4 = pos4[:,2]

ax.plot_trisurf(x_axes4, y_axes4, z_axes4, triangles=tri4, color='grey', alpha=0.1, edgecolor='gray')


plt.show()
```


0.10
0.05
0.00
−0.05
−0.10
0.15
0.10
0.05
0.00
−0.05
−0.075
−0.050
−0.025
0.000
0.025
0.050
0.075

Task 6: Head models and the equivalent current dipole (5 points)

From the boundaries of task 5, a head model can be created using a BEM solver like openMEEG (http://openmeeg.github.io (http://openmeeg.github.io)). This solver assumes homogenity and anisotropy in conductivity within each of the compartements and solves the electrical field and so the current flow within the head by describing it through the jump of the normal current denisty and the jump of the potential on the interfaces (surfaces) between the different subdomains. These jumpy are 0 except if active sources like neuronal activity or current injection in Electrical Impedance Tomography (EIT) or transcrainal current stimulation (tCS) are present. In general also muscle activities leading to EEG artefacts can be modeled this way.

Neuronal source positions are mostly defined based on theoretical assumptions. These are, that the main signals originate in the gray matter close to the cerebral cortex (see lecture 1) of the brain. In this simulation, gridpositions are assumed to be on the interface between gray and white matter which introduces a small error but simplifies source location extraction. The positions of the 3990 single sources are saved in the variable *gridpos*.

The propagation of neuronal activity to the scalp in the range below 100Hz is mostly assumed to be linear in amplitude. This is an effect of the electrostatic assumption in combination with piecewise constant conductivities. The matrix describing the corresponding scalp potential $\mathbf{v}$ for every single neruonal source is mostly called the leadfield L. This leads to the equation:

$$\mathbf{v} = L\mathbf{s}.$$

In general this leadfield is based on a representation incorporating a head matrix *A* connecting the boundary conditions for all the surfaces with the source terms b coming from neuronal activity or current injections. This so-called forward problem is defined by:

$$A\mathbf{x} = \mathbf{b}$$

To calculate the voltages and currents on the surface, the inverse A^{-1} is needed, which is here given in the variable *hminv*. $b_i = [B]_{(i)} \cdot [P]_{(i)}$ for the neuronal sources of EEG is given by the inner product of *dsm* (dipole source model) B with the individual dipole moment $[P]_{(i)}$. B is 3-dimensional in the second dimension to account for the spatial orientation of the dipole. By the inner product with the dipole moment $[P]_{(i)}$ in 3 dimensions for each source location it is reduced to one value per source which is thus reduced to the vector $\mathbf{b}$. Usually this is assumed normal to the cortex but here we simply define it indipendent of that for simplicity. If we take unity dipole moments ($\left|[P]_{(i)}\right|^2 = 1$), the source amplitude $\mathbf{s}$ can be noted as a single vector with one value per source. The corresponding operation for this is inner product between *dsm* and the dipole moment matrix P is *b=np.tensordot(dsm,P,1)* but for a single dipole p_i selecting the adequate rows of *dsm* before the dot product speeds up and simplifies the source code to *b=np.dot(dsm[:,iDip],pi)*.

$$\mathbf{x} = A^{-1}\mathbf{b}$$

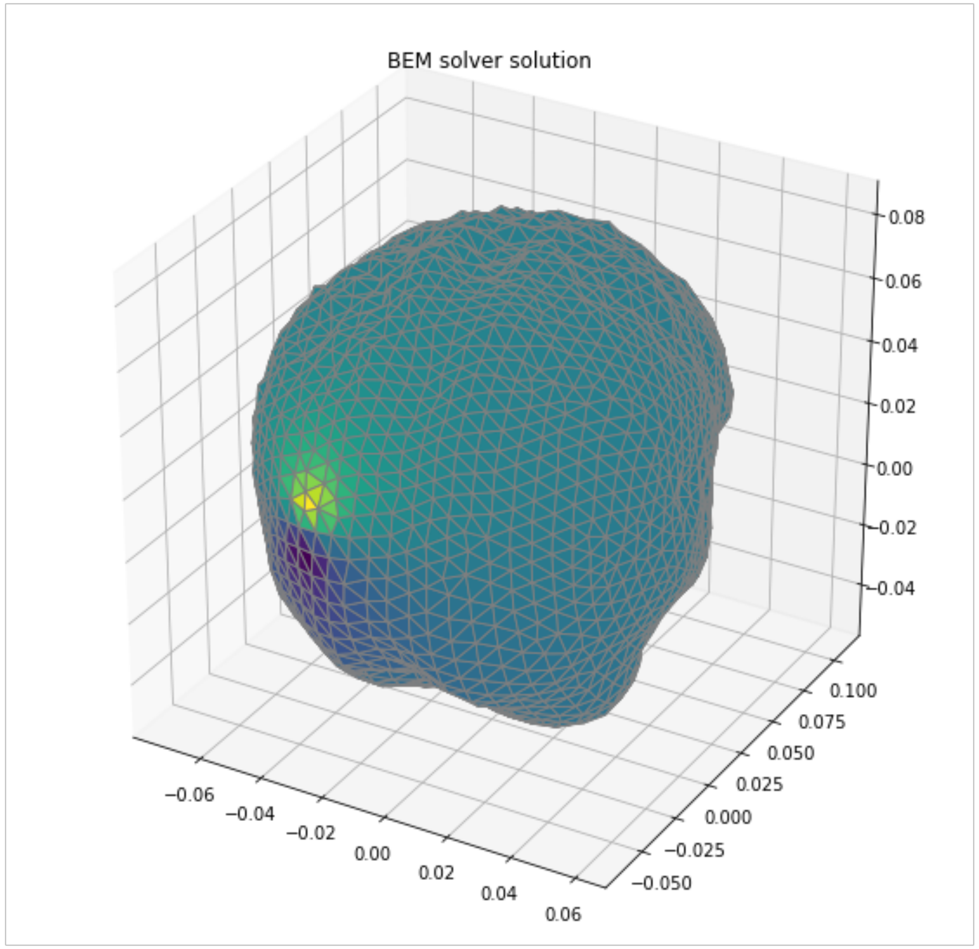
to get from single source activity (amplitude) $\mathbf{s}$ to the state variables (jumps of potentials and currents on the sufaces) $\mathbf{x}$. The potentials on the scalp or brain can then be interpolated from the variable $\mathbf{x}$ by a certain interpolation matrix. For the scalp surface this interpolation matrix W is here called *h2em* (head to electrode model) and the one corresponding to the brain surface potenial is *h2cortex* (head to cortex surface). By the marix product of $\mathbf{x}$ with *h2em* and *h2cortex* we get the resulting potentials on the scalp

In [11]:
```
def phi_dip(r, p, r_0=None):
    r=np.array(r)
    p=np.array(p)
    if r_0 is not None:
        r_0=np.array(r_0)
        r=(r.T-r_0).T
    return 1/(4*np.pi*constants.epsilon_0)*(np.tensordot(p,r,1))/np.power(np.linalg.norm(r,axis=0),3)

h2em=np.load('files/h2em.npy')
h2cortex=np.load('files/h2cortex.npy')
hminv=np.load('files/hminv.npy')
dsm=np.load('files/dsm.npy')
gridpos=np.load('files/gridpos.npy')
pos1=np.load('files/bnd4_pos1.npy')
tri1=np.load('files/bnd4_tri1.npy')
NoTri=tri1.shape[0]
NoPnt=pos1.shape[0]
```

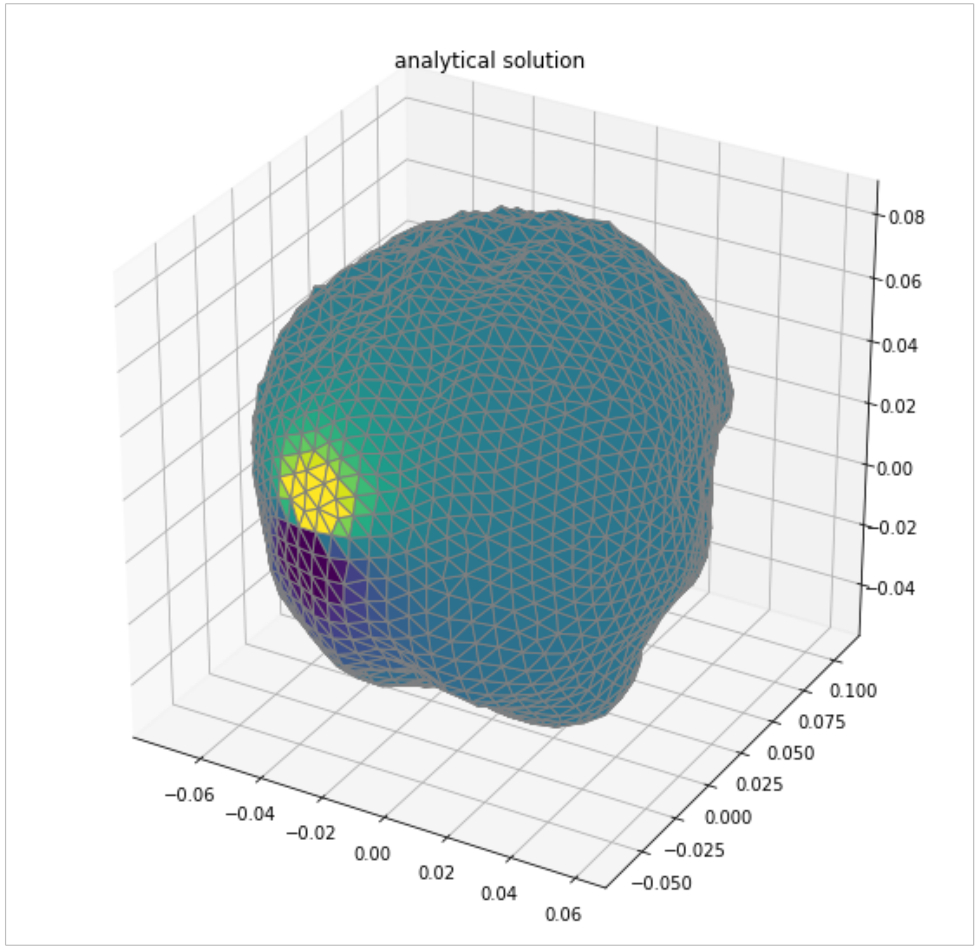
In [12]:
```
iDip=np.random.randint(dsm.shape[1])
p=[0,0,1]
b=np.dot(dsm[:,iDip],p)

x = hminv.dot(b)
#scalp surface: interpolation with h2em
v_scalp = h2em.dot(x)
#brain surface potential: h2cortex
v_cortex = h2cortex.dot(x)

#different constants -> get BEM & analytical on scale
v_cortex = v_cortex*0.201/constants.epsilon_0

#convert size of v_cortex to triangles with average
fcv = np.zeros(NoTri)
for i,ind in enumerate(tri1):
    fcv[i] = (v_cortex[ind[0]]+v_cortex[ind[1]]+v_cortex[ind[2]])/3
```

In [13]:
```
#RGBA color transformation
sm = cm.ScalarMappable()
facecol_cortex = sm.to_rgba(fcv)
```

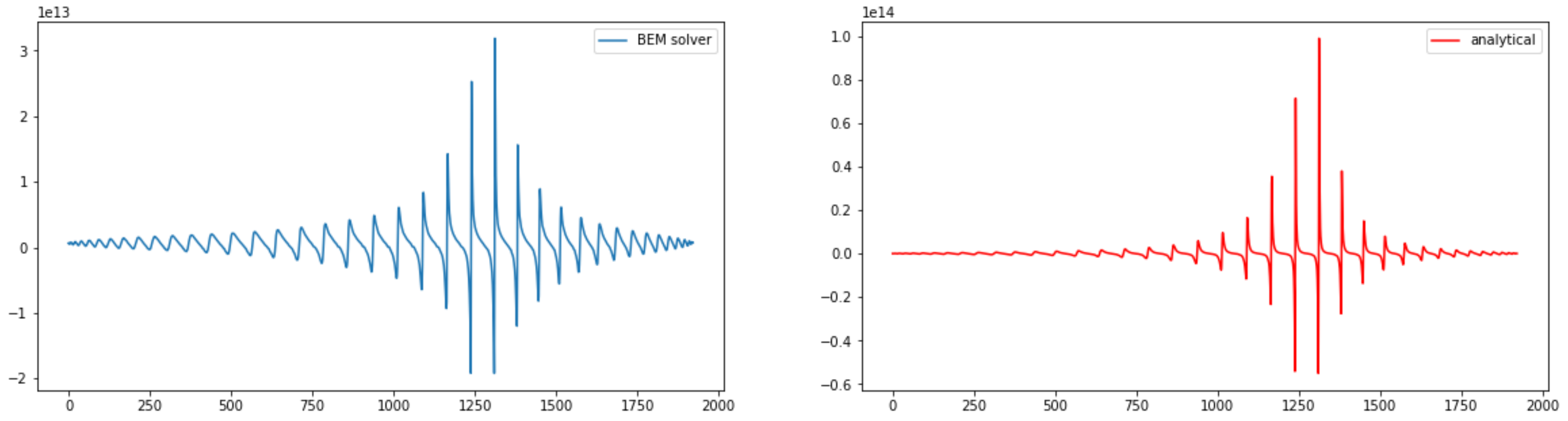
In [14]:
```
x_axes = pos1[:,0]
y_axes = pos1[:,1]
z_axes = pos1[:,2]

fig = plt.figure(figsize=(10, 10))
ax=fig.add_subplot(111, projection='3d')
tsp = ax.plot_trisurf(x_axes, y_axes, z_axes, triangles=tri1,  edgecolor='gray')
tsp.set_facecolors(facecol_cortex)
plt.title('BEM solver solution')
plt.show()
```


BEM solver solution
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.06
−0.04
−0.02
0.00
0.02
0.04
0.06

In [15]:
```
#analytical solution
ana_cortex = phi_dip(pos1.T,p,gridpos[iDip,:])

#convert size of v_cortex to triangles with average
fca = np.zeros(NoTri)
for i,ind in enumerate(tri1):
    fca[i] = (ana_cortex[ind[0]]+ana_cortex[ind[1]]+ana_cortex[ind[2]])/3

facecol_analytical = sm.to_rgba(fca)

fig = plt.figure(figsize=(10, 10))
ax=fig.add_subplot(111, projection='3d')
tsp = ax.plot_trisurf(x_axes, y_axes, z_axes, triangles=tri1,  edgecolor='gray')
tsp.set_facecolors(facecol_analytical)
plt.title('analytical solution')
plt.show()
```


analytical solution
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.06
−0.04
−0.02
0.00
0.02
0.04
0.06

```
In [16]: fig,ax = plt.subplots(1,2,figsize=(20,5))
         ax[0].plot(v_cortex,label='BEM solver')
         ax[1].plot(ana_cortex,label='analytical',color='red')
         ax[0].legend()
         ax[1].legend()
         plt.show()
```



the solutions are not equal. the analytical solution is more acurate where the BEM solver carries some noise. the analytical has less potential farer away from source and more potential around the source

Task 7: The potential on different surfaces (3 points)

The voltage V_{S_i} on each vertex and the normal current p_{S_i} through each triangle can also be accessed directly from the state variable of the linear system. The variables are sorted in the following manner:

$$x = \begin{bmatrix} V_{S_1} \\ p_{S_1} \\ V_{S_2} \\ p_{S_2} \\ V_{S_3} \\ p_{S_3} \\ V_{S_4} \end{bmatrix} = A^{-1}b$$

where V_{S_i} corresponds to the vertex potentials on surface i and p_{S_i} is the normal current through triangle i. All V_{S_i} have 1922 entries (the No. of verteces per surface), while each p_{S_i} consists of 3840 triangles. The surfaces are sorted from inside out and S_1 is the brain surface, while S_4 is the scalp.

A^{-1} is the variable *hminv* and b is called *dsm*.

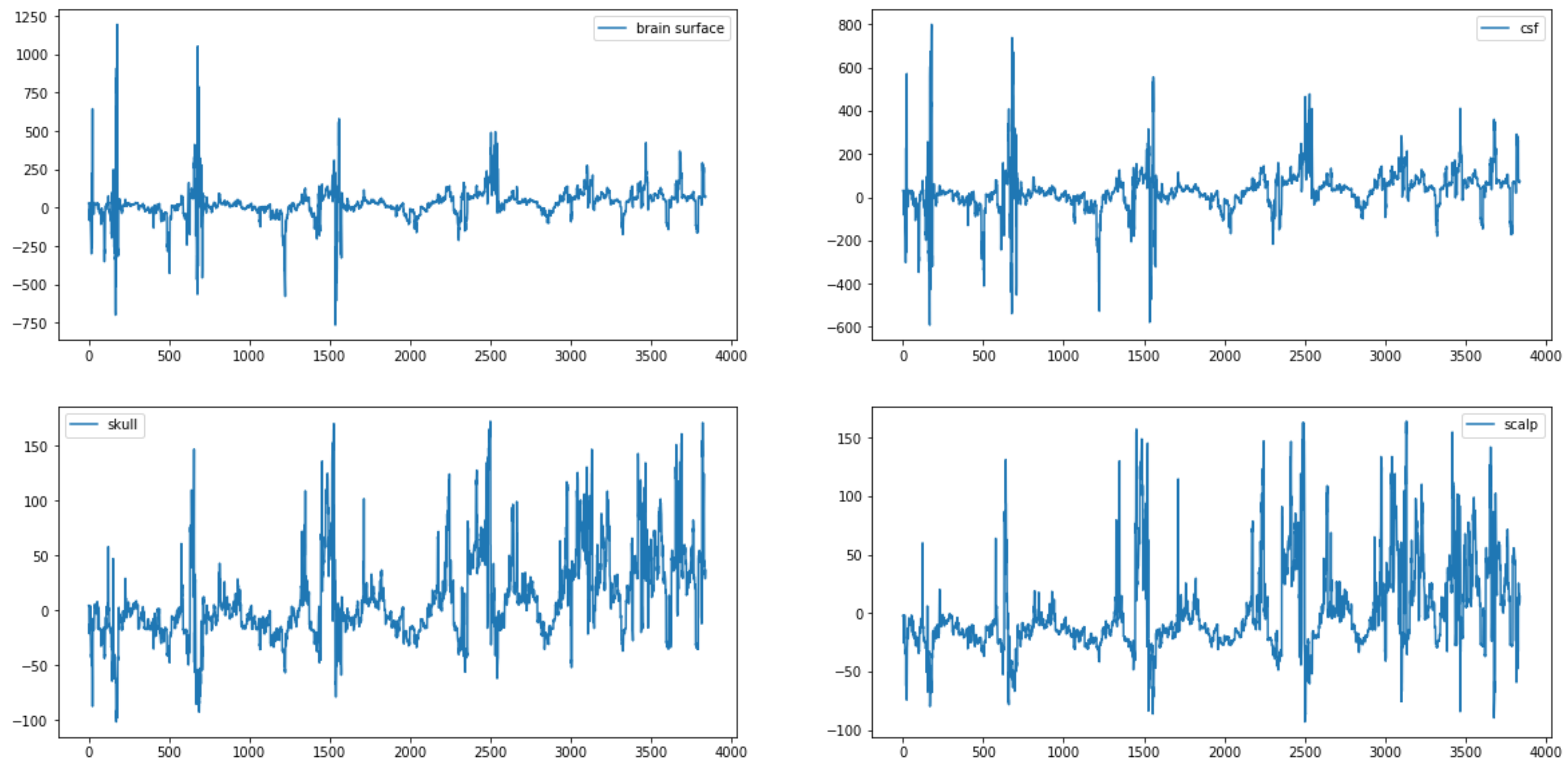
Tasks:

a) Calculate the potential on each triangle by an average directly over the state variables of the adjacent points and then plot it for all 4 surfaces.

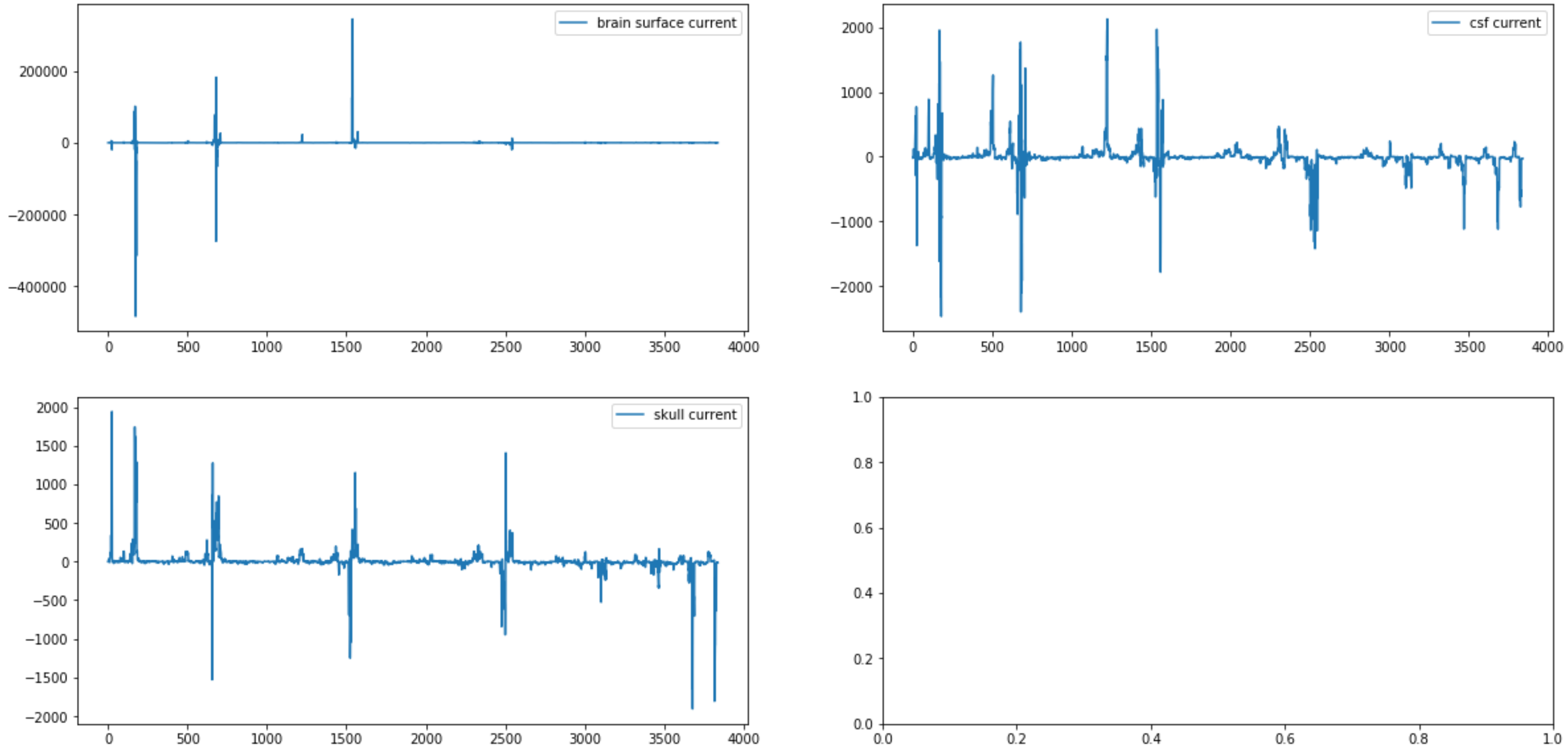
b) Plot the normal current through each triangle for all 4 subjects directly.

In [18]:
```
#getting x-direction
X = hminv.dot(b)
```

In [19]:
```python
def return_triangle_avg(array,tri):
    av = np.zeros(tri.shape[0])
    for i,ind in enumerate(tri):
        av[i] = (array[ind[0]] + array[ind[1]] + array[ind[2]])/3
    return av


vs1 = X[0:1922]
ps1 = X[1922:5762]
vs2 = X[5762:7684]
ps2 = X[7684:11524]
vs3 = X[11524:13446]
ps3 = X[13446:17286]
vs4 = X[17286::]

avS1 = return_triangle_avg(vs1,tri1)
avS2 = return_triangle_avg(vs2,tri2)
avS3 = return_triangle_avg(vs3,tri3)
avS4 = return_triangle_avg(vs4,tri4)

fig,ax = plt.subplots(2,2,figsize=(20,10))
ax[0,0].plot(avS1,label='brain surface')
ax[0,0].legend()
ax[0,1].plot(avS2,label='csf')
ax[0,1].legend()
ax[1,0].plot(avS3,label='skull')
ax[1,0].legend()
ax[1,1].plot(avS4,label='scalp')
ax[1,1].legend()
plt.show()
```


brain surface
csf
skull
scalp

In [20]:
```
fig,ax = plt.subplots(2,2,figsize=(20,10))
ax[0,0].plot(ps1,label='brain surface current')
ax[0,0].legend()
ax[0,1].plot(ps2,label='csf current')
ax[0,1].legend()
ax[1,0].plot(ps3,label='skull current')
ax[1,0].legend()

plt.show()
```



Task 8: EEG: scalp and electrode potential (1 point)

Those of you who attended the BCI-IL course in the winter term might remember the bci_minitoolbox with it's function *scalpmap*. We used this function to plot event-related potentials (ERPs) measured by EEG on a 2-dimensional scalp, which is a standard procedure in EEG analysis. Also, we plotted spatial patterns and filters with it.

Now, we can use the same function to plot a simulated scalp potential from a neuronal source of our head modeling approach. Therefore we will need to use the head to electrode model *h2em* to interpolate the voltages at the electrodes from those at the vertex. For the scalp surface the interpolation matrix W is here called *h2em* (head to electrode model). By the marix product of x with *h2m* and *h2cortex* we get the resulting potentials on the scalp and on the cortex respectively. To put it into one equation, the leadfield L for head or brain surface is:

$$v_{electrodes} = WA^{-1}b$$

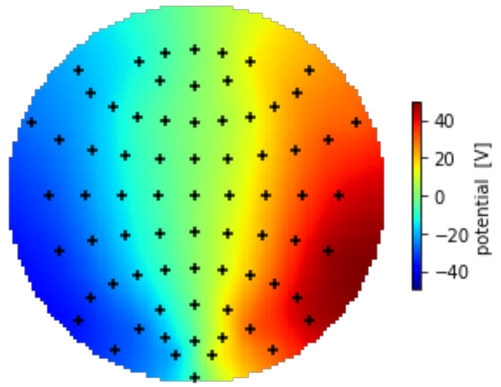
Also, we will need the montage *mnt*, which sets the position for each channel. Additionally you can find the channel labels *clab* to look at the corresponding channel names (not needed to solve the task).

Tasks:

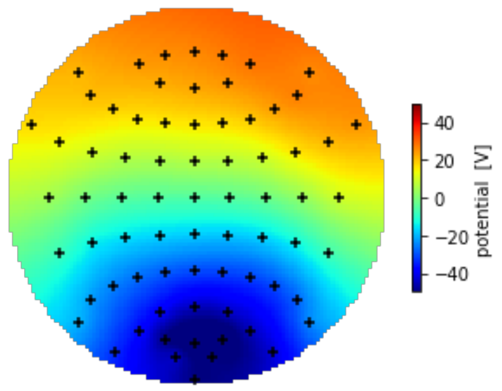
Plot the scalp potential for a dipole of your choice from the leadfield using bci.scalpmap(mnt, V, clim=(-*maxamp*,*maxamp*), cb_label=*labelstring*) from the bci_minitoolbox library that was used in the BCI-IL in winter term.

In [23]:
```
h2em=np.load('files/h2em.npy')
hminv=np.load('files/hminv.npy')
dsm=np.load('files/dsm.npy')
mnt=np.load('files/mnt.npy')
clab=np.load('files/clab.npy')

dip = 100
b = dsm[:,dip]
x=hminv.dot(b)
phielec = h2em.dot(x)
```

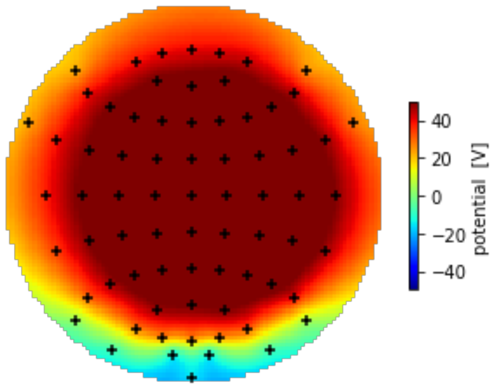
In [33]:
```
maxamp = 50

plt.figure()
bci.scalpmap(mnt, phielec[:,0], clim=(-maxamp,maxamp), cb_label='potential  [V]')
plt.show()
```



```
In [34]: maxamp = 50
         plt.figure()
         bci.scalpmap(mnt, phielec[:,1], clim=(-maxamp,maxamp), cb_label='potential  [V]')
         plt.show()
```



In [40]:
```
maxamp =50
plt.figure()
bci.scalpmap(mnt, phielec[:,2], clim=(-maxamp,maxamp), cb_label='potential  [V]')
plt.show()
```


potential [V]