

# FanPy Manual

Christian Schmidt  
cschmidt18057 at gmail.com

May 31, 2017

## Contents

|          |  |          |
|----------|--|----------|
| <b>1</b> | <b>Introduction</b>  | <b>1</b> |
| <b>2</b> | <b>Tutorials</b>   | <b>2</b> |
| 2.1      | Generating a model of the subthalamic nucleus as target area . . . | 2        |
| 2.2      | Generating a multi-compartment model geometry with Salome . . .    | 4        |
| 2.3      | Meshing the model geometry . . . . .                               | 9        |
| 2.4      | Solving the field equation with FEnICS . . . . .                   | 13       |
| 2.5      | Analytical Validation of the field solution . . . . .              | 17       |
| 2.6      | Computing the neural activation extent during DBS . . . . .        | 20       |

## 1 Introduction

FanPy (Field and Neuron Python) is a Python package for computing the field distribution during deep brain stimulation (DBS) and estimating the neural activation resulting from the applied electrical stimulus. Although the model pipeline is based on an application for deep brain stimulation, it is not limited to it and can be easily transferred to various bioelectrical applications. The Python package was used to estimate the neural activation extent in models of deep brain stimulation rendering different post-operative phases and target area properties, while reducing the computational expense by employing an approximation of the neural activation extent by field threshold values determined from the threshold-distance relationship of the applied axon models. For further details please refer to Schmidt, C and van Rienen, U, Adaptive Estimation of the Neural Activation Extent during Deep Brain Stimulation, 2017, arXiv:1705.10478 [q-bio.NC]

The code is highly experimental and should be used with caution and a proper

amount of healthy suspicion.

FanPy is published under the GNU Lesser General Public License version 3.

## 2 Tutorials

### 2.1 Generating a model of the subthalamic nucleus as target area

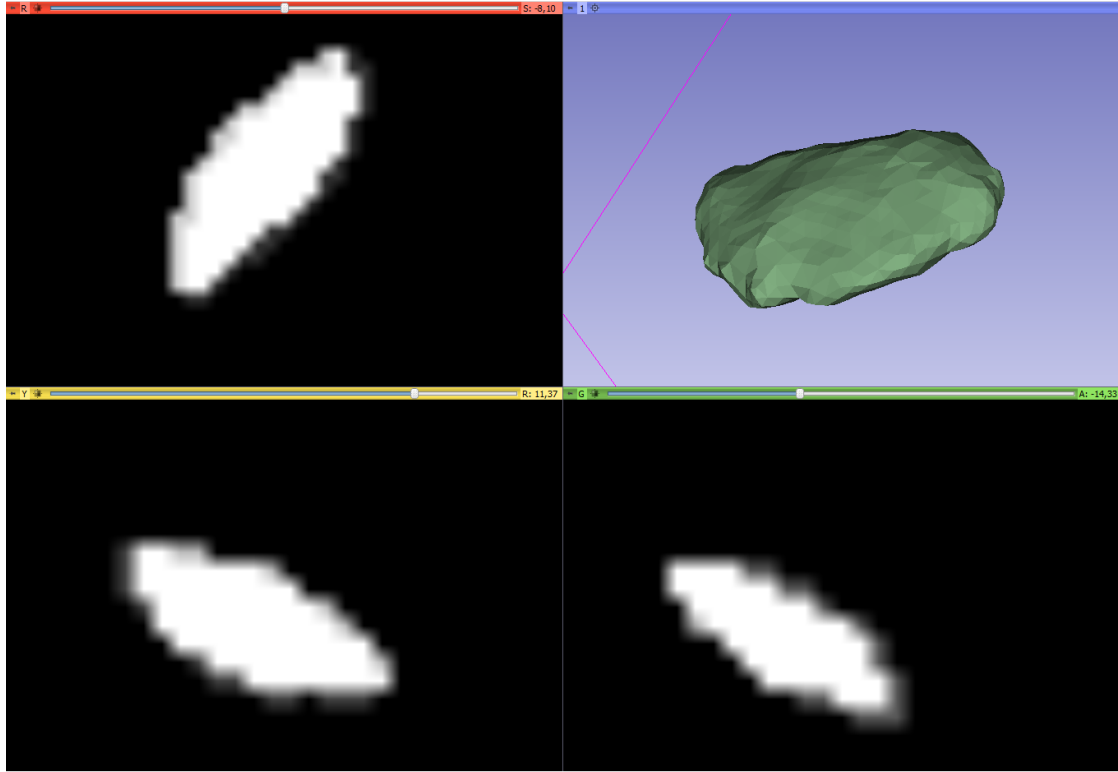


Figure 1: Thresholding of the STN mask and the resulting surface model in 3D Slicer.

The subthalamic nucleus (STN) is a common target area during DBS. To incorporate the STN in a volume conductor model for DBS, the geometry information of the STN in an appropriate format, such as STEP, BREP, STL has to be available. If this geometry is not available, it has to be extracted from imaging data. Here we use the data available from the study

E. A. Accolla, J. Dukart, G. Helms, N. Weiskopf, F. Kherif, A. Lutti, R. Chowdhury, S. Hetzer, J. D. Haynes, A. A. Kühn, and B. Draganski, Brain tissue properties differentiate between motor and limbic basal ganglia circuits, Hum Brain Mapp, vol. 35,

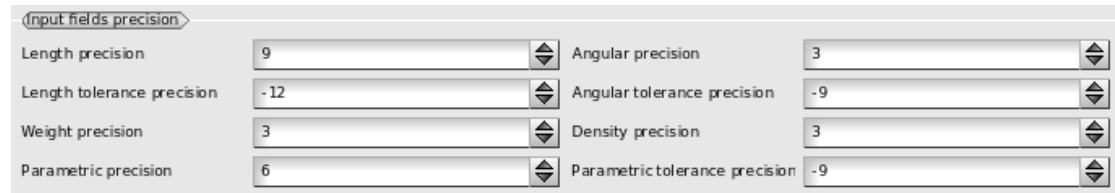
pp. 5083-5092, Oct 2014.

which is also available in the freely available software Lead DBS and provides a data set rendering the right and left STN and its sensorimotoric, limbic, and associative functional zones. The extent of the right and left STN are stored in `gm_mask.nii`, which is processed with the open-source software 3D Slicer. First the right STN is extracted from the data using **Converts→Crop Volume** and choosing a volume around it. Next the data is segmented by using **Segementation→Editor** and applying the Threshold Effect with a range from 0.25 to 1.00 (Fig. 1). The resulting label map is used to build a surface model of the STN using **Surface Models→Model Maker**. The resulting surface model is exported as an `.stl` (see `./files/STN.stl`).

## 2.2 Generating a multi-compartment model geometry with Salome

This tutorial shows the generation of DBS electrode lead model surrounded by multiple tissue compartments by using the open-source software Salome.

**Salome Preferences** The standard length unit in Salome is meters. Since the DBS electrode is in the scale of millimeters the precision in Salome has to be modified. An alternative would be to design the geometry in Salome in meters and treating it as it would be in millimeters by scaling the mesh afterwards. In this tutorial the first approach is used, designing the electrode in millimeters in the Salome meter scale. For this, the precision under File→Preferences→Geometry→Input fields precision has to be modified (Fig. 2).



| Input fields precision     |     |                                |    |
|----------------------------|-----|--------------------------------|----|
| Length precision           | 9   | Angular precision              | 3  |
| Length tolerance precision | -12 | Angular tolerance precision    | -9 |
| Weight precision           | 3   | Density precision              | 3  |
| Parametric precision       | 6   | Parametric tolerance precision | -9 |

Figure 2: Preferences to be adjusted in SALOME in order to create the model in SI units.

**The DBS electrode lead model** The DBS electrode lead model represents a Medtronic 3387 DBS electrode lead, which comprises a cylindrical lead with a diameter of 1.27 mm and four electrode contacts with a height of 1.5 mm and a spacing of 1.5 mm. The electrode will be positioned in a bounding box (representing the surrounding tissue) with an edge length of 100 mm with the second electrode contact of the DBS electrode lead located in the center of the bounding box. We start by first creating a cylinder with height of 1.5 mm and 1.27 mm representing the second electrode contact (Fig. 3). The cylinder gets translated  $-0.75$  mm along the z axis to be centered to the coordinate origin using Transformation→Translation. Then we translate the cylinder again (which creates a copy of the cylinder) by  $-1.5$  mm which represents the insulation between the second electrode contact and the first electrode contact. This is repeated to model the first electrode contact (Fig. 4).

The insulation below the first electrode contact has a rounded tip. We model this by creating a cylinder with a height of 0.75 mm and half sphere with a diameter of 1.27 mm. The half sphere is created by first creating a sphere and a cylinder with a height of 0.635 mm (the radius of the sphere) and using Operations→Boolean→Cut

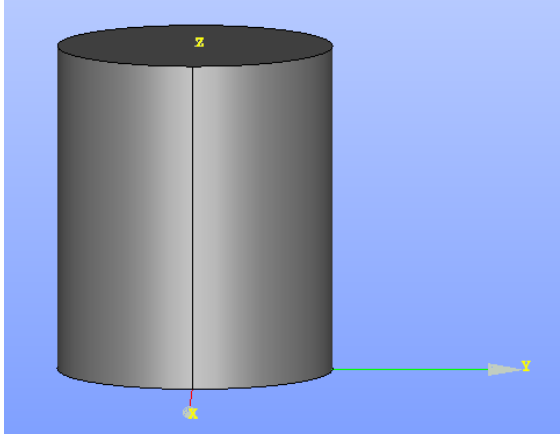


Figure 3: Single cylinder

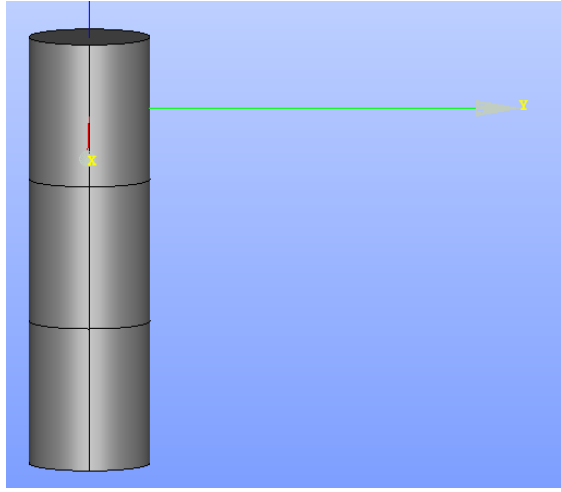


Figure 4: Merged cylinders

to cut the cylinder out of the sphere, resulting in the half sphere. The cylinder and the half sphere are unified in one object using **Operations**→**Boolean**→**Fuse** and translated to the bottom of the first electrode contact (Fig. 5). Translating the second electrode contact cylinder multiple times, the third and fourth electrode contact and their insulation are created (Fig. 6).

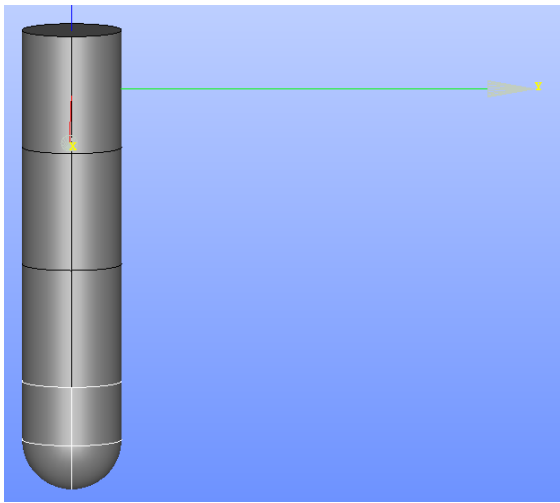


Figure 5: Merged cylinders with electrode lead tip

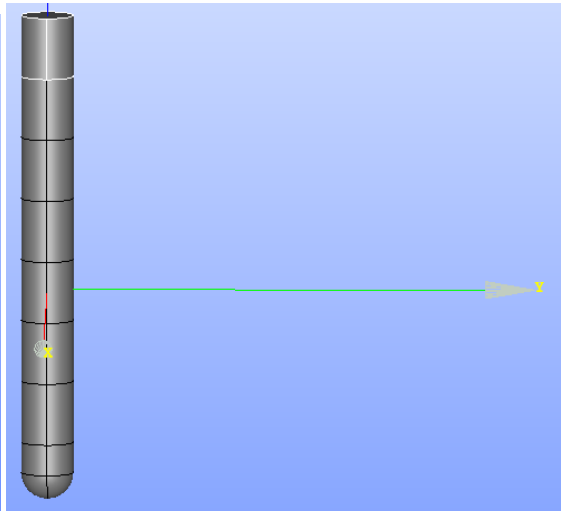


Figure 6: Electrode contacts with electrode lead tip

The lead of the electrode is a cylinder with a height of 43.25 mm and positioned at the top of the fourth electrode contact. The separated objects are then merged by

using **New Entity**→**Build**→**Compound**. We still have duplicated faces at the intersection between two cylinders, which we can remove by using **Repair**→**Glue Faces** with a tolerance of  $1 \cdot 10^{-7}$  (Fig. 7).



Figure 7: DBS electrode geometry

In the next step an encapsulation layer around the electrode with a thickness of 0.2mm is modeled using a cylinder with a diameter of 1.67mm and a height of 54.5mm. The half sphere at the encapsulation layer tip is created like that at the electrode tip, but using a diameter of 0.835mm. Finally, the cylinder and the half sphere are fused to form the encapsulation layer. Since the electrode geometry overlaps with the encapsulation layer object, they have to be separated from each other in order to obtain an electrode geometry surrounded by an encapsulation layer geometry. For this the electrode geometry is cutted from the encapsulation layer object, creating a compound from the resulting object and the electrode geometry and finally glueing their faces using the functions applied in the previous steps (Fig. 8). The target area is represented by a model of the sub-



Figure 8: DBS electrode with encapsulation layer geometry

thalamic nucleus, which modeling is described in the tutorial 2.1. The resulting .stl file is loaded into Salome and scaled using **Operation**→**Transformation**→**Scale Transform** by a factor of 0.001 to transform the STN into the Salome meter coordinate system. The surface model of the STN is transformed into a solid using **New Entity**→**Build**→**Solid** and translated with the negative coordinates of the center of the motor region to position the STN at the second active electrode contact (which is in the origin of the coordinate system) (Fig. 9). To build the distinct compartments of the electrode, the encapsulation layer, the STN, and the bounding box, the geometries have to be separated from each other. First, the encapsulation layer object is cutted from the STN object resulting in the STN geometry. Then the electrode model with the encapsulation layer is partitioned using the STN geometry using **Operations**→**Partition**. This is required to generated common edges and faces at the intersection between STN and encapsulation layer. In the next step, the electrode geometry including the encapsulation layer and the STN geometry are merged into a compound and duplicate faces are removed by glueing their

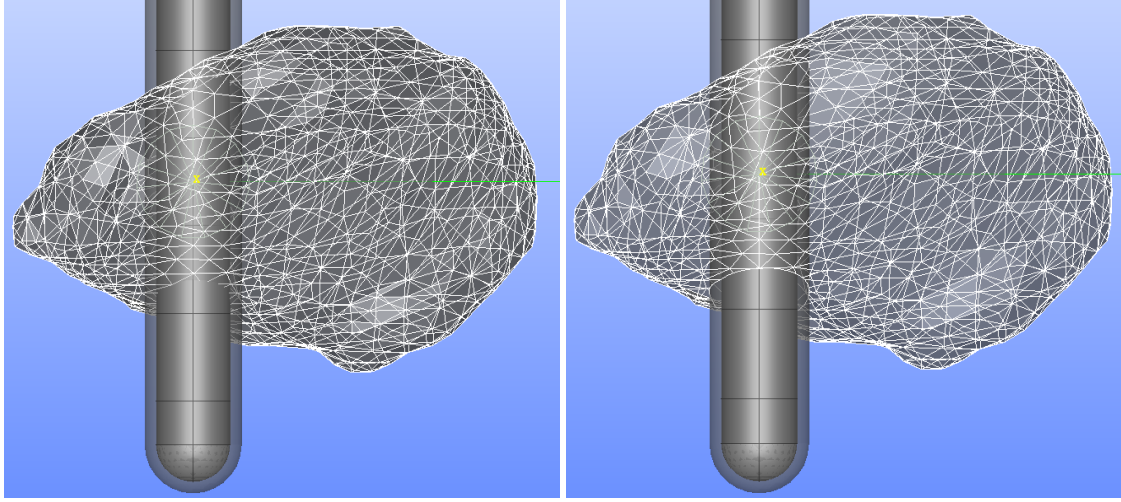


Figure 9: Imported STN geometry, which is not water-tight connected to the electrode geometry. Figure 10: STN geometry is water-tight connected and glued to the electrode with encapsulation layer geometry.

common faces (Fig. 10).

In the last step the bounding box, representing the surrounding tissue is modeled by creating a Box (cube) with an edge length of 100 mm and translating its center to the origin of the coordinate system. The electrode geometry including the encapsulation layer and the STN is cutted from the bounding box resulting in the tissue box geometry, which is merged to a compound together with the electrode-encapsulation-STN geometry. Finally their common faces are glued, which results in the final model geometry of the DBS electrode surrounded by an encapsulation layer, the STN as target area and the whole model geometry surrounded by a tissue bounding box. The model geometry is exported in BREP format (see `./files/dbs_electrode_with_stn_motor.brep`). The whole Salome model pipeline is saved in hdf5 format (see `./files/dbs_electrode_stn_motor.hdf`) (Fig. 11).

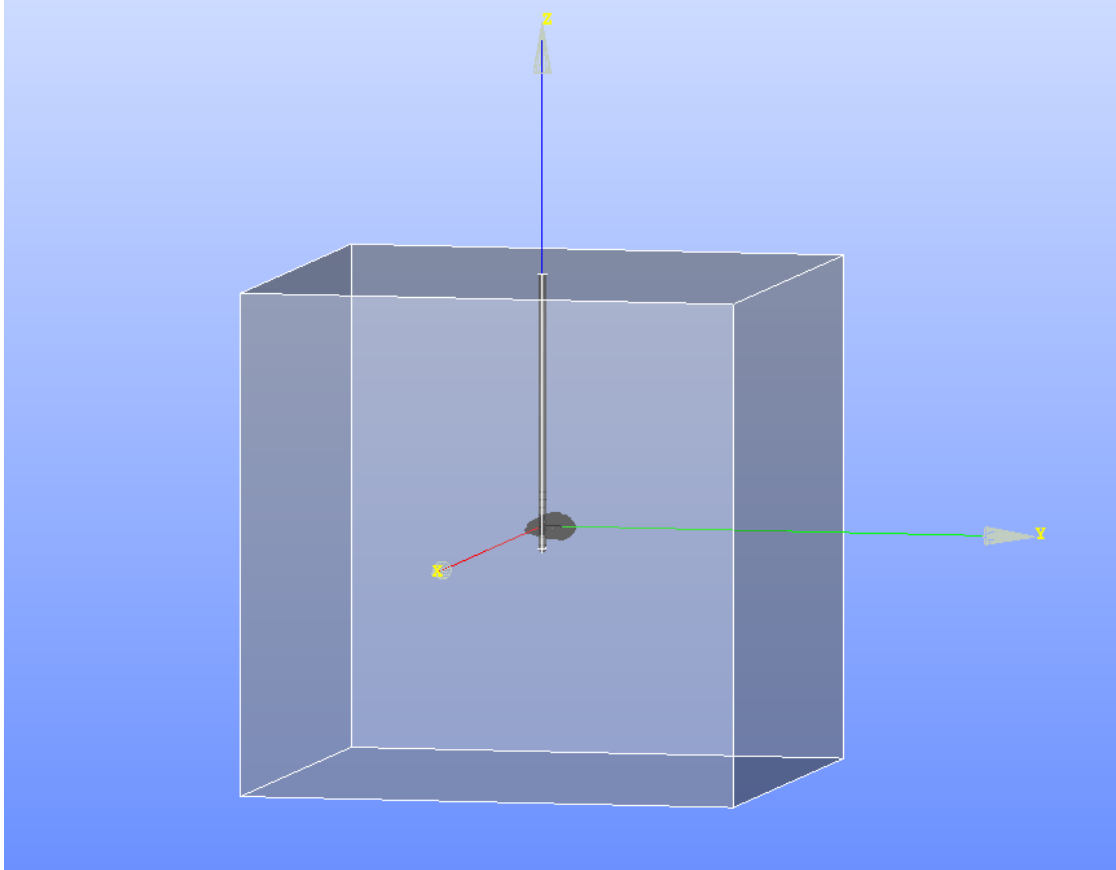


Figure 11: Full model geometry of the DBS electrode, encapsulation layer, STN, and the surrounding bounding box.



## 2.3 Meshing the model geometry

For the meshing of the geometry the freely available software Gmsh is used. The meshing of the geometry requires several steps, from identifying subdomains (model compartments) and special boundaries (compartment surfaces) on which later the material properties and boundary conditions will be applied, to manually refining mesh sizes in specific subdomains or on specific boundaries. For this purpose a .geo file is created in which the model geometry created in 2.2 is imported as a .brep file:

```
Merge "dbs_electrode_with_stn_motor.brep";
```

A way to identify the specific subdomains (volumes) of the model compartments and the special boundaries is the visibility tool under Tools->Visibility. In the Tree browser the model parts matching a certain Volume, Surface, or Point are shown when selected (Fig. 12). For an easier use in the later field models, the information on the subdomains and boundaries can be written as comments in the .geo file.

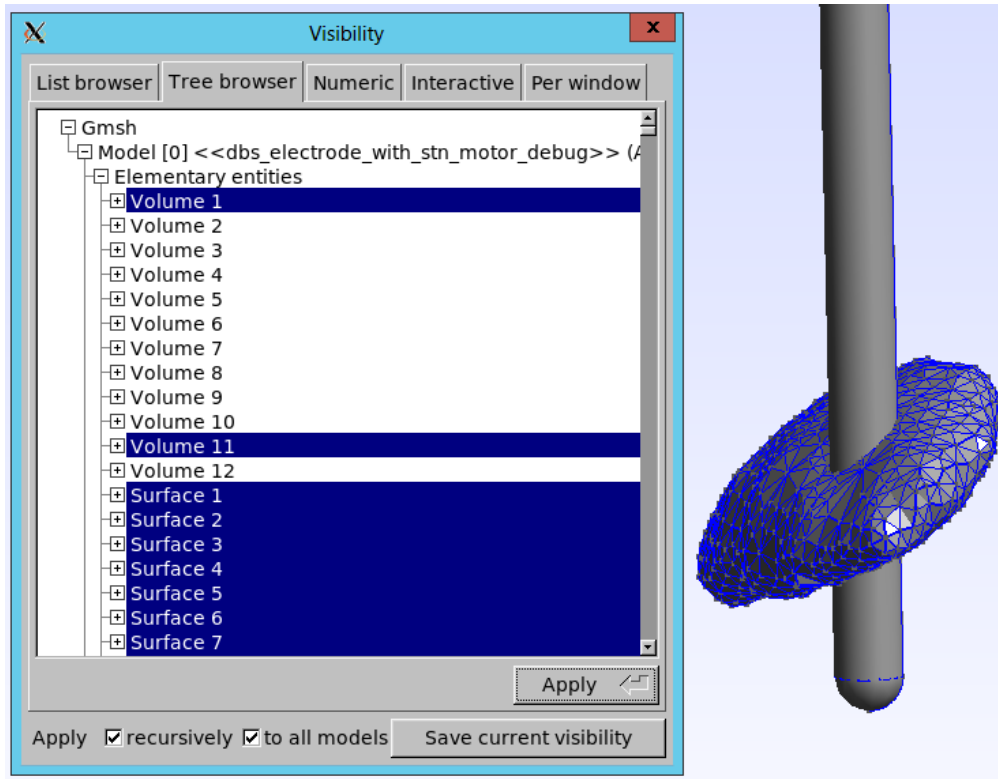


Figure 12: Setting certain model structures visible to identify the corresponding surfaces and points in Gmsh.

```
// STN: Volume 1
// Contact 4: Volume 9
// Contact 3: Volume 7
// Contact 2: Volume 2
// Contact 1: Volume 4
// Insulation Lead : Volume 10
// Insulation 4-3 : Volume 8
// Insulation 3-2 : Volume 6
// Insulation 2-1 : Volume 3
// Insulation 1-t : Volume 5
// Encapsulation : Volume 11
// Bounding Box : Volume 12
```

For the assignment of the material properties, it is not necessary to distinguish between model compartments with the same material properties. For example, the insulation between electrode contact 1 and 2 and electrode contact 2 and 3 does have the same material properties and can be merged to a so called Physical Group, which is a function provided by Gmsh to unify different subdomains to one label. **It is mandatory to label all sub-volumes of the model geometry with Physical Volumes, since only those get meshed by Gmsh.**

```
// Boundaries
// Contact 1
Physical Surface(1) = {1386};
// Contact 2
Physical Surface(2) = {1381};
// Contact 3
Physical Surface(3) = {1392};
// Contact 4
Physical Surface(4) = {1396};
// Ground
Physical Surface(10) = {1404:1409};
```

```

// Subdomains (Caution! Have to be complete, because only physical
// groups get meshed and exported by gmsh!)
// Contacts
Physical Volume(100) = {2, 4, 7, 9};
// Insulation
Physical Volume(101) = {3, 5, 6, 8, 10};
// Encapsulation
Physical Volume(102) = {11};
// Tissue
Physical Volume(103) = {12};
// STN
Physical Volume(104) = {1};

```

For the manual mesh refinement, the **Characteristic Length** of model points can be set in Gmsh, which determines the maximum element size for this point and neighbouring entities of higher dimension (lines, surfaces, volumes). For example, the surface of the electrode contacts, the encapsulation layer, and the target area should be meshed with a finer resolution, since the applied stimulation signal causes the largest changes in the field solution in this region. The surrounding tissue bounding box can be meshed with a coarser resolution. The **Visibility Tool** can be used again to identify these points. For this a certain surface or volume can be selected in the **Tree browser** (Model entities) which highlights the corresponding points as well (Fig. 13).

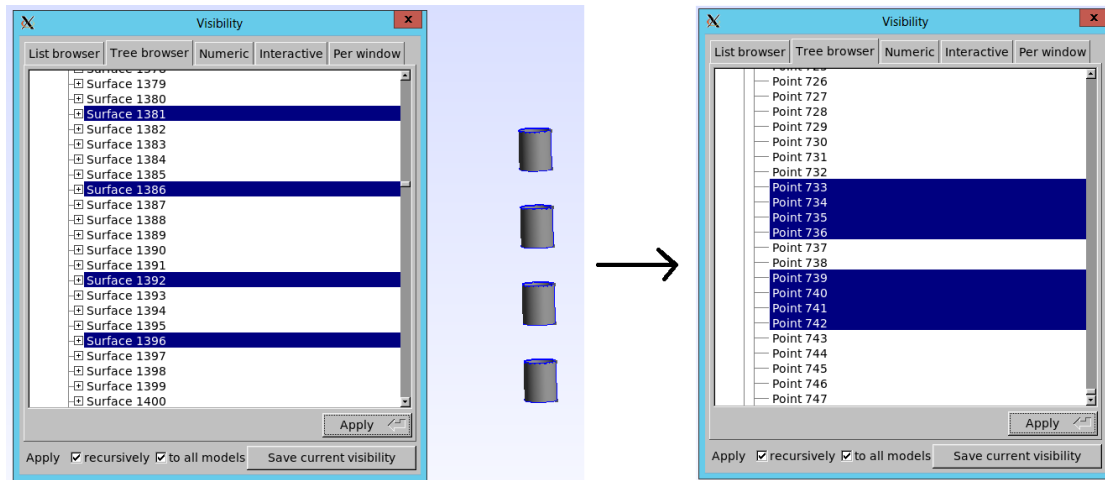


Figure 13: Selecting surfaces in Gmsh to obtain the corresponding points for applying mesh refinement via **Characteristic Length**.

```
// Characteristic Length for Bounding Box
Characteristic Length {744:754} = 5.0e-3;

// Characteristic Length for STN
Characteristic Length {1:732} = 0.2e-3;

// Characteristic Length for Encapsulation
Characteristic Length {28:29, 36:38, 40:87, 733:746} = 0.1e-3;

// Characteristic Length for Insulation
Characteristic Length {733:743} = 0.1e-3;

// Characteristic Length for Contacts
Characteristic Length {733:736, 739:742} = 0.1e-3;
```

In addition to setting the **Characteristic Length** at the desired points, it is also possible to define a region in which the mesh should be refined to a certain value.

```
Field[1] = Box;
Field[1].VIn = 0.5e-3;
Field[1].VOut = 5e-3;
Field[1].XMax = 0.01;
Field[1].XMin = -0.01;
Field[1].YMax = 0.01;
Field[1].YMin = -0.01;
Field[1].ZMax = 0.01;
Field[1].ZMin = -0.01;
Background Field = 1;
```

Currently FEniCS can only import mesh elements of order 1. To restrict the meshing in Gmsh to this order and to further optimize the mesh quality the following parameters are set.

```
Mesh.ElementOrder = 1;
Mesh.Optimize = 1;
```

All the information is stored in a .geo file (see ./files/dbs\_electrode\_with\_stn\_motor.geo), which can be meshed with Gmsh via command line (Fig. 14).

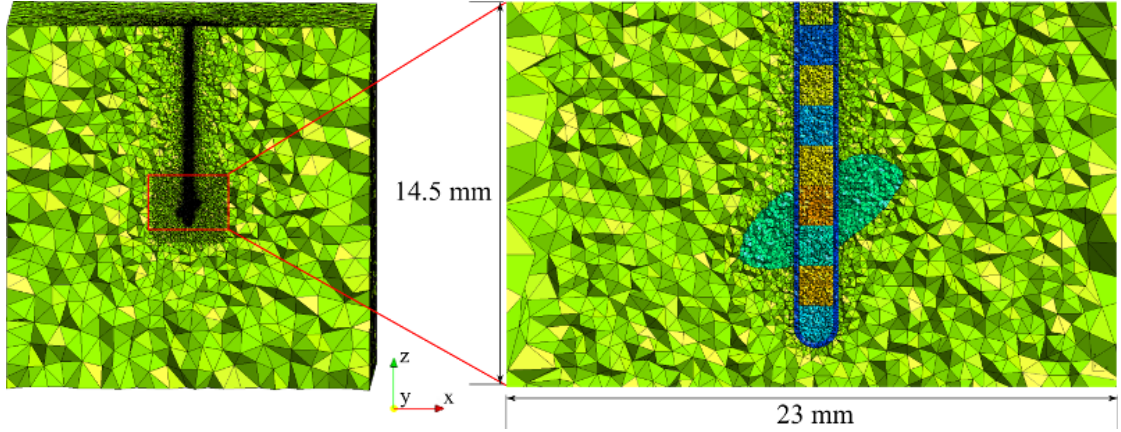


Figure 14: Resulting mesh of the DBS electrode model illustrating the manual mesh refinement at the electrode contacts, encapsulation layer, and the STN.

## 2.4 Solving the field equation with FEnICS

The Field Model in **FanPy** is currently able to compute the solution to the field equation for stationary current field problems (SF) and electro-quasistatic field problems (EQS). The stationary current field problem is based on solving the Laplace equation for heterogeneous, purely resistive material properties, i.e. defined by an electrical conductivity. The electro-quasistatic field problem is based on solving the frequency-domain Laplace equation for heterogeneous, complex-valued material properties, i.e. defined by an electrical conductivity and relative permittivity at a specific angular frequency. The boundary conditions comprise Dirichlet boundary conditions, i.e. used for setting the potential at the surface of the active electrode contact and ground at the exterior boundary surfaces of the bounding box, and Neumann boundary conditions, i.e. setting insulation on the surfaces of the insulating parts of the electrode lead. For further details refer to the paper referenced in the readme of the **FanPy** repository. The mesh of the model geometry on which the field equation with boundary conditions is solved is generated from the created `.geo` file using Gmsh (see 2.3) and converting it into a format readable by FEniCS using the command line tool `dolfin-convert` included in the FEniCS installation.

**Setting up the Model Parameters using a XML Configuration File** To use a model geometry in the `.geo` format created with Salome and Gmsh with the Field Model in **FanPy**, a `.xml` configuration file is required, which serves, on the one hand, to determine the default model parameters required to solve the field equation for the corresponding model and, on the other hand, to validate the physical volumes and surfaces defined in the `.geo` file and the model parameters

(material properties, boundary conditions) set in the .xml configuration file. For the DBS model created in 2.2 and 2.3, an example .xml configuration file is given by:

```
<fmodel>
  <geofile>dbs_electrode_with_stn_motor</geofile>
  <geofile_debug>dbs_electrode_with_stn_motor_debug</geofile_debug>
  <subdomains>
    <subdomain name="electrode_contacts" material="platinum-
    iridium" physid="100" group="electrode"/>
    <subdomain name="electrode_lead" material="insulation" physid
    ="101" group="electrode"/>
    <subdomain name="encapsulation" material="encapsulation"
    physid="102"/>
    <subdomain name="brain" material="brain" physid="103"/>
    <subdomain name="stn" material="stn" physid="104"/>
  </subdomains>
  <boundaries>
    <boundary name="contact1" physid="1"/>
    <boundary name="contact2" physid="2"/>
    <boundary name="contact3" physid="3"/>
    <boundary name="contact4" physid="4"/>
    <boundary name="ground" physid="10"/>
  </boundaries>
  <materials>
    <material name="platinum-iridium" conductivity="1e7"/>
    <material name="insulation" conductivity="1e-7"/>
    <material name="encapsulation" conductivity="0.15"/>
    <material name="brain" conductivity="0.30"/>
    <material name="stn" conductivity="0.20"/>
  </materials>
  <boundary_conditions>
    <boundary_condition name="active_electrode" type="dirichlet"
    boundary="contact2" value="1.0"/>
    <boundary_condition name="ground" type="dirichlet" boundary="
    ground" value="0.0"/>
  </boundary_conditions>
</fmodel>
```

- fmodel: root node for the FEniCS model.
- geofile: name of the .geo file without file ending

- `geofile_debug`: name of the `.geo` file for debug mode (e.g. coarser mesh version)
- `subdomains`: root note for the subdomain parameters (sub-volumes)
  - `subdomain`: node for subdomain parameters
    - \* `name`: attribute for the name of the subdomain
    - \* `material`: attribute for the name of the material defined at ‘materials’
    - \* `physid`: attribute for the id of the corresponding physical group as defined in the `.geo` file
    - \* `group`: (optional) attribute for merging subdomains to a group
- `boundaries`: root note for the boundary parameters (surfaces)
  - `boundary`: node for boundary parameters
    - \* `name`: attribute for the name of the boundary
    - \* `physid`: attribute for the id of the corresponding physical group as defined in the `.geo` file
- `materials`: root note for the materials and their default values
  - `material`: node for the material parameters
    - \* `name`: attribute for the name of the material
    - \* `conductivity`: attribute for the default electrical conductivity value in S/m
    - \* `permittivity`: (optional) attribute for the default relative permittivity value (**required for EQS field problems**)
- `boundary_conditions`: root note for the boundary conditions and their default value
  - `boundary_condition`: note for the boundary condition parameters
    - \* `name`: attribute for the name of the boundary condition
    - \* `type`: attribute for the type of the boundary condition (currently only dirichlet supported in configuration file)
    - \* `boundary`: attribute for the name of the boundary defined at boundaries
    - \* `value`: attribute for the default electric potential value in V
- `frequency`: (optional) frequency value for which field solution should be computed in Hz (**required for EQS field problems**)

**Implementing the Model in Python using the FanPy FieldModel** The model can be implemented in the Python package by inheriting the Field Model with setting the path to the model .xml file under `model_path` and the file name of the .xml file under `MODELNAME`.

```
class VolumeConductorDBSElectrodeWithSTNInMotorRegion(
    AbstractFenicsFieldModel):

    def __init__(self):
        AbstractDBSFieldModel.__init__(self)
        self.MODELNAME = 'dbs_electrode_with_stn_motor'
        self.model_path += 'dbs_electrode_with_stn/'
```

To set all required variables and parameters to run the model, the model has to be initialized with

```
fieldmodel = VolumeConductorDBSElectrodeWithSTNInMotorRegion()
fieldmodel.init()
```

If the model should be run with the default model parameters for the material properties, an empty dictionary can be given as argument.

```
fieldmodel.run(dict())
```

If the material properties should be varied, the dictionary has to define the material properties according their names defined in the .xml model file. For varying the material properties and rerunning the model, the model does not have to be reinitialized.

```
fieldparams = {
    'dielectric_properties': {
        'brain': {'conductivity': 0.3},
        'encapsulation': {'conductivity': 0.15},
        'stn': {'conductivity': 0.5}
    },
}
fieldmodel.run(fieldparams)
```



**Evaluating the Field Solution and Derived Quantities** The computed field solutions can be evaluated at a set of coordinates in the computational domain. The set of coordinates should be in the format of a (n,dim) array, with dim=3 for a 3D field problem. Currently supported field quantities are:

- electric potential: `fieldmodel.get_potential()`
- electric field norm: `fieldmodel.get_electric_field_norm()`
- current density norm: `fieldmodel.get_current_density_norm()`

The field quantity (here the electric potential) is then evaluated with the following line

```
potential = fieldmodel.evaluate_quantity(fieldmodel.get_potential(),
                                         coordinates)
```

with `coordinates` being a numpy (n,dim) array. In addition to field quantities, integral quantities can be computed with:

- Total power: `fieldmodel.get_total_power()`
- Impedance: `fieldmodel.get_impedance()`

**Storing and Loading the Model State** The solution of a fieldmodel can be stored using

```
fieldmodel.store_model_state(save_path, save_name)
```

with `save_path` defining the path and `save_name` defining the name without file-ending. A .hdf5 file containing meta information on the model, such as model parameters and order of the basis functions, and .hdf5 file containing the model data, such as mesh and solution, are created.

The solution of a field model can be loaded using

```
fieldmodel.load_model_state(save_path, save_name)
```

## 2.5 Analytical Validation of the field solution

The example problem consists of determining the electric potential distribution and electric field norm distribution for a layered sphere located in an homogeneous electric field (Fig. 15). Details on solving the stationary current field problem and electro-quasistatic field problem for this geometry can be found in the paper mentioned in the package readme. The geometry parameters for the computational

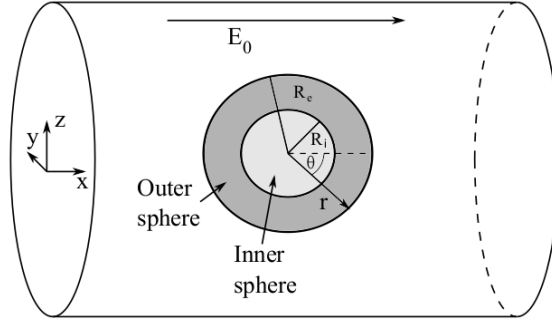


Figure 15: Example geometry of a layered sphere inside a homogeneous electric field.

Table 1: Geometric parameters and electric properties.

\*Only required for electro-quasistatic field problem.

| Compartment  | Property               | Value          |
|--------------|------------------------|----------------|
| Inner Sphere | Radius                 | 5 mm           |
|              | Conductivity           | 2.0 S/m        |
|              | Relative Permittivity* | 120            |
| Outer Sphere | Radius                 | 15 mm          |
|              | Conductivity           | 0.1 S/m        |
|              | Relative Permittivity* | $2 \cdot 10^6$ |
| Bounding Box | Edge length            | 100 mm         |
|              | Conductivity           | 1.0 S/m        |
|              | Relative Permittivity* | 80             |
| Frequency*   |                        | 35 kHz         |

model are set to the following: A potential of 0.5 V and - 0.5 V was applied to the left and right surface of the bounding box, respectively, to generate a homogeneous field of 10 V/m in the bounding box. For solving the stationary current field problem the conductivities in Table 1 are used (for solving the electro-quasistatic field problem the additional model parameters for the relative permittivity and the corresponding frequency from Table 1 are applied). The example problem can be carried out with FanPy by using the `AnalyticFieldModel` class. An example study can be carried out in FanPyStudies using

```
AnalyticValidationModel().run()
AnalyticValidationModel().postprocessing()
```

The model files to be used with the can be used with `AbstractFenicsFieldModel` class can be found in the local files:

- Salome Model Geometry: `./files/analytic_example_model.hdf`
- Exported Model Geometry: `./files/analytic_example_model.brep`
- Gmsh Geo File: `./files/analytic_example_model.geo`
- XML Model File: `./files/analytic_example_model.xml`

**Comparison of the Analytic and Numeric Field Solution** The following image shows a comparison between the analytic and numeric field solution for the example model with the model parameters mentioned above (Fig. 16).

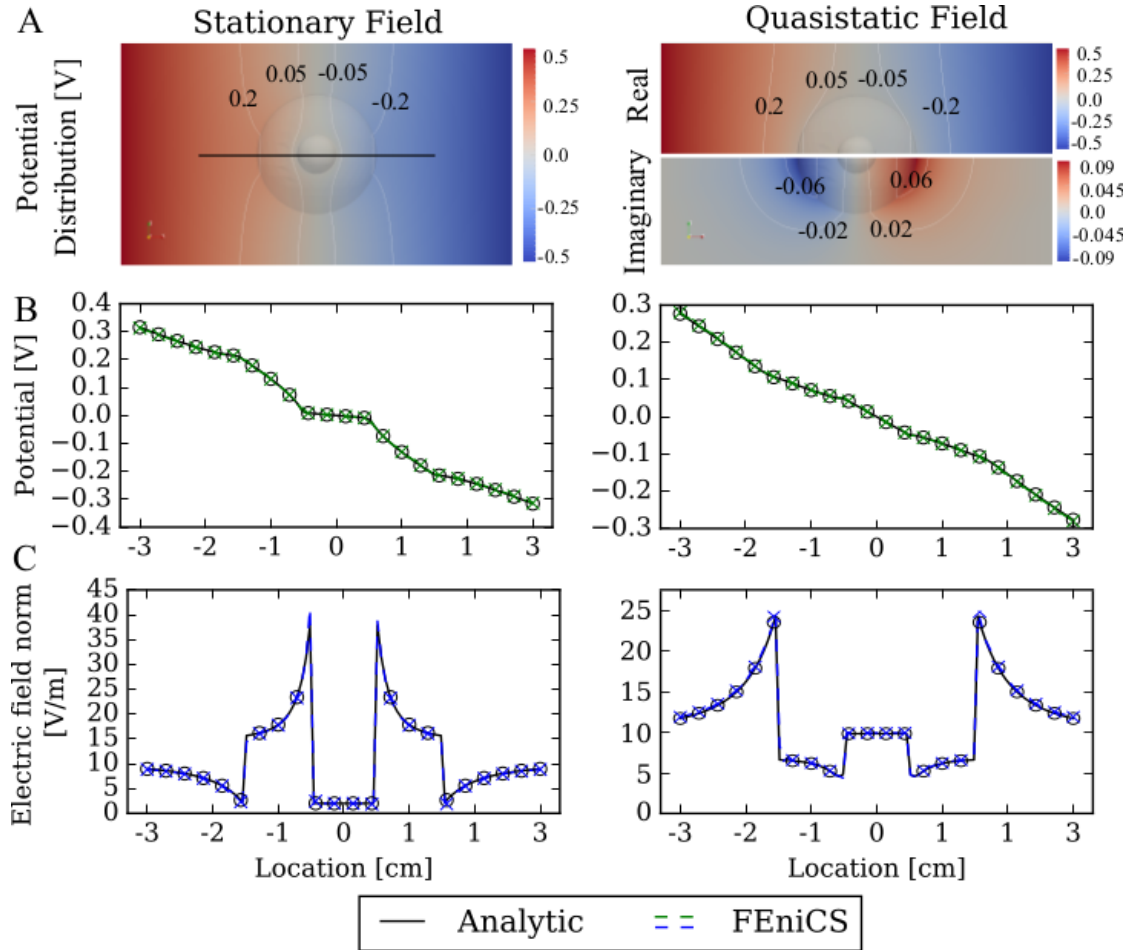


Figure 16: Comparison of the numeric with the analytic field solution.

## 2.6 Computing the neural activation extent during DBS

During deep brain stimulation (DBS), the applied stimulation signal on the electrode contacts of the DBS electrode results in a stimulation of the neuronal tissue in the proximity of the stimulation site. To estimate the extent of neural activation in the target area, the computational approach of determining the activation of perpendicularly oriented axons positioned around the electrode is commonly applied (Volume of Tissue Activated: VTA). For further details on the estimation of the neural activation extent refer to the paper mentioned in the repository description.

**DBS Signal** The extent of neural activation depends on the applied stimulation signal. In DBS for movement disorders, such as Parkinson’s disease, commonly a cathodic (negative stimulation voltage or current with respect to the equilibrium potential), rectangular pulse with a frequency of 130 Hz and pulse duration of 60  $\mu$ s is applied. Common stimulation protocols comprise voltage-controlled and current-controlled stimulation, depending on if the stimulation amplitude during the pulse duration is a constant voltage or a constant current, respectively.

The time-dependent field solution of the field model (see 2.4) resulting from the applied stimulation signal is obtained by multiplying the field solution (e.g., the electric potential) at a coordinate with the signal. While this approach is applicable for stationary field problems and electro-quasistatic field problems with constant frequency, a dispersive field problem would require the convolution of the field solution with the signal by using the Fourier Finite Element Method. The field solution for a monopolar (only one active electrode contact and one ground) current-controlled stimulation signal is obtained in a similar way with the field solution multiplied by the impedance of the model, which is equivalent to scale the unit potential applied to the active electrode contact by the impedance in order to obtain a unit current.

In FanPy a DBS signal can be created by

```
signal_params = {
    DBSignal.FREQUENCY: 130,
    DBSignal.AMPLITUDE: -1e-3, # 1 mA
    DBSignal.DT: 5e-6,
    DBSignal.NUMBER_OF_PULSES: 10,
    DBSignal.PULSE_DURATION: 6e-5}
dbs_signal = DBSignal(signal_params)
```

which represents a train of 10 rectangular current-controlled DBS signals with a frequency of 130 Hz, a stimulation amplitude of 1 mA, a pulse duration of 60  $\mu$ s, and a time step of 5  $\mu$ s.

**Axon Model** The axon model used for determining the activation with respect to an applied time-dependent electric potential is based on the model of a mammalian nerve fiber published in Cameron McIntyre, et al, 2002. The model is implemented in NEURON, which is an open-source software tool for modeling and simulating neurons, neuronal networks, their dynamics, and response to external stimuli.

The model comprises a multi-compartment double cable model of an axonal fiber with 21 nodes of Ranvier and 10 internodal segments resulting in a total of 221 compartments. The time-dependent electric potential along the axon is applied as an extracellular potential to each of the compartments in order to compute the inner potential of the axon model for detecting whether an action potential is elicited.

In FanPy, the computation of the activation of the axon model is carried out with the `Threshold_Volume` class. For a set of axons added to the `Threshold_Volume`, the minimum stimulation amplitude required to elicit an action potential for each axon model is computed. In the following example the threshold-distance relationship for 8 axons with a diameter of  $5.7\text{ }\mu\text{m}$  starting in a distance of 1 mm the active electrode contact center and a spatial stepping of 0.5 mm should be computed. For computing the applied extracellular potential see 2.4.

```

import numpy as np
import matplotlib.pyplot as plt

axons = Threshold_volume()
start_distance = 1e-3
spatial_step = 0.5e-3
ncpus = 1          # number of cpus, that should be used for computation
naxons = 8
for i in xrange(naxons):
    location = Threshold_Location(0,0,i)
    point = Threshold_Point(location)
    transform = point.get_transform()
    coordinate = np.array([0, start_distance+location.get_normal
() * spatial_step, 0])      # axons are positioned on y-axis
    axon = Axon({'diameter': 5.7}) # 5.7 um axon model
    nodes = transform.transform_coords(axon.get_nodes()) #
    transform axon nodes to the defined coordinate
    point.set_coordinate(nodes[0]/2+nodes[-1]/2)
    point.set_axon(axon)
    point.set_dbs_signal(dbs_signal)
    potential = fieldmodel.evaluate_quantity(fieldmodel.
get_potential(), nodes)
    point.set_potential(potential)
    axons.add_point(point)

axons.compute_optimal_activation(ncpus, 1.0)
ct_matrix = axons.get_coord_threshold_matrix()
unit_amplitude = dbs_signal.get_params()[DBSignal.AMPLITUDE]
plt.plot(ct_matrix[:,1],ct_matrix[:,3]*unit_amplitude) # plot y-axis
    of coordinates versus the stimulation amplitude
plt.xlabel('distance_[m]')
plt.ylabel('Stimulation_amplitude_[mA]')
if unit_amplitude < 0:
    plt.gca().invert_yaxis()
plt.show()

```

**Adaptive Estimation of the Neural Activation Extent** Based on the model parameters, such as the dielectric properties of the tissue, the neural activation extent for the same DBS signal may vary, which requires pre-knowledge on how

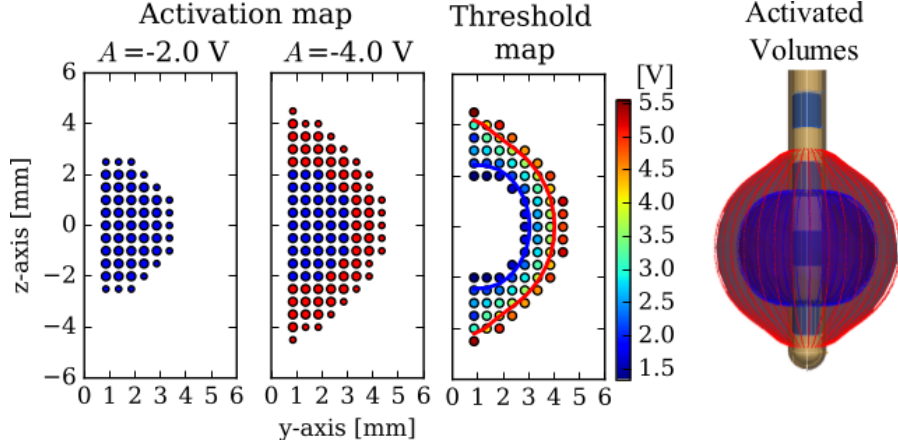


Figure 17: Illustration of the adaptive algorithm to compute the neural activation during DBS. Starting at seed points (e.g. placed in front of the active electrode contact) the algorithm determines whether an axon at this location would be activated by the applied stimulus. If yes, it places new seed points above, below, and right to the current seed point. This procedure continues until a closed zero-activation hull (seed points with no activation) is found. After removal of unneeded seed points to determine threshold isolines in the desired stimulation amplitude range, the minimum required stimulation amplitude to elicit an action potential in each axon model is determined via a bisectioning approach. From this threshold map, the VTA can be determined by deriving the isolines (isosurfaces) for each stimulation amplitude within the given stimulation amplitude range.

many axons have to be distributed in the target area in order to estimate the neural activation extent for a given stimulus. To overcome this pre-knowledge, an adaptive approach is implemented in FanPy, which uses seed points in the target area to estimate the region of activation for a given stimulation amplitude range (Fig. 18). First the activation region for the minimum stimulation amplitude is determined. Starting from the determined axon locations, the activation region for the maximum stimulation amplitude is determined. In order to further save computation time, all points, which are not necessary to determine the neural activation extent within the stimulation amplitude range, are removed. For the remaining axon models the minimum stimulation amplitude required to elicit an action potential is computed, resulting in a threshold map. From this threshold map the neural activation extent is determined by computing isolines for a given stimulation amplitude within the stimulation amplitude range. The neural activation extent can be determined by assuming rotational symmetry, which requires only the estimation of the neural activation extent in one plane, or assuming rotational asymmetry, which computes the neural activation extent in ‘n’ planes around the

stimulation electrode. An example code in FanPy to estimate the neural activation extent for a given stimulation signal, stimulation amplitude range, and field solution with assuming rotational symmetry is given by

```
ncpus = 1          # number of cpus to be used for computation
neuronparams = {
    'g_step': 0.5e-3,          # spatial
    stepping
    'rot_degree_step': 10,    # rotational degree
    stepping
    'dbs_signal': dbs_signal, # dbs signal with
    unit amplitude
    'fiberD': 5.7,            # axon fiber
    diameter
    'stim_min': 2.0,          # minimum
    stimulation threshold
    'stim_max': 4.0,          # maximum
    stimulation threshold
    'symmetric': True         # assume
    rotational symmetry
}
vta = vtautil.compute_vta(ncpus, neuronparams)
```

Often it is of interest which stimulation amplitude range is required to obtain a neural activation extent within a specified distance. For example, in DBS of the subthalamic nucleus (STN), the stimulation amplitude, which activates a certain region in the (STN), but not exceeds a certain distance to reduce side effects, is required. One approach to determine the required stimulation amplitude range is to put axon models at the specified distances and compute their minimum required stimulation amplitude to elicit an action potential. An example for determining the required stimulation amplitude range between a distance of 2 mm and 4 mm to the active electrode contact is given by



```

def create_single_threshold_point(distance, axon, dbs_signal,
    fieldmodel):
    number_of_points = len(axon_min_max.get_points())
    point = Threshold_Point(Threshold_Location(0, 0,
number_of_points)
    transform = point.get_transform()
    transform.set_position_vector(np.array([0, distance, 0]))
    nodes = transform.coord_transform(axon.get_nodes())
    point.set_coordinate(nodes[0]/2+nodes[-1]/2)
    point.set_axon(axon)
    point.set_dbs_signal(dbs_signal)
    potential = fieldmodel.evaluate_quantity(fieldmodel.
get_potential(), nodes)
    point.set_potential(potential)
    return point

ncpus = 1
axon_min_max = Threshold_Volume()
axon_model = Axon({'diameter': 5.7})
point_min = create_single_threshold_point(2e-3, axon_model,
    dbs_signal, fieldmodel)
axon_min_max.add_point(point_min)
point_max = create_single_threshold_point(4e-3, axon_model,
    dbs_signal, fieldmodel)
axon_min_max.add_point(point_max)
axon_min_max.compute_optimal_activation(ncpus, 1.0)
stim_min = axon_min_max.get_point(point_min).get_threshold()
stim_max = axon_min_max.get_point(point_max).get_threshold()

```

**Field Threshold Approximation of the Neural Activation Extent** Following an approach presented by Åström, et al, 2015, field threshold values of field quantities can provide a good approximation the neural activation extent. Nevertheless, using pre-determined field threshold values for a certain axon diameter and field model can result in deviations if the model parameters of the field model are varies. To overcome these deviations, the field treshold values in FanPy are determined by computing the threshold-distance relationship for the corresponding field and axon model and using this relationship to determine the location at which the axon gets activated by a given stimulation amplitude of the applied DBS signal. The following image shows threshold-distance relationship for axons with

varying fiber diameter. In FanPy an example code to use the field approximation

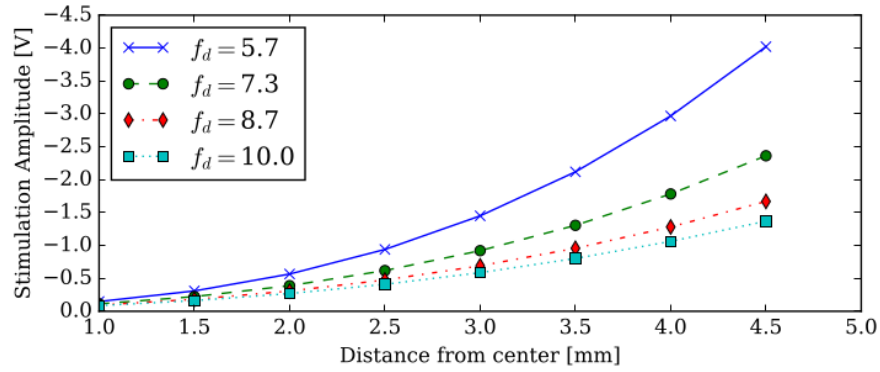


Figure 18: Threshold-distance relationship for axon fibers with varying fiber diameters  $f_d$ .

approach to estimate the neural activation extent by the iso-volume of an electric potential threshold value is given by

```
npcus = 1
neuronparams = {
    'neuronparams': {
        'g_step': g_step,
        'rot_degree_step': rot_degree_step,
        'dbs_signal': dbs_signal,
        'fiberD': fiberD,
        'approx_by_field': True,
        'approx_by_field_quantity': '
potential',
        'stim_min': 2.0,
        'stim_max': 4.0,
        'symmetric': symmetric,
    }
}
vta = vtautil.compute_vta(npcus, neuronparams)
```