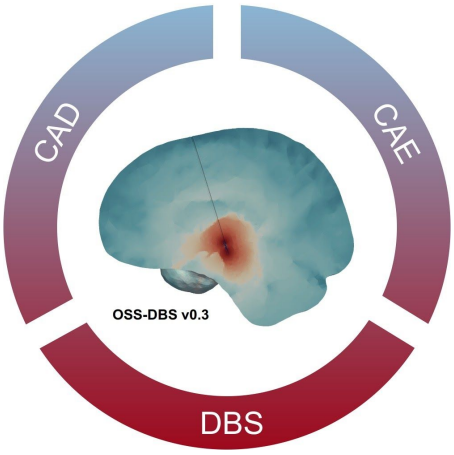
Elaine
electrically active implants
SFB 1270



OSS-DBS

Automated simulation platform for deep brain stimulation

By K. Butenko (konstantin.butenko@uni-rostock.de)

The main aim of this tutorial is to provide potential users with sufficient information so they can set up their simulations in OSS-DBS. As the platform employs highly automated routines and requires only proper definition of input parameters, I see no need to go into details of the source code. However, some aspects of the software logic and its implementation will be highlighted to give users a better understanding of the conducted simulations.

The initial motivation of this project was to create an open-source workflow that will allow us to run multiple iterative DBS simulations with no/minimal interruptions. As a first step, we have adopted and adapted an approach used in FanPy (https://bitbucket.org/ChrSchmidt83/fanpy/src/master/), where the main focus was on Python-based packages. Therefore, we decided to use Python as the programming language for the core of our platform and control all 3rd party software and data flow with Python scripts. If you take a look at the source code, you will notice that functions of the platform are not easy to call directly, as they require multiple inputs. Partially, this is done intentionally, as the platform is intended to be used with the fixed workflow. This workflow is governed by Launcher_OSS_lite.py - the main script of the platform. It is written in a self-explanatory (though not optimal) manner, and, if necessary, I encourage users to modify it to adapt the platform for their needs. However, I must warn against changing the rest of the code: it was developed by an inexperienced programmer (i.e. me) and might be difficult to understand. In this version, we strived to ensure code's reliability and efficiency, sometimes sacrificing its clearness. This shortcoming will be fixed in the next releases.

Content

1. Quick setup with Docker (in collaboration with Max Schröder)

OSS-DBS consists of a number of open-source software products, and, from our experience, their installation is not a simple task, especially for users who are not confident with Linux. For this reason, and to port the platform to other operating systems, it was decided to create a docker image. The main idea is to encapsulate software services into a so-called container that can be easily shared and run on different hosts, e.g. a server in the clinical research environment or a laptop for demonstration purposes. To set up the platform, Ubuntu and macOS users simply need to run commands below in their terminal. They will install docker (https://www.docker.com/) on the system, pull the repository of the project and the docker images. Instructions for Windows users will be provided in the next release.

```
//be careful, lines with small spacing are actually on the same line!

//First, we will install docker
//For Ubuntu 18.04
sudo apt install apt-transport-https ca-certificates curl software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu
bionic stable"
```

```
sudo apt update

sudo apt install docker-ce
```

//For Ubuntu 17.10

```
sudo apt install apt-transport-https ca-certificates curl
software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu artful stable"

sudo apt update

sudo apt install docker-ce
```

//For Ubuntu 16.04

```
sudo apt-get install apt-transport-https ca-certificates curl
software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu
xenial stable"

sudo apt-get update

sudo apt install docker-ce
```

//to test the installation

```
sudo docker run hello-world
```

//now we will set up our docker image

```
//clone the repository (use provided username and password for Elaine gitlab)
//here we assume that oss_platform will be saved in the home directory
git clone https://gitlab.elaine.uni-rostock.de/kb589/oss_platform.git

cd oss_platform

//here change *** to the provided username.
sudo docker login --username=*** gitlab.elaine.uni-rostock.de:4567

//enter password for sudo (if asked) then password for gitlab

sudo docker pull gitlab.elaine.uni-rostock.de:4567/kb589/oss_platform:base

sudo docker pull gitlab.elaine.uni-rostock.de:4567/kb589/oss_platform:fenics19

sudo docker pull gitlab.elaine.uni-rostock.de:4567/kb589/oss_platform:platform
```

//To run as a sudo user

```
sudo docker run --name OSS_docker --volume ~/oss_platform:/opt/oss_platform -it --rm
gitlab.elaine.uni-rostock.de:4567/kb589/oss_platform:platform python3
Launcher_OSS_lite.py
```

//if sudo approach is fine with you, skip the section below

==========

//Though Docker should be setup with sudo rights, we can run it also from users

//For this, we need to setup a docker group and add users (with sudo account)

```
sudo groupadd docker
```

```
sudo usermod -aG docker konstantin     //"konstantin" is the user's account
```

// reboot the system to apply the changes, alternatively logout and login again

// clone the repository using the user's account (**use provided username** and password for Elaine gitlab)

```
git clone https://gitlab.elaine.uni-rostock.de/kb589/oss_platform.git
```

```
cd oss_platform
```

//you do not need to pull images again though

//recompile the last stage of the docker image to make it user specific (make sure you run the command in the repository folder (oss_platform/))

```
docker build --build-arg OSS_UID=$(id -u) --build-arg OSS_GID=$(id -g) -t
custom_oss_platform .
```

```
docker run --volume ~/oss_platform:/opt/oss_platform -it --rm custom_oss_platform
python3 Launcher_OSS_lite.py
```

==========

//remove `python3 Launcher_OSS_lite.py` just to launch the container

// if to run with MPI for mesh adaptation (experimental version)

// to launch from sudo user

```
sudo docker run --name OSS_docker --volume ~/oss_platform:/opt/oss_platform
--cap-add=SYS_PTRACE -it --rm
gitlab.elaine.uni-rostock.de:4567/kb589/oss_platform:platform python3
Launcher_OSS_lite.py
```

```
// to launch from user

docker run --volume ~/oss_platform:/opt/oss_platform --cap-add=SYS_PTRACE -it --rm
custom_oss_platform python3 Launcher_OSS_lite.py

//now you are running the docker container
//to exit the container
Exit

//to interrupt the container
CTRL+C

//to launch the container the next time, you just need to use one of the commands with
docker run and enter the password for sudo if requested.

====================================macOS=========================================

//install Docker Desktop (you might need to register, only on Windows and macOS)

//contact your system administrator in case of proxy/protected networks

//make sure python3 is installed

//use the same commands as for Ubuntu starting from //now we will set up our docker
image

//run docker as a sudo user

//set up maximum allowed number of CPUs/Memory usage in Docker Desktop

================================================================================
```

In the current state, our docker image does not provide a graphical interface. We recommend users to install Paraview 5.4.1 (https://www.paraview.org/), a powerful tool for scientific visualization, and use it to display .pvd and .csv files. In Visualization_files/ we prepared a collection of Paraview scripts that load the files and make screenshots (stored in Images/). The scripts can be executed directly in Paraview with Python Shell or through the terminal of your system with `python "script_name".py` (make sure that you execute from oss_platform/OSS_platform/)

---

2. GUI tree for input parameters (in collaboration with Sosoho-Abasi Udongwo)

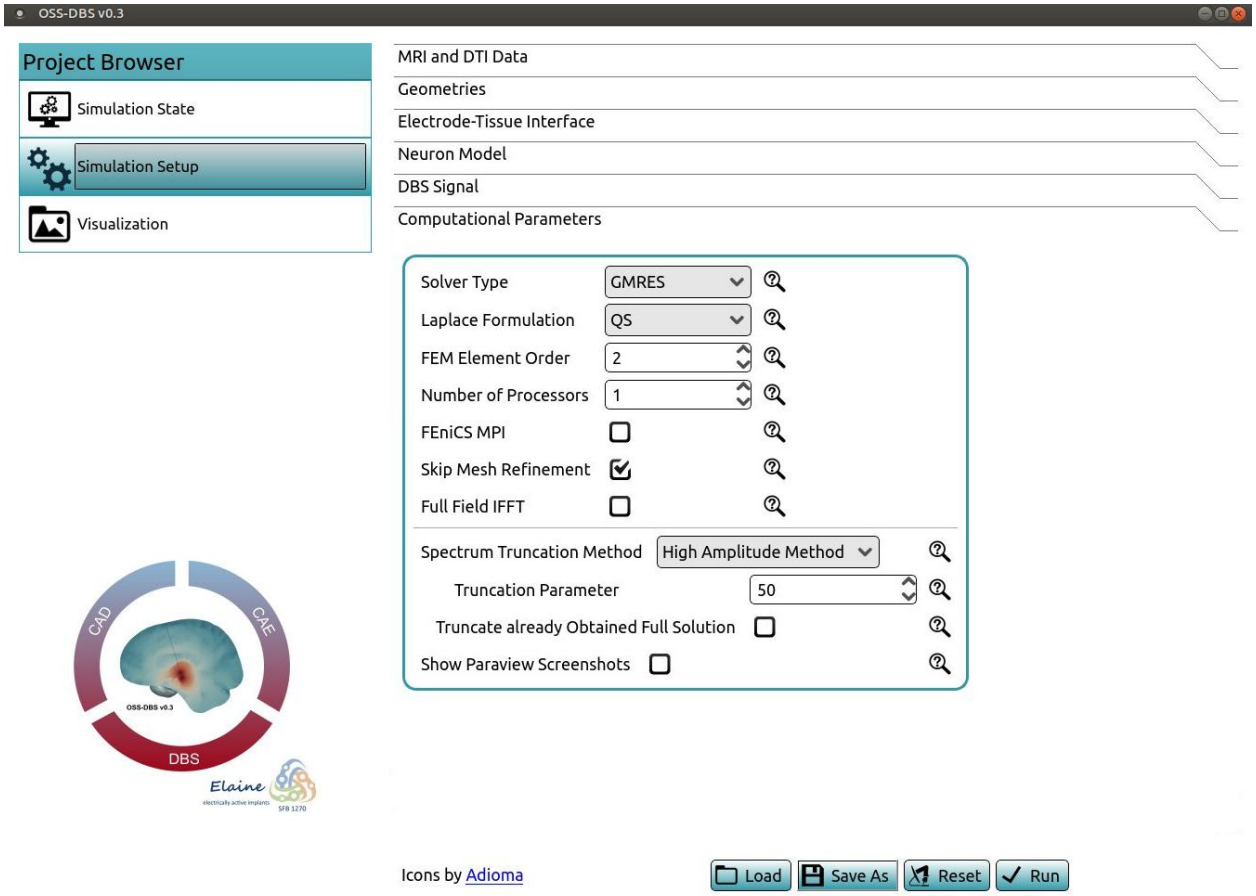
The platform reads the input parameters from a Python dictionary. As the number of the parameters is high, and some of them might not be clear for users, it is not convenient to define the dictionary manually. Therefore, we have developed a GUI tree where users can set up their simulations. Next to every parameter entry, there is a note sign that will display info about the corresponding parameter. The GUI tree has already some logical restrictions, e.g. disabling of the constant phase element for current-controlled stimulation, and in the next release we will implement a foolproof identification mechanism. To install and launch the GUI tree, Ubuntu and macOS users need to run the following commands

```
//Note: GUI tree should be launched in the user's system, not in the docker container!
So, open a new terminal.
sudo apt-get install python3-pyqt5 (python3 -m pip install PyQt5 on macOS)

cd

cd oss_platform/OSS_platform            //if oss_platform is in home directory

python3 GUI_tree_files/AppUI.py
```

After you set up all parameters, click "Save As" and here you have two options:

1) use a custom name just to save this setup
2) use the name "GUI_inp_dict" and overwrite the existing file. This dictionary is used as the input to the platform.

We also prepared several dictionaries for predefined simulations that will demonstrate some functionality of OSS-DBS. In Section *10. Examples* it will be explained how to launch them.



3. Custom input data

Unfortunately, some small preparations might be necessary before launching user's custom simulations. Although, we provide sets of MRI/DTI/DWI data, brain geometries and neuron arrays*, they might not cover the needs of the investigator. In this section, I will explain which formats of the imported data the platform supports.

a) MRI/DTI/DWI data

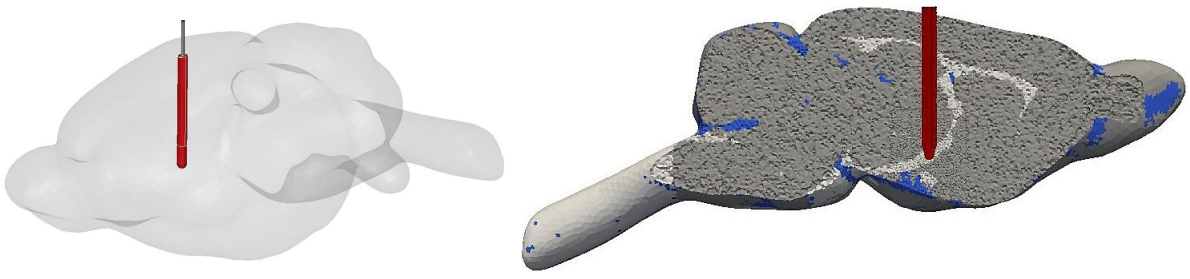
In OSS-DBS MRI/DTI/DWI data are required to define a distribution of dielectric properties of brain tissue. MRI data must be segmented into grey, white matter and cerebrospinal fluid (GM, WM, CSF) and can be provided in NIfTI format with the correct affine transformation (recommended option) and voxel sizes (stored in the header), or in a .txt file as for COMSOL interpolation function (look Example_files/Dummy_MRI.txt in the repository. Real data examples for both formats are also available in the repository: Example_files/Neural_activation_in_basal_ganglia/icbm_avg_152_segmented.nii.xz - NIfTI format Example_files/Direct_VTA_estimation/SRI24_Rohlfing_segmented.txt.zip) - .txt format

The platform will reassign material indices of GM, WM, CSF to 3,2,1, reserving 0 for the default material, and the dielectric properties of each voxel will be defined based on its material index and the simulation frequency according to Gabriel et al. All processed MRI and metadata are saved in MRI_DTI_derived_data/.

To take into account anisotropic nature of brain tissue, we apply conductivity tensors. They are extracted from DTI/DWI data, the formatting requirements are similar to MRI data (look Example_files/Dummy_DTI.txt and Example_files/Stimulation_rat_model/Waxholm_Papp_ROI_DTI.txt[1] in the repository. For a nifti file, make sure that the fourth axis (DTI directions) has order XX,XY,XZ,YY,YZ,ZZ). Here I must note that the platform will not scale the tensors, as there are different approaches available in the literature, and users will have to take care of it before importing the data.

b) External geometries

In the platform, external geometries can be used to define a computational domain of the simulation, e.g. brain geometry. Usually, this is an overkill, as the electric field induced by DBS is mostly focused around the electrode lead, especially in the case of a multipolar stimulation. Therefore, the platform offers users to construct an elliptic approximation of the region of interest or of the whole brain. However, it might be convenient to use an accurate geometry, when the electrode lead is implanted close to the brain's border or for visualization purposes.

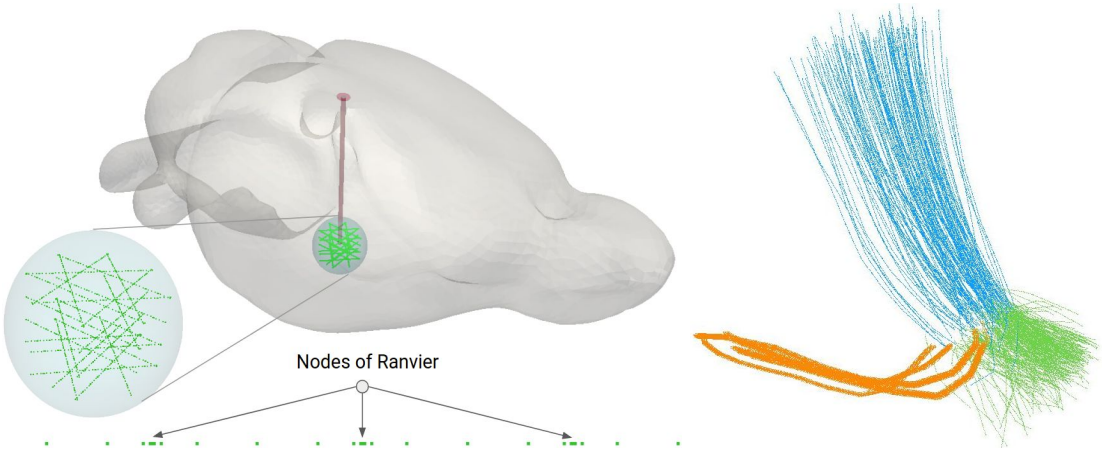


Left: an electrode and its encapsulation layer inserted in an imported .brep model of the rat brain. Right: a distribution of tissue (including the encapsulation layer) mapped onto the FEM model of the brain.

Currently, OSS-DBS has a support of .brep, .iges and .step files. Geometries should be created in *mm* and in the same coordinate space as the provided MRI data.

c) Neuron (axon) models

While users can set up neuron models and their arrays inside the platform (e.g. for volume of tissue activated estimation), it is more common to import an already available model/array. From the perspective of electric field calculations, a neuron model is merely a collection of points where the electric potential will be probed to use it as an input for neural simulations. To import a custom neuron model, users need to save coordinates of its compartments in a .csv file (delimiter=' '), and they will be offered to build a regular or irregular array using the model. Another option is to import a whole neuron array defined externally, for example, built basing on a fiber tractography analysis. I recommend to use .h5 files (https://www.h5py.org/) with multiple data sets, where each data set contains neurons of the same morphology and population[1] (e.g. axon models located on the same pathway, with the same number of nodes of Ranvier and fiber diameter). If all neurons have the same morphology, the array can be also stored in a .csv file[2]. As in the case with external geometries, coordinates should be in *mm* and in the space of the provided MRI data. All manipulations with neuron arrays are conducted by Neuron_models_arangement_new.py and processed data (and metadata) are stored in Neuron_model_arrays/.



Arrays of axons in OSS-DBS. Left: a regular array of straight axons defined in the platform and deployed in ROI. Right: an imported array of three axonal populations with different lengths (and the number of nodes of Ranvier).

4. Geometry and initial mesh generation in SALOME (in collaboration with Pham HaTrieu To)

Since the platform is intended for optimization and uncertainty quantification of DBS, it was crucial to develop an automated solution for geometry and mesh generation. SALOME conveniently offers to export a CAD study in a Python-like script, which is easy to parametrize. We prepared a collection of such scripts for different DBS electrodes, saved as "electrode name"+_profile.py. Profiles are stored in Electrode_files/ along with DBS_lead_position_V10.py and Profile_Process_V6.py that are used to generate files in format "electrode name"+_position.py (stored in the main directory). These are the files that contain simulation specific information such as dimensions of the provided MRI data, implantation point, encapsulation thickness, etc. SALOME executes position files and generates a computational domain that consist of the following geometries:

1) Brain / it's approximation
2) Region of interest (ROI), which encompasses previously defined neuron array

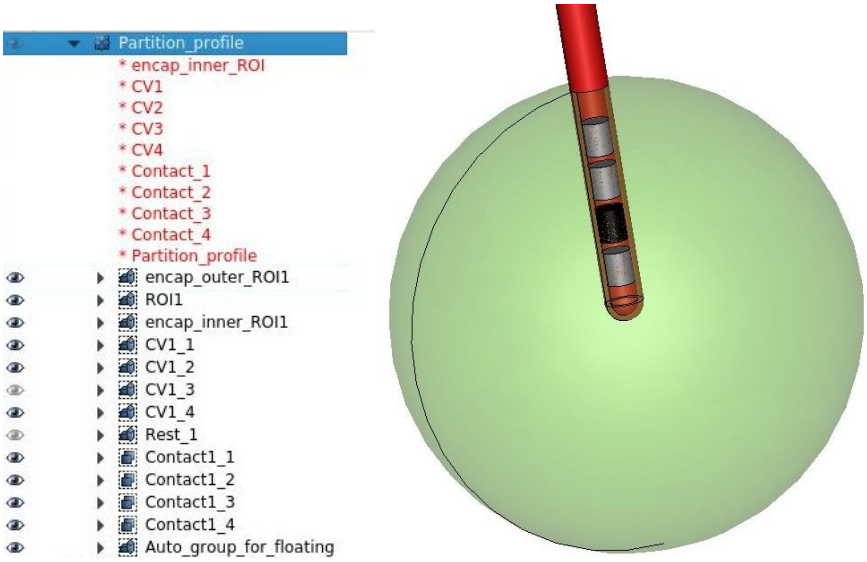
[1] Check out example for seven axonal populations of the basal ganglia-thalamo-cortical loop with different lengths in Example_files/Neural_activation_in_basal_ganglia/Basal_ganglia_axons_corrected.h5

[2] Check out HDP_STN_Motor_cortex.csv

3) Encapsulation layer inside ROI
4) Encapsulation layer outside ROI

The electrode itself is subtracted from the domain as it is assumed to be extremely conductive in comparison to the brain tissue and its surface (excluding contacts) is assumed as electric insulation. Additionally, we mark surfaces of electrode contacts that will be assigned potential/current. If there are other contacts on the electrode, they will be marked and modelled as a volume with extremely high conductivity and permittivity to simulate floating potential (look virtual permittivity method). All these geometrical entities are collected in a partition object, and for each we create a geometrical group (see the image below).

A special case is a current-controlled stimulation with more than 2 active contacts (including the ground). In this case we use superposition of fields, obtained from simulations with each contact and the ground while other contacts are modelled as floating potentials. Due to the above mentioned marking procedure, multicontact stimulations require a re-generation of the geometry if we change the configuration of active contacts. However, it is not needed if only values of the DBS signal are changed.



Some geometrical groups defined in the partition: ROI (green), encapsulation layer in and outside of the ROI (red), three contacts where potential/current will be assigned (grey) and a floating contact (black). Note CV1_… that are volumes of the electrode contacts required to simulate floating potentials with the virtual permittivity method.

In SALOME, we use NETGEN module to generate the initial mesh. For each aforementioned geometrical entity we also defined a submesh, which parameters we encourage to modify depending on the problem (see below). Note that these parameters should be changed in the electrode profile and not in the position files. Mesh is exported in .med format and then converted to .msh using GMSH. Then "dolfin-convert" command is used to obtain 3 .xml files with the mesh itself, subdomains for volumes (_physical_region.xml) and subdomains for surfaces (_facet_region.xml)

that will be imported to FEniCS (the files are stored in Meshes/). All of the described processes are managed in the platform by script CAD_Salome.py.

```
NETGEN_1D_2D_3D_3 = Mesh_1.Tetrahedron(algo=smeshBuilder.NETGEN_1D2D3D,geom=ROI1)
Sub_mesh_7 = NETGEN_1D_2D_3D_3.GetSubMesh()
NETGEN_3D_Parameters_4 = NETGEN_1D_2D_3D_3.Parameters()
NETGEN_3D_Parameters_4.SetMaxSize( 25.4615 )
NETGEN_3D_Parameters_4.SetSecondOrder( 0 )
NETGEN_3D_Parameters_4.SetOptimize( 1 )
NETGEN_3D_Parameters_4.SetFineness( 2 )
NETGEN_3D_Parameters_4.SetMinSize( 0.00328242 )
NETGEN_3D_Parameters_4.SetUseSurfaceCurvature( 1 )
NETGEN_3D_Parameters_4.SetFuseEdges( 1 )
NETGEN_3D_Parameters_4.SetQuadAllowed( 0 )
```

Snippet of submesh definition for ROI from Electrode_files/Medtronic3389_profile. It is quite usual to adjust maximum size of the FEM elements (SetMaxSize), or change the fineness of the submesh, especially if the geometrical surface of the electrode or the encapsulation layer are distorted. However, one should remember that the model will undergo physics-based refinement during the simulation, so the initial mesh should only preserve the correct geometry.

Users with experience in CAD modeling can also create their own profile files following instructions in Electrode_files/How_to_create_profile.docx and taking a look at the prepared profiles. In case the electrode lead you want to create for simulations is widely used in the DBS community, feel free to contact us, and we will add it to the platform.

5. Adjustment of neuron arrays

Neuron arrays, composed of neuron models and imported or defined within the platform, undergo an adjustment process in order to ensure their rather realistic placement. First of all, all models with compartments lying outside of the computational domain are deleted. If compartments of a model intersect with the encapsulation layer, the model is considered damaged by the implantation and also removed. Additionally, neurons passing through voxels with CSF are disregarded as unrealistic. For large MRI data sets, extraction of CSF voxels in the vicinity of the neuron models might take some time, but the obtained array (MRI_DTI_derived_data/"MRI_name"_voxel_array_CSF.csv) will be reused for the next simulations unless the imported MRI data is changed. The remaining models, which will be used for simulations in NEURON, are stored in Neuron_model_arrays/Vert_of_Neural_model_NEURON.csv. All manipulations mentioned in this section are also conducted by Neuron_models_arangement_new.py

6. Mesh refinement

Since the platform is intended to be used for optimization and UQ studies, efficiency of the computations is a major factor. To keep a low amount of FEM elements in the mesh while achieving a required accuracy, we implemented two mesh refinement algorithms. Firstly, the platform estimates the effect of high conductive tissue in the MRI data. For brain models, it is CSF. The algorithm goes over the mesh and refines elements, which midpoints points lie in CSF voxels. Refinement is done until the longest edge of the elements reaches a specified size (in the GUI tree *Minimum Element to Voxel Ratio*). Then, the distribution of electric potential on the neuron compartments is computed and it will be used as the benchmark. The algorithm will be comparing it with the results obtained on meshes with descending *Minimum Element to Voxel Ratio* until maximum deviation on the compartments is below the specified threshold (in the GUI tree *Rel. Deviation for CSF Refinement*). Usually, the CSF effect is only prominent

in the vicinity of the electrode, and therefore users can restrict the refinement area (in the GUI tree *CSF Refinement Area*).

I recommend new users to run a simulation described in Example 10.1 to get a better understanding of the mesh refinement procedure. On the snippet below you can see an example of output in the terminal during CSF refinement (here the scaling factor is *Minimum Element to Voxel Ratio*). The simulations with scaling factors 32/16 were skipped as no elements were actually refined on the initial mesh. You might also notice that the number of degrees of freedom is less than the number of elements. This is due to the 1st order Lagrange functions applied only during CSF refinement (which is the first and crude refinement) to speed up the process. At the next steps of the simulation, the 2nd or 3rd order functions will be used.

```
scaling factor is  32.0
----- voxel_array_CSF for  10.0  mm vicinity was prepared in 6.24845700000003 seconds -----
refining CSF voxels with scaling  32
Number of cells after CSF refinement iteration:  176869
skipping scaling  32.0
scaling factor is  16.0
----- voxel_array_CSF for  10.0  mm vicinity was prepared in 6.256635000000017 seconds -----
refining CSF voxels with scaling  16
Number of cells after CSF refinement iteration:  176869
skipping scaling  16.0
scaling factor is  8.0
----- voxel_array_CSF for  10.0  mm vicinity was prepared in 6.25402200000002 seconds -----
refining CSF voxels with scaling  8
Number of cells after CSF refinement iteration:  184328
CSF_Subdomains_refinement file with scaling  8  was created
_________________________
dofs:  40408
N of elements:  184328
--- assembled and solved in  0  min  1  s
Integrated voltage on the ground:  0.0
Tissue impedance:  (3768.1554038300874+0j)
Current on the ground (unscaled):  6.634545903969117e-07
Deviation threshold:  0.0235509920138 V
Deviation at least:  0.0236017592314 V
At point:  103.36333215 122.31736027 64.88144437
Need further refinement of CSF
----- CSF refinement iteration took  1  min  11  s -----

scaling factor is  4.0
----- voxel_array_CSF for  10.0  mm vicinity was prepared in 6.223048999999946 seconds -----
refining CSF voxels with scaling  4
Number of cells after CSF refinement iteration:  220672
CSF_Subdomains_refinement file with scaling  4  was created
_________________________
dofs:  46988
N of elements:  220672
--- assembled and solved in  0  min  1  s
Integrated voltage on the ground:  0.0
Tissue impedance:  (3770.7059622265283+0j)
Current on the ground (unscaled):  6.630058204071151e-07
Max. deviation:  0.0182080239993 V
Deviation threshold:  0.0235669375958 V
CSF is refined enough
----- CSF refinement iteration took  1  min  23  s -----
```

At the next step, adaptive mesh refinement is conducted in three regions of the VCM: outside of the region of interest (ROI), in the vicinity of the electrode contacts and neuron models, and in the ROI, where the neuron models are placed. The reason behind this separation is a costly uniform refinement that is always conducted as the initial

iteration and as the final iteration to evaluate the mesh quality and ensure that the mesh did not converge to a local solution. Note that uniformly refined meshes are used only for evaluation of the locally refined.

More information on the adaptive mesh refinement can be found in the paper (Section *Design and Implementation/Mesh Adaptation*). Here, I just want to make some comments about the messages you might see in the terminal during the refinement.

```
----- Conducting mesh convergence study -----
At frequency:  520.0
----- Conducting it_outside_ROI -----
___________________________
Calling FFC just-in-time (JIT) compiler, this may take some time.
dofs:  253856
N of elements:  84417
Calling FFC just-in-time (JIT) compiler, this may take some time.
Calling FFC just-in-time (JIT) compiler, this may take some time.
Calling FFC just-in-time (JIT) compiler, this may take some time.
--- assembled and solved in  0  min  17  s ---
Tissue impedance:  (23381.979392082514-3667.973422790347j)
Solving for a scaled potential on contacts (to match the desired current)
--- assembled and solved in  0  min  12  s ---
Current on the ground (unscaled):  4.174078957637 55e-09 6.547953201270591e-10
Absolute error threshold on neuron compartments:  0.00344995969453 V

--- Initial uniform refinement step for it_outside_ROI
___________________________
dofs:  479774
N of elements:  166341
--- assembled and solved in  0  min  52  s ---
Tissue impedance:  (23384.670158634257-3663.57413451145 98j)
Solving for a scaled potential on contacts (to match the desired current)
--- assembled and solved in  0  min  59  s ---
Current on the ground (unscaled):  4.173862001879481e-09 6.539007292972205e-10
Av. dev. on the neuron segments:  0.000361600622545 V
Max dev. on the neuron segments: 0.00204463237391 V
Absolute error threshold on neuron compartments: 0.00344995969453
Checking current convergence for it_outside_ROI
Current deviation:  0.008352863551079996 %
Relative error threshold for current convergence:  0.12 %
Current converged

mesh meets the requirements (or no cells were marked for refinement) it_outside_ROI was completed

--- For it_outside_ROI was adapted in  3  min  45  s

----- Conducting it_on_contact -----
Absolute error threshold on neuron compartments:  0.00344995969453 V

--- Initial uniform refinement step for it_on_contact
```

On the snippet above you see a refinement iteration outside of ROI, where no refinement was actually performed as it was not necessary. In the terminal we can see some internal messages on a preprocessor step from FEniCS (Calling FFC...); the occurrence of these messages depends on the state of the precompiled FEM C++ code. We can also see the information on the degrees of freedom, the number of tetrahedrons and the impedance at the refinement frequency in the initial mesh (253856, 84417, 23381.98-3667.97j) and after its uniform refinement (479744, 166341, 23384.67-3663.57). The solutions are compared in terms of the current flowing through the grounded contact (for current-controlled case or if CPE element is enabled) and the electric potential distribution on the neuron compartments. Note that in the particular case, the unscaled current is computed that does not necessarily match the currents assigned to the contacts, but can be used for the convergence analysis due to linearity of the system (more in the paper, section *Design and Implementation/Physics of Volume Conductor Model*). The mesh is considered valid

if after the uniform refinement, the deviations in the aforementioned criteria are below a chosen threshold set up in the GUI tree (*Deviation Threshold in Relation to Voltage and Rel. Deviation of Current Magnitude*) (which is the case on the snippet). If this wouldn't be the case for the current convergence, the algorithm would refine a cell outside of the ROI if

$$\frac{\sum_{i=1}^{n}\left(\iiint_{cell}|\underline{\mathbf{J}}(\mathbf{r})|\mathrm{d}\Omega_i\right)-\iiint_{Cell}|\underline{\mathbf{J}}(\mathbf{r})|\mathrm{d}\Omega}{\sum_{i=1}^{n}\iiint_{cell}|\underline{\mathbf{J}}(\mathbf{r})|\mathrm{d}\Omega_i}>\Theta_{\mathbf{J}},$$

where **J** is the complex current, *Cell* refers to the cell in the initial mesh, *n* is the number of *cells* that comprise *Cell* in the uniformly refined mesh and Θ_J is the relative deviation threshold of current, which is set up in the GUI tree. This condition will not be evaluated in the cells with small currents as their refinement will not influence the solution (*threshold_current_in_element* in Mesh_adaptation_hybrid.py defines the minimum ratio of the current density in a cell to the maximum current density calculated in all cells, 25% by default).

(On the snippet above you can also see multiple *Calling FFC just-in-time (JIT) compiler, this may take some time* lines This refers to JIT compilations of finite element code, i.e. C++ code is generated by FFC (FEniCS from compiler) and compiled by C++ compiler. These compilations are normally required only during the first launch of a fenics project, and later they are cached. However, everytime the docker container is restarted, the cache is cleared, and the compilation will take place again. To avoid most of them during your simulation, run a dummy study first.)

If the electric potential at least on one neuron compartment deviated above assigned *Deviation Threshold in Relation to Voltage* (more info on the parameter in the GUI tree), the algorithm would go over all cells outside of ROI and check how the solution for the electric field changed for the midpoint of a cell. If the deviation was above *Rel. Deviation of E-field Adaptive Refinement,* the cell would be refined. The same way the algorithm works for the refinement in the vicinity of the electrode contacts and in the ROI, but with two special cases:

1) the solution did not converge on the compartments (neuron segments), but the refinement in the vicinity of the contacts was completed. This is the case only for the refinement around the contacts, where edges with assigned Dirichlet boundary conditions can cause "flickering" of the solution. For other domains (in and outside of the ROI), the criterion of the electric potential convergence must also be fulfilled to complete the refinement;
2) during refinement in the ROI, all cells where the solution on the neuron compartments had a large deviation are marked irregardless of the E-field solution on their midpoints.

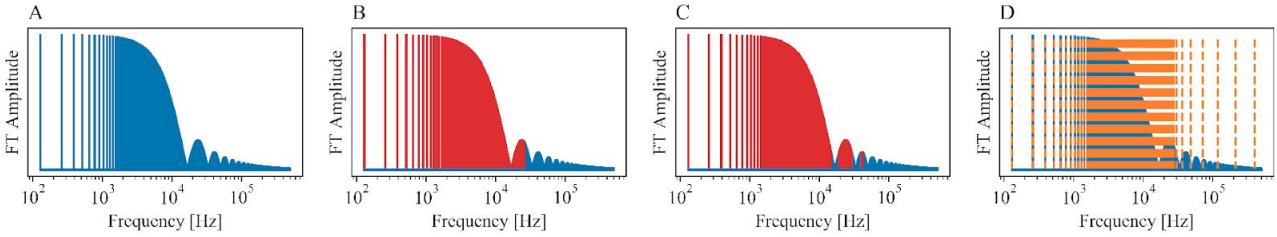
During tests of the algorithm, we have observed a situation when a solution on the uniformly refined mesh shows large deviations, but a solution at the next step (derived local refinement) does not. For this case, we accept the locally refined mesh as a new initial and apply a uniform refinement step to it.

7. Field computations in frequency spectrum

The conventionally used DBS 60 − 90 μs rectangular signal poses an additional challenge for the computational model as it necessitates to solve Laplace's equation (the EQS or QS formulations) for a multitude of time harmonic sine-wave stimuli that constitute the signal. It is important to note that grey and white matter have dispersive nature and thus demonstrate considerable changes in the dielectric properties over the frequency spectrum. Luckily, the computations on different frequencies are mutually independent, and thus we parallelized the solution of Laplace's equation. The number of parallel processes is defined by *Number of Processors* in the GUI tree. The computed solution in the frequency spectrum is then scaled by the Fourier transformation (FT) of the DBS signal and the inverse Fourier transformation (IFT) is applied to the result to obtain the electric potential in time domain. This approach was named Finite Fourier Element method (FFEM) (look Butson and McIntyre).

The platform supports two modes of FFEM: for the neuron compartments and for the whole computational domain (*Full Field IFFT* in the GUI tree). While both can be used for the further simulation with NEURON, the former is faster when the number of all neuron compartments is less than the degrees of freedom in the computational model (usually the case). The latter can be used for visualization (field in time will be saved in Animation_field_in_time/) or for VTA estimations based on derivatives of the electric potential. For these modes, field solutions at frequencies are stored in two different formats, therefore the computations have to be redone if users decide to switch (we do not store the data in both formats as it can be rather large for a disc space).

To reduce the amount of computations in frequency domain, we have implemented three spectrum truncation methods: a *Cutoff method*, where the user defines a number of frequencies that will be taken sequentially (Fig. below, B); a *High Amplitude method*, where the sine-wave stimuli with the largest contributions are considered (Fig. below, C); an *Octave Band method*, where the user chooses a frequency that will be used as a start for octave bands (Fig. below, D). While the first two methods use an explicit truncation of the signal in frequency domain, the latter preserves the full FT, with solution of (1) obtained on the center octave frequency and assigned to frequencies in this octave. Tests of these truncation methods revealed that the *High Amplitude method* is applicable for problems with a large number of neuron compartments, where the solution scaling and the IFT in the full spectrum are expensive, while the latter is more suitable for complex VCMs, where the solution of Laplace's equation requires large computational resources.



Truncation methods for the frequency spectrum of a 130 Hz 60 μs rectangular pulse. Here the spectrum is truncated to 200 highlighted frequencies, which will be used for field evaluations. A: full frequency spectrum. B: the Cutoff method (in red). C: the High Amplitude extraction (in red). D: the Octave Band (yellow dash), with the octave bands starting at 24440 Hz.

In the GUI tree (*Advanced Simulation Parameters*), users can choose a method and set up a truncation parameter (which is the number of frequencies in the truncated spectrum or the frequency after which octaves bands will be

applied). Also, *Cutoff method* and *High Amplitude method* can be evaluated against an already computed full solution without recalculations (*Truncate already Obtained Full Solution* in the GUI tree).

8. Neuron (axon) model simulation

The alleviation of symptoms in Parkinson's patients during DBS is associated with regularization of neural activity in the basal ganglia-thalamo-cortical loop. Different sources of the regularization were suggested including initiation of antidromic action potentials in projecting axons. This criterion became a standard measure of the DBS effect in computational models and it is also employed in the platform.

The axon model simulation reveals whether elicitation of action potential occurs in response to the time dependent extracellular electric potential, computed on the previous step in OSS-DBS. The original computational model of the mammalian myelinated axon was developed by McIntyre in simulation environment NEURON, and in the platform we have adapted routines previously used in FanPy (https://bitbucket.org/ChrSchmidt83/fanpy/src/master/). Users can choose the fiber diameter and number of Ranvier nodes per axon, while other parameters can be altered in the model files stored in Axon_files/. The NEURON simulation is launched through the platform and controlled by Axon_files/NEURON_direct_run.py. Since we assume a dominating effect of DBS on the neuron activity, the communication between neurons is neglected (networking effects will be added in the future studies), and thus the simulation for each axon is conducted in parallel to speed up the process. We also plan to add somatic models in the next release.

9. Miscellaneous

The platform supports application of the constant phase element (CPE) defined as in McAdams and J. Jossinet. The voltage drops over CPEs are subtracted from the initial Dirichlet BC, and the electric potential is recomputed with the new conditions (more information in paper Section II E. *Impedances of the Computational Model*). However, we do not provide values for the scaling and the exponential factors for the electrodes, so users will have to provide it in the GUI tree (*Alpha and K_s*).

In the current state, our docker image does not provide a graphical interface. We recommend users to install Paraview 5.4.1 (https://www.paraview.org/), a powerful tool for scientific visualization, and use it to display .pvd and .csv files. In Visualization_files/ we prepared a collection of Paraview scripts that load the files and make screenshots (stored in Images/). The scripts can be executed directly in Paraview with Python Shell or through the terminal of your system with `python "script_name".py` (make sure that you execute from oss_platform/OSS_platform/)

10. Examples

The following examples were tested on workstations with 4/4/24 physical cores and 8/32/256 GBs of RAM. Examples 10.2 and 10.4 can be run on machines with 8 GBs, 10.1 and 10.3 will require at least 64 GBs. You will be able to run all of the examples on 8 GBs machine if you ease the mesh refinement requirements (or skip it). Also, make sure that you have enough of disc space, as the size of the output files might reach 20 GBs.

10.1 Adaptive mesh refinement and activation in axonal populations of the basal ganglia

This example demonstrates a DBS simulation using a predefined neuron array that contains axons of direct, indirect and hyperdirect pathways, which are further divided into internuclear populations. The populations have different axonal morphologies (number of nodes of Ranvier and fiber diameter). In this study, mesh refinement is

conducted, and users can observe how the solution on axonal compartments and current over contacts converge below certain thresholds.

How to launch the example:

1) In OSS_platform/Example_files/Neural_activation_in_basal_ganglia/ locate Basal_ganlia_axons_corrected.h5 and copy it to OSS_platform/ (By default, OSS_platform/ is in oss_platform/ in your home directory)

2) Open terminal in OSS_platform/ and launch the GUI tree with "python3 GUI_tree_files/AppUI.py" (in your system, not with docker!)

3) Using the GUI tree, load Example_dict_MNI.py from OSS_platform/Example_files/Neural_activation_in_basal_ganglia/ and resave it as GUI_inp_dict.py in OSS_platform/

4) Using docker, start the simulation typing "python3 Launcher_OSS_lite.py"

The platform will process the imported MRI data of the human brain (Grabner et al), import the axon populations from Basal_ganlia_axons_corrected.h5, generate a computational domain using a specified brain approximation and electrode (St Jude 6148). Further, the axon array will be adjusted by removing axons outside of the brain or in the encapsulation layer or in cerebrospinal fluid. At the next step, the mesh will undergo CSF- and adaptive refinement, this might take some time. On the image below one can see the mesh before CSF-refinement (CSF in dark blue) and after the elements containing CSF voxels were refined until their largest edges reached the voxel size. The electric potential on the axonal compartments is computed on both meshes and compared. In this example, the deviation is below the threshold (meaning that CSF does not significantly affect the solution on axons), and the mesh on the left will be sent for the adaptive mesh refinement. The image can be reproduced by users in Paraview (in their system):

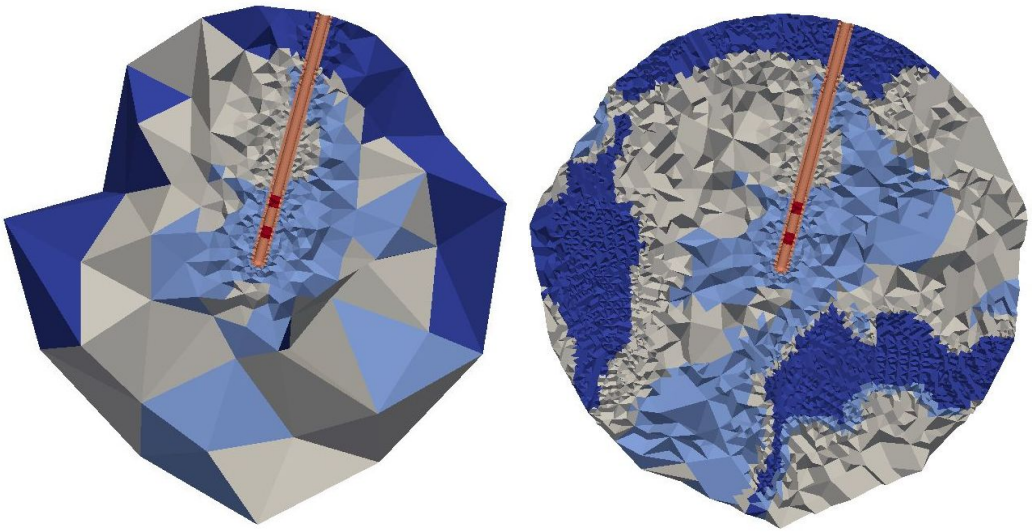
5) Open CSF_ref/CSF_Subdomains_unref.pvd and CSF_ref/CSF_Subdomains_full_ref.pvd
#Filters -> Extract Cells by Region

Additionally, we can visualize the adjusted axon array:

6) Open Neuron_model_arrays/Vert_of_Neural_model_NEURON.csv
#uncheck *Have Headers* and put " " instead of "," in *Field Delimiter Characters* and press *Apply*
#Filters->Search and type "Table to Points"
#Change *Y Column* to "Field 1"
#Change *Z Column* to "Field 2" and press *Apply*
#Click twice on the eye icon near TableToPoints1

7) Alternatively, run script Visualization_files/Paraview_connections_processed.py[3] in Python Shell of Paraview

[3] If the neuron array was provided in an .h5 file, use visualization scripts with "connections" in their names. If a .csv format was used, run scripts with "csv" to show the neuron models and their activation.



The adaptive mesh refinement will run several iterations refining elements in the vicinity of the electrode contacts, where the dirichlet boundary conditions are defined. The algorithm will monitor a) convergence of the current through contacts b) convergence of the electric potential on the axonal compartments. In this case, the refinement will be conducted due to a high current deviation on the initial mesh. The step will be completed with the following message:

```
dofs:  9520546
N of elements:  2054056
--- assembled and solved in  8  min  56  s
Integrated voltage on the ground:  0.0
Tissue impedance:  (3131.9035590000494+0j)
Current on the ground:  0.0049999999999999906
Av. dev. on the neuron segments:  0.08814619803108738 V
Max dev. on the neuron segments: 0.11806716568058029 V
Absolute error threshold on neuron compartments: 0.12807298136472464
Current deviation:  0.9075498688169737 %
Relative error threshold for current convergence:  1.2 %
mesh_new meets the requirements (or no cells were marked for refinement) it_on_contact was completed

--- For it_on_contact was adapted in  52  min  23  s

----- Conducting it_in_ROI -----
```

Here we see the evaluation of the solution at the last step of the adaptive mesh refinement in comparison to the solution obtained after uniform refinement of that mesh. Such information is displayed for every step of the process so that users can see whether the problem is converging. During iterations you can also check with Paraview which cells were marked (1 for marked):

Results_adaptive/Last_marked_cell_current_ref.pvd - cells marked using current convergence criterion

Results_adaptive/Last_Marked_cells.pvd - cells marked for refinement on the last iteration

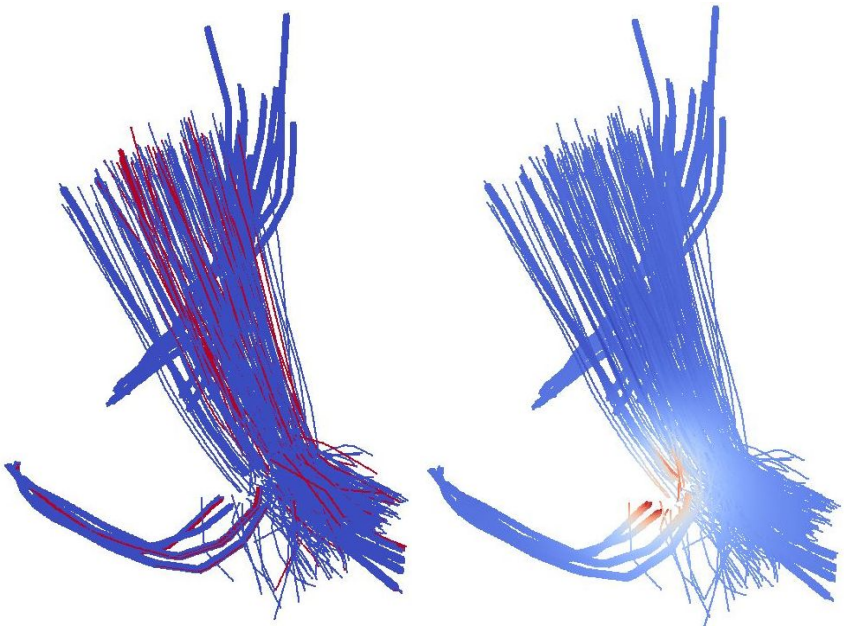
Marked_cells_start_ref_mode....pvd - cells marked for initial uniform refinement in different subregions

The adapted mesh with the tissue distribution and the computed distribution of electric potential can be visualized in Paraview using files "Field_solutions/parallel_Subdomains.pvd" and "Results_adaptive/Last_Phi_r_field_QS.pvd". Afterwards, the platform will conduct FFEM computations and the obtained potential distribution on axonal compartments in time will be used for NEURON simulations. The output of this study is activation in the axonal populations of the basal ganglia (image below, left). In can be reproduced in Paraview:

8) Open Field_solutions/Activation/Neuron_model_results...
#you will see several files with such a name, one for each population
#same procedure as with Neuron_model_arrays/Vert_of_Neural_model_NEURON.csv
#in *Coloring* choose "Field 3"

9) Alternatively, run script Visualization_files/Paraview_connections_activation.py in Python Shell of Paraview

If you plot the electric potential distribution on the axons (image below, right) at the refinement frequency (here 13130 Hz), stored in file Results_adaptive/Ampl_on_vert.csv, you will see that there is no clear relation between the proximity of the axons to the electrode contacts and their activation. This is because the activation due to an external stimulus is defined by the second derivative of the electric potential in the double cable model.



10.2 Direct VTA estimation from E-field and visualization

This example demonstrates a DBS simulation using an imported brain geometry as the computational domain. The effect of DBS is assessed with VTA, defined using a threshold for the electric field. Additionally, a Paraview file with the potential distribution in time is created.

How to launch the example:

1) In OSS_platform/Example_files/Direct_VTA_estimation/ locate Human_Brain_from_SRI24.brep.zip and SRI24_Rohlfing_segmented.txt.zip and extract the content to OSS_platform/ (By default, OSS_platform/ is in oss_platform/ in your home directory)

2) Open terminal in OSS_platform/ and launch the GUI tree with "python3 GUI_tree_files/AppUI.py" (in your system, not with docker!)

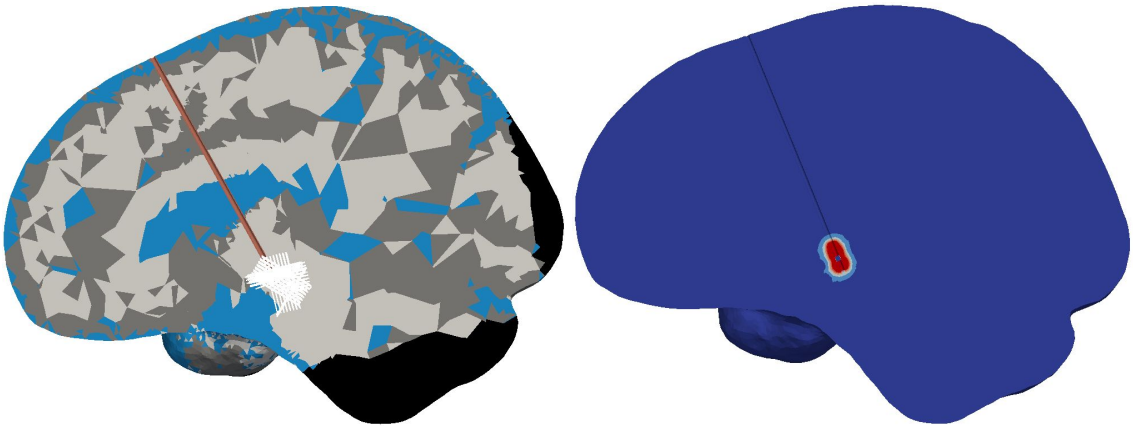
3) Using the GUI tree, load Example_dict_SRI.py from OSS_platform/Example_files/Direct_VTA_estimation/ and resave it as GUI_inp_dict.py in OSS_platform/

4) Using docker, start the simulation typing "python3 Launcher_OSS_lite.py"

The platform will process the imported MRI of the human brain (Rohlfing et al), create a regularly ordered axon array, generate a computational domain from the brain geometry and a chosen electrode (Medtronic 3389). Further, the axon array will be adjusted by removing axons outside of the brain or in the encapsulation layer or in cerebrospinal fluid. For this study, the mesh refinement routines are disabled, and after generating the signal (4 ms triangular pulse with 130 Hz repetition rate, for demonstration purposes only, plot is in Images/Signal.png) the platform will map the MRI data onto the mesh and will conduct FEM computations in the frequency spectrum of the signal. The results of the mapping and the adjusted axon array can be visualized using Paraview (in your system):

5) Open Field_solutions/parallel_Subdomains.pvd
#Use *Clipping* it at X = 73.76
#Adjust *Mapping Data* in *Color Map Editor* putting 0 to black, 1 to blue, 2 to white, 3 to grey, 5 to red

6) Open Neuron_model_arrays/Vert_of_Neural_model_NEURON.csv
#Uncheck *Have Headers* and put " " instead of "," in *Field Delimiter Characters* and press *Apply*
#*Filters->Search* and type "Table to Points"
#Change *Y Column* to "Field 1"
#Change *Z Column* to "Field 2" and press *Apply*
#Click twice on the eye icon near TabletoPoints1



The result should be as on the image above (left). Note that the axons intersecting with CSF (in blue) were subtracted. If instead of clipping you apply filter *Extract Cells by Region,* you might see that the mesh is more refined in the region of the axon array. This is due to the settings for the initial mesh created in SALOME.

After the computations are completed, the solutions in frequency domain will be scaled by the FT of the signal, and IFT will be applied to the result to obtain the solution in time domain. Then the platform will find the time step at which the electric potential magnitude is absolute maximum and will evaluate the electric field at this step. Lastly, the magnitude of the electric field will be integrated over each mesh element and normalized by its size. Elements, where the value is above the activation threshold (here 0.2 V/mm), will be summed, and the result will be the VTA. To visualize the result, we will need to use Paraview again:

7) Open Field_solutions_functions/E_field_at_stim_peak.pvd
#Use *Clipping* it at X = 73.76
#Change *Coloring* to function (f_....)
#Put 0.2 to red in *Mapping Data* (*Color Map Editor*)

The result should be as on the image above (right). Users can also assess the VTA from the divergence of the electric field (check in the GUI tree->Advanced Simulation Parameters), or conduct a NEURON simulation using the previously deployed axon array. For this, the VTA option in the GUI tree->*Advanced Simulation Parameters* should be disabled. NOTE: to try these methods you do not need to re-simulate the whole study. Only the last step *Scaling and IFFT done* should be unchecked in the GUI tree.

Users can also visualize how the electric potential distribution changes in time using Paraview:
8) Open Animation_Field_in_time/Field..vtu
#Use *Clipping* it at X = 73.76
#Press Play button
#Use *Rescale to data range* in *Color Map Editor*

8) If the VTA was assessed by a NEURON simulation, the activation can be visualized by running script Visualization_files/Paraview_csv_activation.py in Python Shell of Paraview.

10.3 Effect of model complexity on activation of axons due to DBS in rat model

Using this example, users can observe how the simulation results depend on the complexity of the computation model. Here we will take a look at the difference between EQS and QS formulation of Laplace's equation and will also estimate the effect of the locally introduced anisotropy.

How to launch the example:

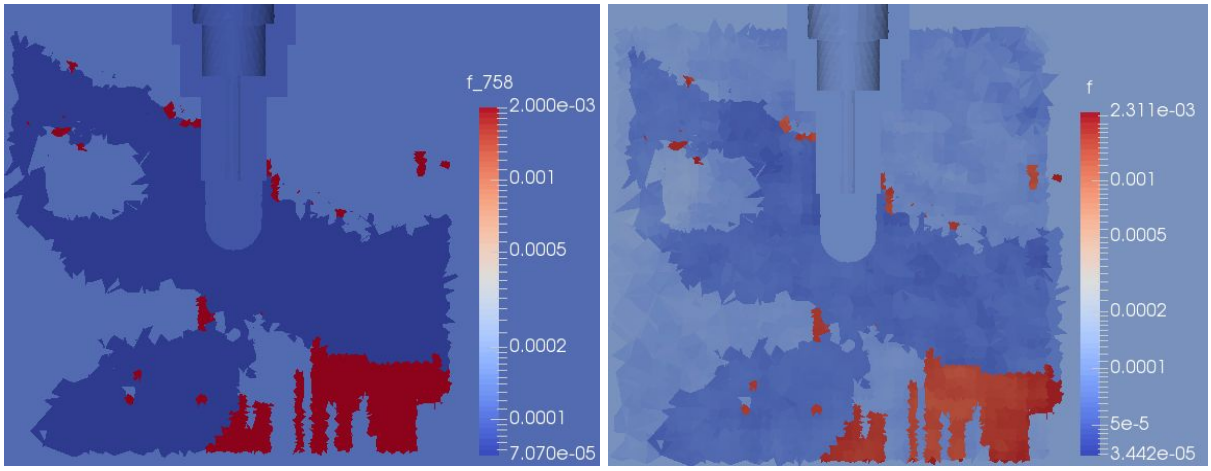
1) In OSS_platform/Example_files/Stimulation_in_rat_model/ locate Waxholm_Papp_ROI_MRI.txt and Waxholm_Papp_ROI_DTI.txt and copy them to OSS_platform/ (By default, OSS_platform/ is in oss_platform/ in your home directory)
2) Open terminal in OSS_platform/ and launch the GUI tree with "python3 GUI_tree_files/AppUI.py" (in your system, not with docker!)

3) Using the GUI tree, load Example_dict_Waxholm.py from OSS_platform/Example_files/Stimulation_in_rat_model/ and resave it as GUI_inp_dict.py in OSS_platform/

4) Using docker, start the simulation typing "python3 Launcher_OSS_lite.py"

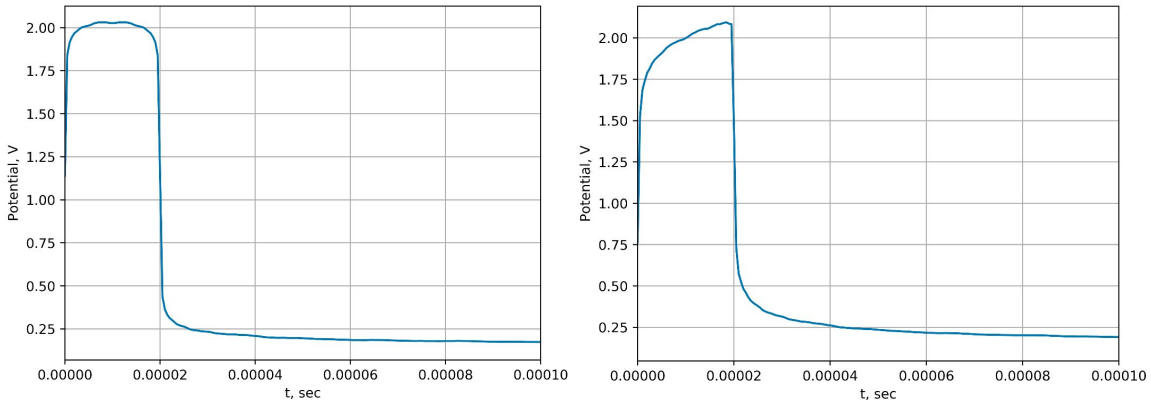
The platform will process the imported MRI data of the rat brain (Papp et al, here we use only a subset of MRI for the vicinity of the STN), create a regularly ordered axon array, generate a computational domain using a specified brain approximation and electrode (Microprobes SNEX-100). Further, the axon array will be adjusted by removing axons outside of the brain, in the encapsulation layer or in the cerebrospinal fluid. At the next step, the mesh will undergo CSF and adaptive refinement check; this might take some time. Afterwards, the platform will conduct FFEM computations and the obtained potential distribution on axonal compartments in time will be used for NEURON simulations. The output of this study is axonal activation rate in the ordered array placed in the STN region (which can be used for VTA estimations).

For this example, we provide a small set of diffusion tensors in the vicinity of the STN derived from Waxholm DTI data (Papp et al). By default, the tensors are employed in the simulation, but users can estimate the influence of anisotropy on the results by clearing out *DTI Data Name* in the GUI tree (it will disable the diffusion tensors). The scaled conductivity distribution by tensor components can be visualized in OSS_platform/Tensors/c00_mapped.pvd (for XX component), etc.



Effect of anisotropy on tissue conductivity (S/mm) in the vicinity of the rat STN. Left: scalar conductivity distribution based on the tissue segmentation. Right: conductivity in transversal direction obtained using tissue segmentation and diffusion tensors.

It is of particular interest to assess qualitatively the effects of the QS formulation on the electric potential, as these physics change the tissue voltage response in the computational model. It was previously reported that for current-controlled stimulations the capacitive term in Laplace's equation leads to the rise of the voltage during the DBS pulse. Setting *Laplace Formulation* in the GUI tree to EQS, users will observe this effect in the plot of electric potential in time for the first compartment of the first axon (plot is saved in Images/Signal_convoluted_1st_point.png, shown below on the right). In case of the QS formulation, the tissue response preserves the signal shape (shown below on the right). The occurred difference between the formulations leads to a different axonal activation rate, which is especially prominent for short DBS signals with a larger contribution of high frequency components.



Tissue voltage response during rectangular DBS pulse calculated using QS (left) and EQS (right) formulations of Laplace's equation. In both cases, tissue works as a low-pass filter, but for the latter we can also observe capacitive charging, introduced by the imaginary term of the equation.

10.4 Fast run

This is a small example that we use to test installations of the platform and to quickly check its functionality (it is also a default simulation after the platform is downloaded, but only with one processor used). The example does not require large computational resources, and thus can be run on ordinary machines.

How to launch the example:

1) Open terminal in OSS_platform/ and launch the GUI tree with "python3 GUI_tree_files/AppUI.py" (in your system, not with docker!)

2) Using the GUI tree, load Example_dict_quick_test.py from OSS_platform/Example_files/ and resave it as GUI_inp_dict.py in OSS_platform/

3) Using docker, start the simulation typing "python3 Launcher_OSS_lite.py"

During the simulation, the platform will process the imported MRI data of the human brain (Grabner et al), create an ordered axon array, generate a computational domain using a specified brain approximation and electrode (Medtronic 3389). Further, the axon array will be adjusted by removing axons outside of the brain or in the encapsulation layer or in cerebrospinal fluid. At the next step, the mesh will undergo CSF and adaptive refinement, but the deviation thresholds are put to 10% and 5%, so no refinement will be needed. Afterwards, the platform will conduct FFEM computations using QS formulation, and the obtained potential distribution on axonal compartments in time will be used for NEURON simulations. The output of this study is axonal activation rate in the ordered array placed in the STN region.

* For DBS in humans: segmented atlases in MNI (ICBM152 non-linear 6th generation symmetric Average Brain, Grabner et al) and SRI (SRI24 multichannel atlas of normal adult human brain structure, Rohlfing et al) spaces, roughly classified fiber tracts of the basal ganglia-thalamo-cortical loop in MNI space (basing on Marek et al, Ewert et al, using atlases from Ewert et al, Chakravarty et al, Fan et al), brain geometry from SRI space courtesy of C.Schmidt). For DBS in rodents: Waxholm atlas space (Papp et al) based segmentation of the whole brain, in the vicinity of the subthalamic nucleus (including scaled DTI values) (courtesy of A. Andree) and classified fiber tracts.

11. References:

C. R. Butson and C. C. McIntyre, "Tissue and electrode capacitance reduce neural activation volumes during deep brain stimulation," *Clinical Neurophysiology*, vol. 116, pp. 2490 – 2500, Oct 2005.

M. M. Chakravarty et al., "The creation of a brain atlas for image guided neurosurgery using serial histological data," *NeuroImag*,. vol. 30(2), pp. 359 – 376, 2006.

S. Ewert et al., “Toward defining deep brain stimulation targets in MNI space: A subcortical atlas based on multimodal MRI, histology and structural connectivity,” *NeuroImage*, vol. 170, pp. 271 – 282, 2018.

L. Fan et al., “The Human Brainnetome Atlas: A New Brain Atlas Based on Connectional Architecture,” *Cerebral Cortex.* vol. 26(8), pp. 3508 – 3526, 2016.

S. Gabriel, R. W. Lau, and C. Gabriel, “The dielectric properties of biological tissues: III parametric models for the dielectric spectrum of tissues,” *Physics in Medicine & Biology*, vol. 41, pp. 2271 – 2293, 1996.

G. Grabner, A. L. Janke, M. M. Budge, D. Smith, J. Pruessner, and D. L. Collins, "Symmetric atlasing and model based segmentation: an application to the hippocampus in older adults," *Med Image Comput Comput Assist Interv Int Conf Med Image Comput Comput Assist Interv*, vol. 9, pp. 58 – 66, 2006.

K. Marek et al., “The Parkinson Progression Marker Initiative (PPMI),” *Progress in Neurobiology*, vol 95(4), 629 – 635, 2011.

E. T. McAdams and J. Jossinet, “Physical interpretation of Schwan’s limit voltage of linearity,” *Medical and Biological Engineering and Computing*, vol. 32, pp. 126 – 130, Mar 1994.

C. C. McIntyre, A. G. Richardson, and W. M. Grill, “Modeling the Excitability of Mammalian Nerve Fibers: Influence of Afterpotentials on the Recovery Cycle,” *Journal of Neurophysiolog*y, vol. 87, pp. 995 – 1006, Feb 2002.

E. A Papp, T. B. Leergaard, E. Calabrese, G. A. Johnson GA, and J. G. Bjaalie, “Waxholm Space atlas of the Sprague Dawley rat brain,” *NeuroImage,* vol. 97, 374 – 386, 2014.

E. A Papp, T. B. Leergaard, E. Calabrese, G. A. Johnson GA, and J. G. Bjaalie, “Addendum to "Waxholm Space atlas of the Sprague Dawley rat brain"”, *NeuroImage,* vol. 97, 374 – 386, 2014.

T. Rohlfing, N. M. Zahr, E. V. Sullivan, and A. Pfefferbaum, “The SRI24 multichannel atlas of normal adult human brain structure,” *Human Brain Mapping*, vol. 31, no. 5, pp. 798 – 819, 2010.