15/15 points

Exercise Sheet 2: 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
```

Task 1: Equivalent current dipole close to a conductivity change (4 points)

4/4

A simple way of modeling the effect a change in conductivity between two homogeneous subspaces is the so-called "method of images": it can be used to solve the internal field of the source subspace by a superposition of the partial solution and a homogeneous solution. The electrical field/potential of a source distribution outside of the subspace is homogeneous within and can be used to generate correct boundary conditions. This is then added to the partial solution to get correct boundary conditions.

We will only look at a very simple and abstract case, the jump from a compartment containing the source with finite conductivity (the head) to a non-conductive subspace (e.g air outside the head). This is similar as in the lecture.

In this case, no current can leave the head and so the normal current on the boundary has be $J_n(x=0)=0$. This resembles von-Neumann boundary conditions.

Task:

Imagine a subspace with finite conductivity while there is a sudden change to non-conductivity at a vertical line of $x=0$.

Use the functions phi_dip and J_dip from the first exercise sheet (copied below) to model the field of a dipole sitting at $r=[-2,0]$ in infinite homogeneous media. Use the same analytic solution in a different position and orientation to correct the boundary conditions. There is a restriction on the normal current to be zero while the potential can be arbitrary.

Make a plot of the two variables in two dimensions similar to the one from sheet 1 task 5 from -10cm to 0cm for the x axes -10cm to 10cm for the y axes in the x/y-plane for a dipole with dipole moment $\vec{Q}=[20,0]fAm$. Repeat the same with $\vec{Q}=[0,20]fAm$.

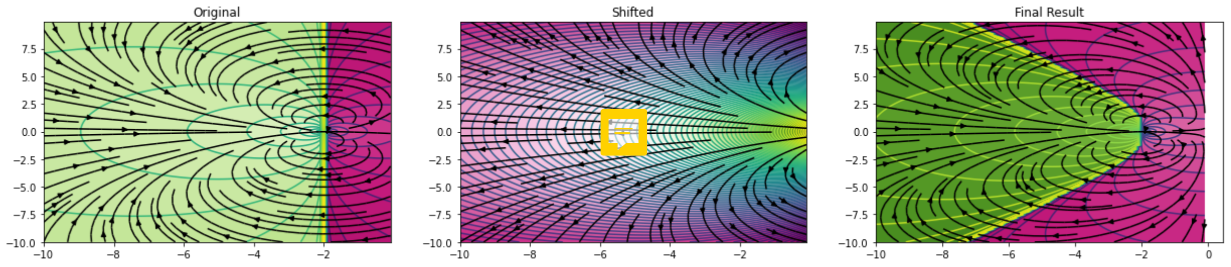
Plot the potential similar as in sheet 1 using the matlpotlib function plt.contour/contourf for plotting and the field strength using the function plt.streamplot. Draw both plots into the same figure.

Hint: Think about how you can reach $J_n(x=0)=0$ by mirroring the infinite solution at the straight line $x=0$ using different values for the dipolar moment.

In [2]:
```
import numpy as np
from scipy import constants
from matplotlib import pyplot as plt

def phi_dip(r, Q, r_0=None,sigma=0.33):
    r=np.array(r)
    Q=np.array(Q)
    if r_0 is not None:
        r_0=np.array(r_0)
        r=(r.T-r_0).T
    return 1/(4*np.pi*sigma)*(np.tensordot(Q,r,1))/np.power(np.linalg.norm(r,axis=0),3)

def J_dip(r, Q, r_0=None,sigma=0.33):
    r=np.array(r)
    Q=np.array(Q)
    if r_0 is not None:
        r_0=np.array(r_0)
        r=(r.T-r_0).T
    nom=((3*np.tensordot(Q,r,1)*r/np.power(np.linalg.norm(r,axis=0),2)).T-Q).T
    return (1/(4*np.pi)*nom/np.power(np.linalg.norm(r,axis=0),3))


# Q([20,0])*1e-15

#initialize the variables (x and y in cm, p as a current dipole with 20fAm)
x=np.arange(-10,0,0.1)
y=np.arange(-10,10,0.1)
r_0=np.array([-2,0])*1e-2
Q=np.array([20,0])*1e-15

#create the meshgrid for calculation and plotting (in SI --> m)
XX,YY=np.meshgrid(x/100,y/100)

#Calculate the potential on the whole meshgrid
phi1=phi_dip([XX,YY],Q,r_0);
#Calculate the current density on the whole meshgrid
J1=J_dip([XX,YY],Q,r_0);


#Shift the dipole to (2,0) and inverse its direction
r_0=np.array([2,0])*1e-2
Q=np.array([-20,0])*1e-15

#Calculate the potential on the whole meshgrid
phi2=phi_dip([XX,YY],Q,r_0);
#Calculate the current density on the whole meshgrid
J2=J_dip([XX,YY],Q,r_0);


# Calculate the final result
phi=phi1+phi2;
J=J1+J2;

#the values in phi from 1e-9 to 1e16, so need a scaling, best way: log base10
#log scaling y = sign(x) * (log|x|)
phi_log1 = np.sign(phi1) * np.log10(np.abs(phi1))
```

```
phi_log2 = np.sign(phi2) * np.log10(np.abs(phi2))
phi_log = np.sign(phi) * np.log10(np.abs(phi))

#PLOT
plt.figure()
plt.figure(figsize=(21,4))

plt.subplot(131)
potplot=plt.contour(x, y, phi_log1,100)
potplot=plt.contourf(x, y, phi_log1,100, cmap=plt.cm.PiYG)
fieldplot=plt.streamplot(x,y,J1[0,:,:],J1[1,:,:],color='k')
plt.title('Original')

plt.subplot(132)
potplot=plt.contour(x, y, phi_log2,100)
potplot=plt.contourf(x, y, phi_log2,100, cmap=plt.cm.PiYG)
fieldplot=plt.streamplot(x,y,J2[0,:,:],J2[1,:,:],color='k')
plt.title('Shifted')

plt.subplot(133)
potplot=plt.contour(x, y, phi_log,100)
potplot=plt.contourf(x, y, phi_log,100, cmap=plt.cm.PiYG)
fieldplot=plt.streamplot(x,y,J[0,:,:],J[1,:,:],color='k')
plt.title('Final Result')
plt.show()
```

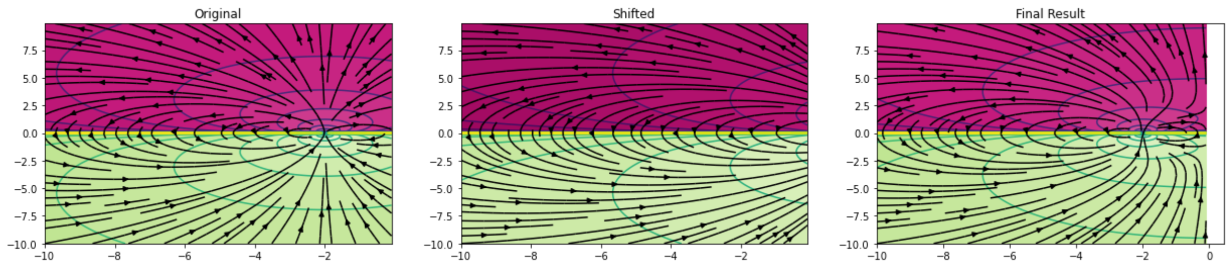
```
<Figure size 432x288 with 0 Axes>
```



In [3]:
```
# Q([0,20])*1e-15

#initialize the variables (x and y in cm, p as a current dipole with 20fAm)
x=np.arange(-10,0,0.1)
y=np.arange(-10,10,0.1)
r_0=np.array([-2,0])*1e-2
Q=np.array([0,20])*1e-15

#create the meshgrid for calculation and plotting (in SI --> m)
XX,YY=np.meshgrid(x/100,y/100)

#Calculate the potential on the whole meshgrid
phi1=phi_dip([XX,YY],Q,r_0);
#Calculate the current density on the whole meshgrid
J1=J_dip([XX,YY],Q,r_0);


#Shift the dipole to (2,0) and inverse its direction
r_0=np.array([2,0])*1e-2
Q=np.array([0,20])*1e-15

#Calculate the potential on the whole meshgrid
phi2=phi_dip([XX,YY],Q,r_0);
#Calculate the current density on the whole meshgrid
J2=J_dip([XX,YY],Q,r_0);


# Calculate the final result
phi=phi1+phi2;
J=J1+J2;

#the values in phi from 1e-9 to 1e16, so need a scaling, best way: log base10
#log scaling y = sign(x) * (log|x|)
phi_log1 = np.sign(phi1) * np.log10(np.abs(phi1))
phi_log2 = np.sign(phi2) * np.log10(np.abs(phi2))
phi_log = np.sign(phi) * np.log10(np.abs(phi))

#PLOT
plt.figure()
plt.figure(figsize=(21,4))

plt.subplot(131)
potplot=plt.contour(x, y, phi_log1,100)
potplot=plt.contourf(x, y, phi_log1,100, cmap=plt.cm.PiYG)
fieldplot=plt.streamplot(x,y,J1[0,:,:],J1[1,:,:],color='k')
plt.title('Original')

plt.subplot(132)
potplot=plt.contour(x, y, phi_log2,100)
potplot=plt.contourf(x, y, phi_log2,100, cmap=plt.cm.PiYG)
fieldplot=plt.streamplot(x,y,J2[0,:,:],J2[1,:,:],color='k')
plt.title('Shifted')

plt.subplot(133)
potplot=plt.contour(x, y, phi_log,100)
potplot=plt.contourf(x, y, phi_log,100, cmap=plt.cm.PiYG)
```

```
fieldplot=plt.streamplot(x,y,J[0,:,:],J[1,:,:],color='k')
plt.title('Final Result')
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



Task 2: T1-MRI scan (1 point)

1/1

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 (on the nose between the eyes). 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.

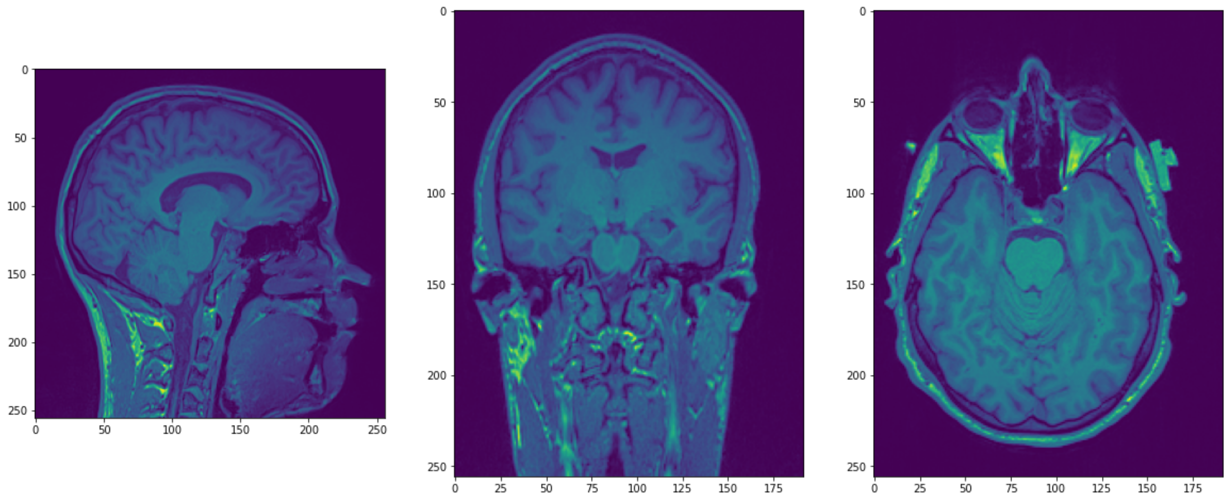
What can you say about the intensity values of different tissue types? Are they well separated?

In [4]:
```
mri=np.load('C:/Users/OwenChang/Desktop/BCI/Leture 2/Exercise 2/Data/T1.npy');
```

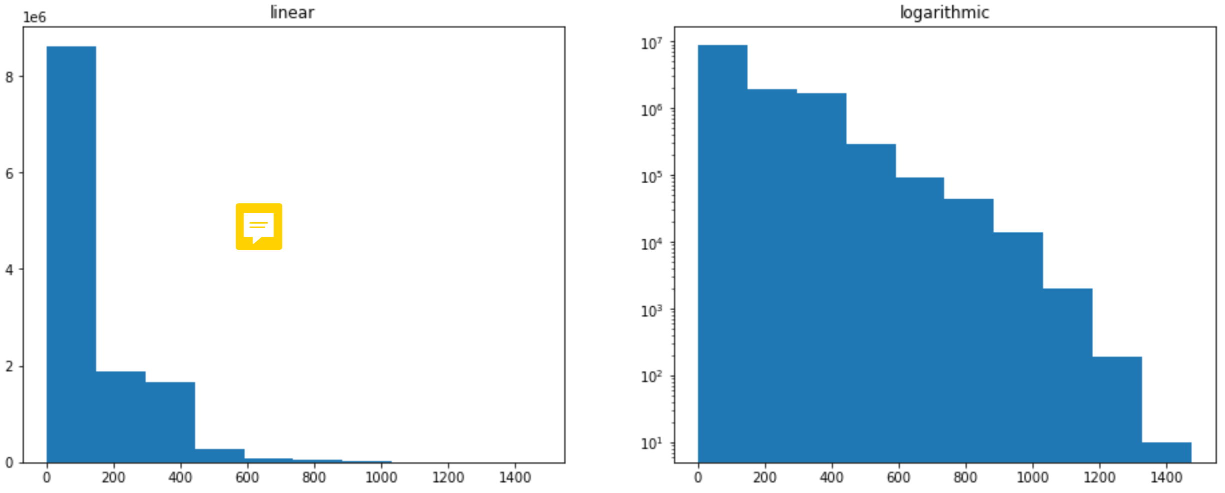
In [5]:
```
mri.shape
```

Out[5]:
```
(192, 256, 256)
```

In [6]:
```
#Show images

lateral = mri[96,:,:]
lateral = np.rot90(lateral)
frontal = mri[:,128,:]
frontal = np.rot90(frontal)
superior = mri[:,:,128]
superior = np.rot90(superior)

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



In [7]:
```
#show histograms

# hist,x = np.histogram(mri.ravel())
# plt.plot(0.5*(x[:-1]+x[1:]),hist)

plt.figure(figsize=(16,6))
plt.subplot(121)
plt.title('linear')
plt.hist(np.ravel(mri)) # bins=20

plt.subplot(122)
plt.title('logarithmic')
plt.hist(np.ravel(mri),log=True)

plt.show()
```



In [8]:
```
#The tissues are not well discrete. In normal histogram, we can see the tissues are focus o
#Interms of logorithmic histgrom, the distribution is a bit better.
```

Task 3: Tissue masks (1 point)

1/1

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].

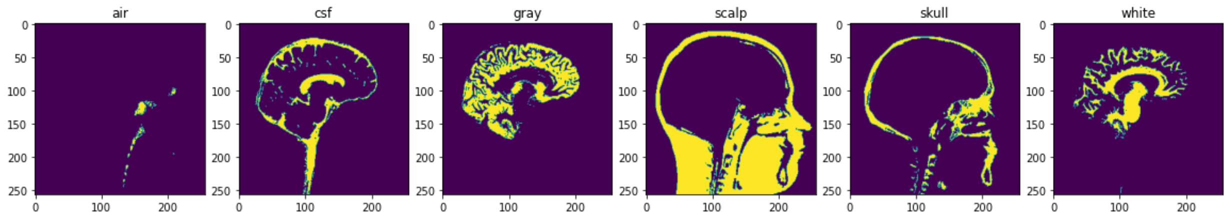
The tissue masks extracted by this algorithm have been based upon the SPM-toolbox (http://www.fil.ion.ucl.ac.uk/spm/ (http://www.fil.ion.ucl.ac.uk/spm/)) and the new_segement algorithm therein. The algorithm uses a tissue probability map (TPM) extended to the neck, where local probabilities of tissue 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 [9]:
```
air=np.load('C:/Users/OwenChang/Desktop/BCI/Leture 2/Exercise 2/Data/air.npy')
csf=np.load('C:/Users/OwenChang/Desktop/BCI/Leture 2/Exercise 2/Data/csf.npy')
gray=np.load('C:/Users/OwenChang/Desktop/BCI/Leture 2/Exercise 2/Data/gray.npy')
scalp=np.load('C:/Users/OwenChang/Desktop/BCI/Leture 2/Exercise 2/Data/scalp.npy')
skull=np.load('C:/Users/OwenChang/Desktop/BCI/Leture 2/Exercise 2/Data/skull.npy')
white=np.load('C:/Users/OwenChang/Desktop/BCI/Leture 2/Exercise 2/Data/white.npy')
```

In [10]:
```
#Show 6 images

types = [air, csf, gray, scalp, skull, white]
types_name = ["air", "csf", "gray", "scalp", "skull", "white"]

fig, ax =plt.subplots(1,6,figsize=(20,20))
for i in range(6):
    lateral = np.rot90(types[i][96,:,:])
    ax[i].title.set_text(types_name[i])
    ax[i].imshow(lateral)
plt.show()
```



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

2/2

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 type by using the masks from exercise 2 on the MRI. The value of each MRI voxel is an intensity value.

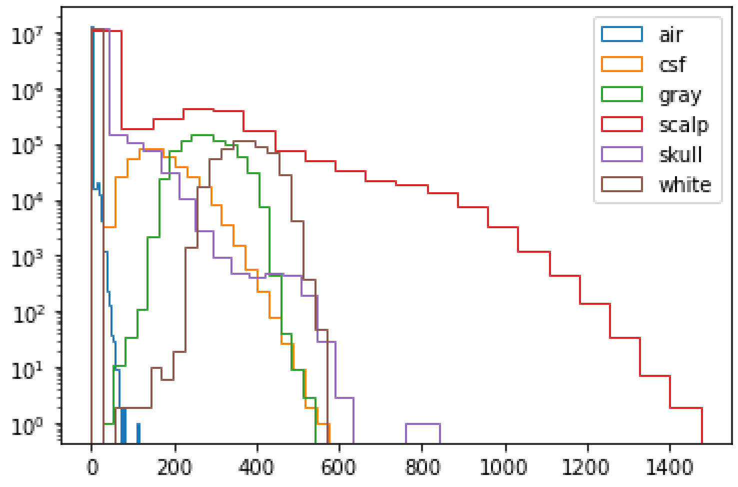
Also, plot a histogram of all 6 tissue types at once (this is different than the histograms from task 1, because the air outside the head has been removed from the segmentations).

Plot all histograms into one plot.

Be careful about the number of voxels per class: Normalize the output to be comparable.

In [11]:
```
mriair=mri*air;
mricsf=mri*csf;
mrigray=mri*gray;
mriscalp=mri*scalp;
mriwhite=mri*white;
```

In [12]:
```
for i in range(6):
    plt.hist(np.ravel(types[i]*mri), histtype="step", label=types_name[i], log=True, bins
plt.legend()
plt.show()
```



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

1/1

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.

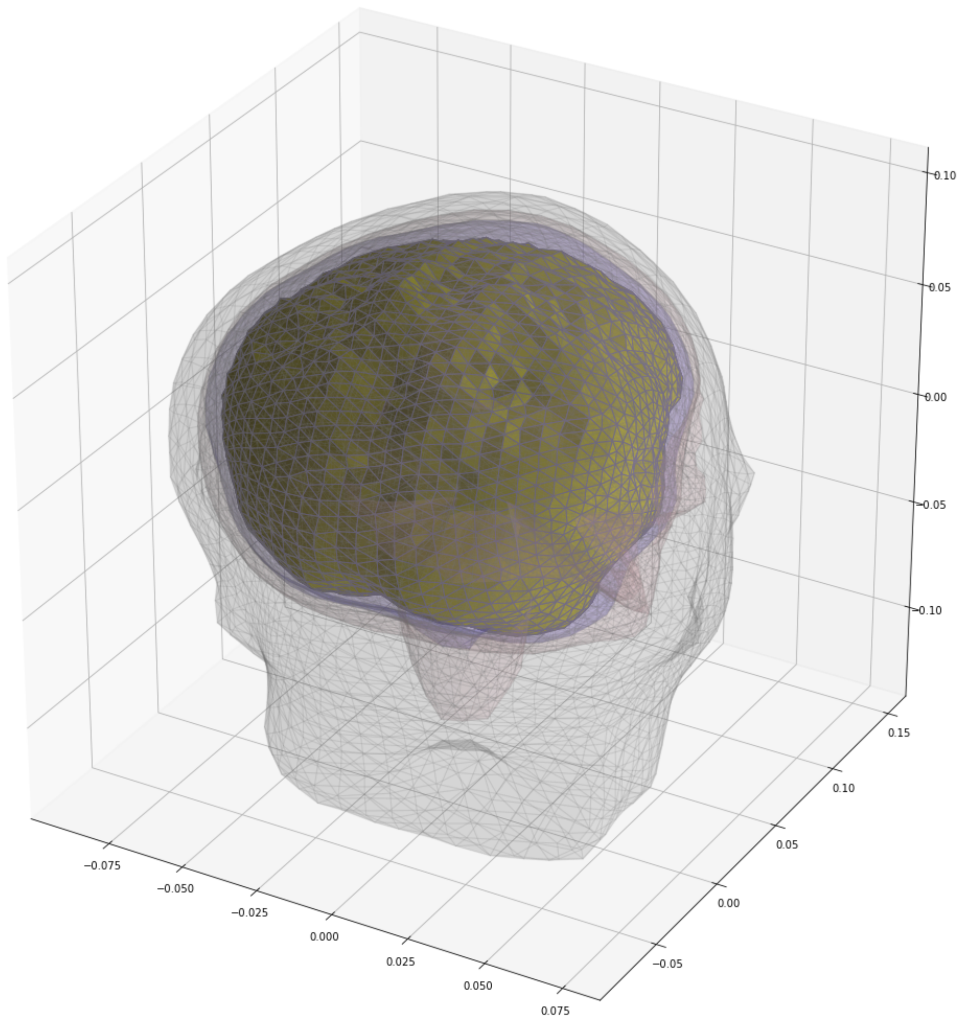
The extraction of the meshes was in our case done by the *project_mesh* algorithm of the fieldtrip tolbox (http://fieldtriptoolbox.org/ (http://fieldtriptoolbox.org/)), which starts with a uniform polygon for each tissue type and projects every vertex of it to the outermost position still within the 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 [13]:
```
pos1=np.load('C:/Users/OwenChang/Desktop/BCI/Leture 2/Exercise 2/Data/bnd4_pos1.npy')
tri1=np.load('C:/Users/OwenChang/Desktop/BCI/Leture 2/Exercise 2/Data/bnd4_tri1.npy')
pos2=np.load('C:/Users/OwenChang/Desktop/BCI/Leture 2/Exercise 2/Data/bnd4_pos2.npy')
tri2=np.load('C:/Users/OwenChang/Desktop/BCI/Leture 2/Exercise 2/Data/bnd4_tri2.npy')
pos3=np.load('C:/Users/OwenChang/Desktop/BCI/Leture 2/Exercise 2/Data/bnd4_pos3.npy')
tri3=np.load('C:/Users/OwenChang/Desktop/BCI/Leture 2/Exercise 2/Data/bnd4_tri3.npy')
pos4=np.load('C:/Users/OwenChang/Desktop/BCI/Leture 2/Exercise 2/Data/bnd4_pos4.npy')
tri4=np.load('C:/Users/OwenChang/Desktop/BCI/Leture 2/Exercise 2/Data/bnd4_tri4.npy')
```

In [14]:
```
fig = plt.figure(figsize=(20, 20))
ax=fig.add_subplot(111, projection='3d')

x_axes1 = pos1[:,0]
y_axes1 = pos1[:,1]
z_axes1 = pos1[:,2]
ax.plot_trisurf(x_axes1, y_axes1, z_axes1, triangles=tri1, color='yellow', alpha=1, edgeco

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, edgeco

x_axes3 = pos3[:,0]
y_axes3 = pos3[:,1]
z_axes3 = pos3[:,2]
ax.plot_trisurf(x_axes3, y_axes3, z_axes3, triangles=tri3, color='pink', alpha=0.1, edgeco

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, edgeco

plt.show()
```



Task 6: EEG leadfield: electrode potential of a source (2 points)

2/2

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.

The file 'em2dsm.npy' contains the leadfield for the head above. This is what most headmodels in the end look like for an EEG researcher. The ledfield describes the potential on the scalp for every single modeled source within the brain.

The leadfield L from 'em2dsm.npy' has the dimensions (NoElec x NoSource x 3). Select an arbitrary dipole and build the inner product of the result with an arbitrary dipolar moment p:

$$v = L_i \cdot p.$$

Now, we can use the *scalpmap* function to plot a simulated scalp potential from a neuronal source of the head modeling approach.

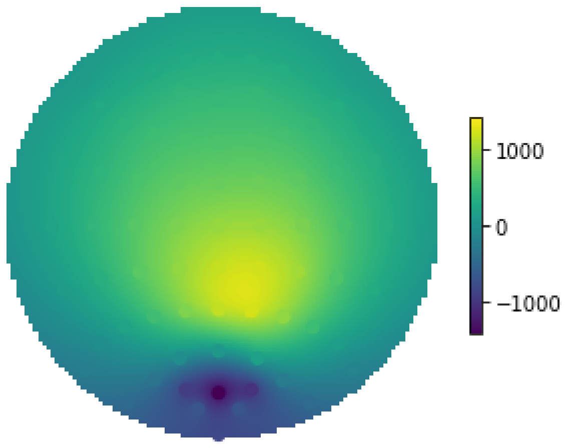
Also, we will need the montage *mnt*, which sets the position for each channel. Additionally you can find the channel labels *clab* to look at the corresponding channel names (not needed to solve the task).

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 (provided on course page).

In [15]:
```
import bci_minitoolbox as bci

em2dsm=np.load('C:/Users/OwenChang/Desktop/BCI/Leture 2/Exercise 2/Data/em2dsm.npy')
mnt=np.load('C:/Users/OwenChang/Desktop/BCI/Leture 2/Exercise 2/Data/mnt.npy')
#clab=np.load('C:/Users/OwenChang/Desktop/BCI/Leture 2/Exercise 2/Data/clab.npy')
#load error, may need reinstall python, so just skip it for the moment
```

In [16]:
```
source = 3000

p = np.array([2,5,10])

phis = np.dot(em2dsm[:,source,:],p)
# phis_normed = phis / phis.max(axis=0)

maxamp = np.max(np.abs([phis]))

plt.figure()
bci.scalpmap(mnt, phis, clim=(-maxamp,maxamp),)
plt.show()
```



In [17]:
```
max(abs(phis))
```

Out[17]:
```
1436.8014754561837
```

4/4

Task 7: The scalp potential (4 points)

In the file 'head2dsm.npy' you can find a similar file as in task 6 but this time not mapping to the electrode locations but to the vertices of the scalp surface, which is the poistion of the modeled scalp potentials.

Tasks:

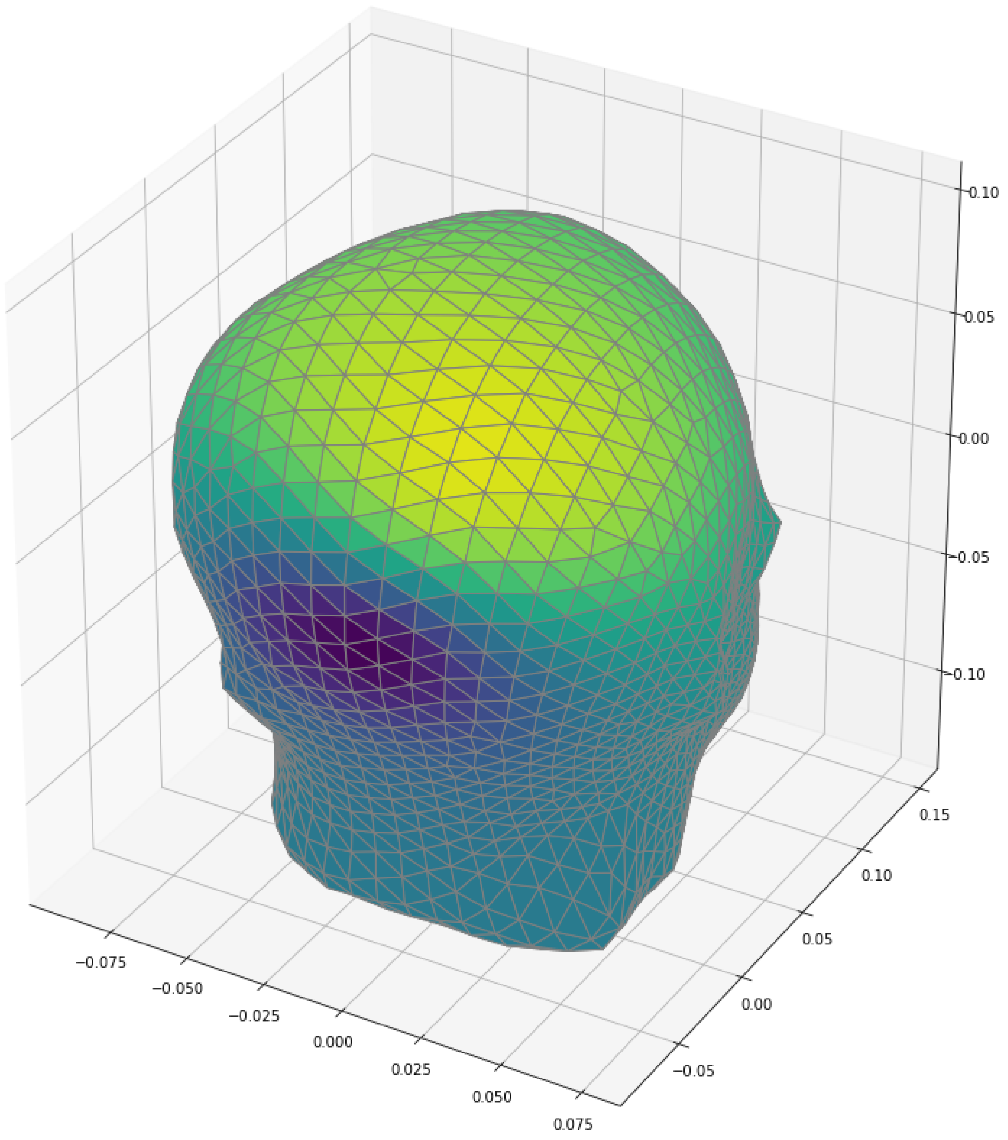
a) Chose the same dipole and dipolar moment as in task 6 and simulate the scalp potential using the file 'head2dsm.npy'.

b) Plot the distribution of the potential on the brain surface using the *plot_trisurf* function from task 5 in combination with indiviudally colored triangles. The key function to this is the method *.set_facecolors(colors)* of the object returned by plot_trisurf, where colors is of shape (NoTri,3) or (NoTri,4) depending on the colormap and the RGB definition the colormap uses.

Hint: See here how to translate scalar values to RGB colormaps: [https://matplotlib.org/api/cm_api.html (https://matplotlib.org/api/cm_api.html)](https://matplotlib.org/api/cm_api.html). As the potential is defined point-wise, for plotting the color of each triangle you can use the average of the 3 surrounding points for each triangle, which are indexed in tri. An example for the use of colomaps is also given below.

In [18]:
```
pos4=np.load('C:/Users/OwenChang/Desktop/BCI/Leture 2/Exercise 2/Data/bnd4_pos4.npy')
tri4=np.load('C:/Users/OwenChang/Desktop/BCI/Leture 2/Exercise 2/Data/bnd4_tri4.npy')

head2dsm=np.load('C:/Users/OwenChang/Desktop/BCI/Leture 2/Exercise 2/Data/head2dsm.npy')
```

In [19]:
```
n=np.mean(tri4,1)
tri_average_n = n / n.max(axis=0)

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

source = 3000
p = np.array([2,5,10])

phis = np.dot(head2dsm[:,source,:],p)

phis_avg = np.array([0]*tri4.shape[0])
for i,pos in enumerate(tri4):
    phis_avg[i] = (phis[pos[0]]+phis[pos[1]]+phis[pos[2]])/3
# phis_avg = phis_avg / phis_avg.max(axis=0)

absmax=np.max(np.abs([phis_avg]))
mappy=plt.cm.ScalarMappable()
mappy.set_clim([-absmax,absmax])
mappy.set_array([])

fig = plt.figure(figsize=[15,16])
ax = fig.add_subplot(111, projection='3d')
trisufplt=ax.plot_trisurf(x_axes4, y_axes4, z_axes4, triangles=tri4, color='yellow',edgeco

trisufplt.set_facecolors(mappy.to_rgba(phis_avg))
```


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

```
In [ ]:
```